Mr. Tool
Autonomous Garage Butler

"Because You're a Tool and Left the Garage Dirty, Again!"

Max Koessick
Final Report
EEL5666C, Intelligent Machine Design Lab
Summer 2003
Professors: Dr. A. Arroyo and
Dr. E. Schwartz
TA's: Uriel Rodriquez, Vinh Trinh, Louis Brandy

Table of Contents

ABSTRACT

Mr. Tool is an autonomous garage cleaner. He is designed to randomly navigate a dark garage at night picking up tools as he finds them. Mr. Tool implements object avoidance, metal detection, object gathering and decision making.

Executive Summary

Mr. Tool is an autonomous vehicle based on a remote control tank platform. Mr. Tool’s objectives are to randomly maneuver around a garage floor while avoiding obstacles and detecting metallic tool. He will then collect them in his basket and move on.

An Atmel ATMega323 is used as the microprocessor. A winch is attached to the back of Mr. Tool It manipulates a carbon fiber arm that has an electromagnet attached. Pulse width modulated (PWM) servos control speed and direction. Also, PWM controls the speed of the winch.

Obstacle avoidance is accomplished with two main sensors, sonar and infrared. The sonar is mounted on a servo for 180° field of view. This is the most critical sensors in obstacle avoidance. IR is rearward looking.

Tool detection is accomplished by a Hall-effect gear tooth sensor. It is located in the lower front apex of a vee-shaped trough. Mr. Tool ‘stumbles’ on his targets and locates them underneath the magnet by pushing them.

Introduction

Mr. Tool was an idea born out of frustration. After many a long day in the garage, the last thing one wants is to clean up. Introducing Mr. Tool, he will pick up your tools for you.

This report will detail all of Mr. Tool's components. It will also document the build and testing processes. First, the platform and drivetrain will be discussed. Next, the arm subsystem will be tackled. Finally, the electronic subsystems will be revealed.

The appendices contain all source code as well as behavioral flowcharts. Also included are circuit schematics. Lastly, two special reports detailing the operation of the sonar array and the metal sensing hall-effect sensor are presented.

Integrated System

Mechanical Overview—*Platform Fabrication*

Main Body Construction/Integration

The overall platform consists of two major subsections. First, the lower half is the cannibalized bottom of the remote control tank. This consists of the gearbox, motor, suspension and lower tub.

The gearbox is a stout dual clutch design powered by a Marubuchi RS-540S racing motor that draws 2.2A at stall and is powered by a 7.2V 3000 mAh NiMH battery. The suspension consists of 18 wheels, 14 of which are independently suspended using a mini-torsion bar system. Of the remaining four wheels, two are the main drive sprockets and two are used to keep tension on the tracks. These four do not move. The overall concept of the lower half remains virtually unchanged from the original R/C tank with the exception of mounting brackets for servos and the hall sensor. Figure 1 details the lower tub, including dual clutches, gearbox, motor, speed controller and torsion bars. Figure 2 shows typical suspension deflection.

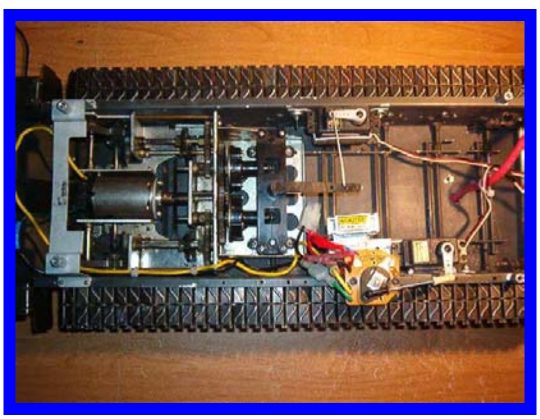


Figure 1. Lower Tub and Drive Mechanism

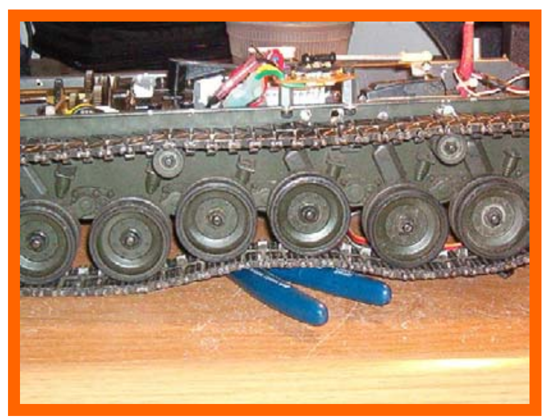


Figure 2. Suspension Deflection

The upper body houses the microcontroller (µC) development board as well as the 3 daughter boards. The top, with the exception of the microcontroller housing, was fabricated in the Mechanical Engineering machine shop from sheet aluminum. The side skirts are bolted on using standard 6-32 socket head screws. This detail is shown in Figure 3. The rear skirt is a floating design. Moreover, it is suspended on springs. Figure 4 illustrates the suspended aft bumper. Originally, a front floating skirt was employed, but removed in the final stages. It was non-functional as it is the sonar's responsibility for front object avoidance.

The upper body is attached to the lower via a four thumbscrews and a main electrical trunk.

Figure 3. Upper Body Detail

Figure 4. Aerial View of Floating Rear Skirt

Arm Systems

Arm The arm is almost composed entirely of lightweight carbon fiber composite. It is 1/2 inch in diameter. It is boxed together with ¼ inch threaded rod (6-32 pitch). Moreover, the rod serves to sandwich the carbon fiber together. The All Thread rod is secured with both socket head set screws as well as nuts. In order to smooth 90° transitions, the carbon fiber tube ends were coped. Figure 5 shows the set screws and nuts as well as the coping detail.

Figure 6 details the 5/8 inch nylon spacers that are used to 1) determine appropriate box diameter of the arm as well as 2) reduce friction between the arm and the body. These spacers were turned on a Hardinge lathe from 1" nylon stock.

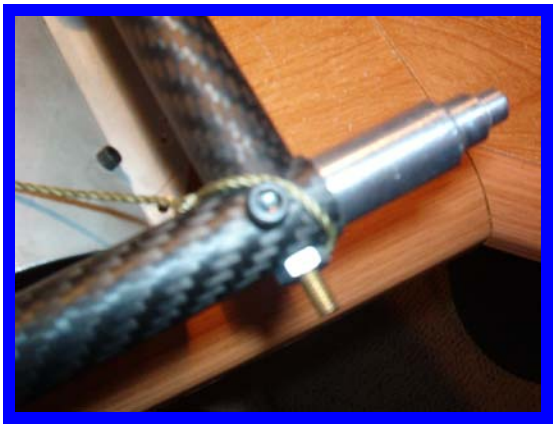


Figure 5. Front Arm Joint with Coping Detail, Set Screws, and Threaded Rod

Figure 6. Rear Arm Detail with Nylon Spacer

Arm travel is determined by stop switches located on the body at both extremes of travel. At the raised limit, the stop switch also incorporates a leaf type spring to push the arm down to the lower rest position. More information will be discussed later.

Winch The winch motor is a commercially available kit made by Tamiya Model Company. It is a planetary gear drive system that uses a 3V DC motor that spins at 18000 rpm. Motor actuation is controlled through a National Semiconductor LM18200 H-Bridge integrated circuit that is discussed later. The shaft energy is then reduced through a set of four planetary gears to a final drive ratio of 400:1. The output shaft is coupled to a take up spool via a standard servo horn. A bracket is wrapped around the spool and bolted to the upper body. The support bracket's purpose is to counter the upward force on the output shaft caused by the pulling cable. Lastly, the winch cable is fed though an elevated guide to provide a proper fulcrum to facilitate lift.

The manufacturer boasts a lifting capacity of 15Kg with the 400:1 drive ratio. This specification far exceeds the need as the target lift will be under ½ pound.

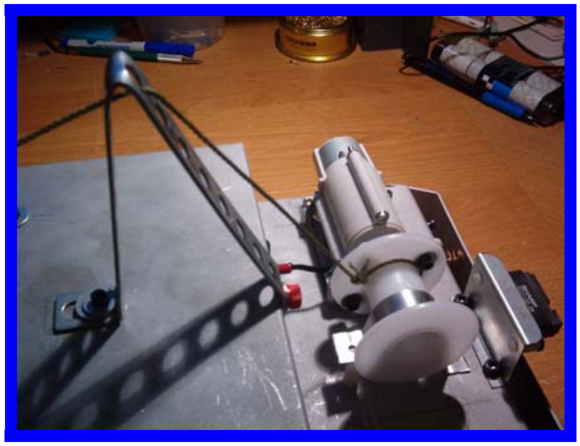


Figure 7. Planetary Gear Winch, Cable Guide and Bracket Detail

Electromagnet Solenoid City's E-20-100 electromagnet ($32.50) is the second of the two lifting workhorses. When a positive target is identified, the microcontroller activates it via field effect transistor (Fairchild Semiconductors HUF76107P3 Power FET, discussed later). The electromagnet then stays energized through the entire cycle finally de-energizing at the apex of the lift.

From Graph 1, Typical Hold Force vs. Input Power (located in Appendix C), hold force is greater than the minimum of 18 pounds. Again, this specification far exceeds the needed ½ pound coupled with any gravitational effects.

It is attached to the lift arm by a floating collar. This way, the magnet is free to rotate and remain parallel to the ground. The attaching collar was machined on the Hardinge lathe from one inch aluminum circular stock. The magnet assembly is retained by two set screws on either side that prevent lateral movement while the electrical wiring is routed inside the carbon fiber arm for protection.

More information is available in Appendix E, "Special Sensor Report, Solenoid City's E-20-100 Electromagnet."

Figure 8. Magnet Mount, Collar and Wire Route

Peripherals

LCD Display The LCD display is a standard 2 line by 16 character dot display that uses the standard ASCII set. It is a parallel (8 data bus lines) type display. It uses the industry standard Hitachi HD44780 LCD controller.

The original intent of the LCD was to display the range to the closest target. Unfortunately, time was short and the end result is that it displays the robot's name and other curt information. The ASCII to hex conversion was just too time invasive.

LED Display Mr. Tool has a 'Knight Rider' style bar of LEDs that is for display. The circuit board was constructed by hand on a protoboard. All of the traces were fabricated from spent resistor leads.

The circuit is active low, i.e. the anode is tied to a port through a current limiting resistor and the cathode is applied to 5V.

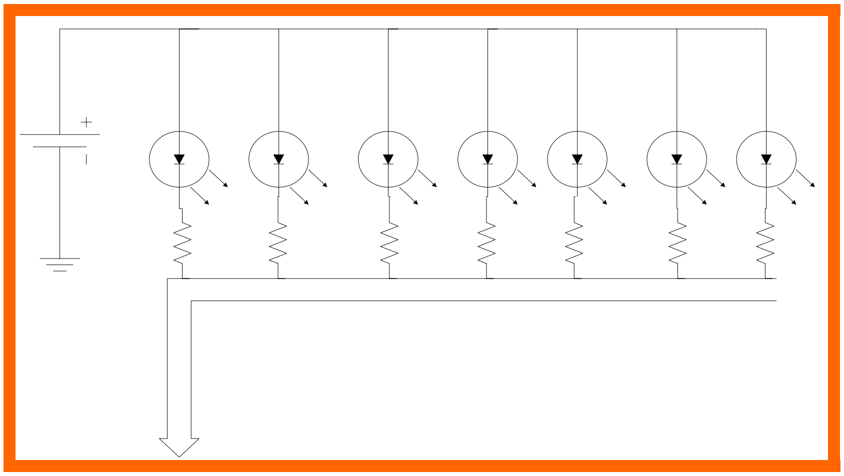


Illustration 1. LED Schematic

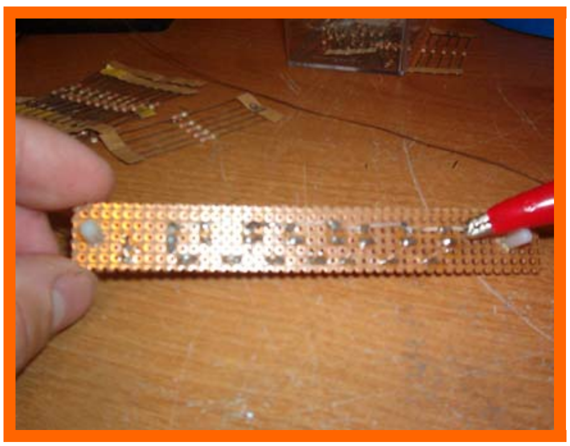


Figure 9. LED Protoboard

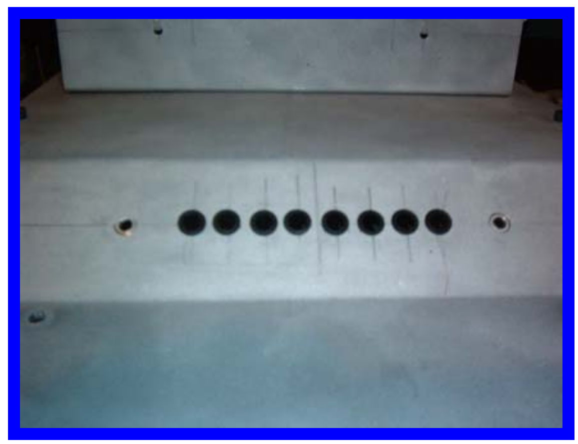


Figure 10. Mounted LED Arra

Mechanical Overview—*Drive Platform*

Speed Actuation The main drive motor is controlled via a mechanical switch and servo combination. There are 2 speeds in forward and reverse as well as a neutral (stop) position.

The servo requires a 1-2ms pulse every 10ms to determine position. For example, a 1ms pulse produces a full right position and a 2ms produces a full left position. A pulse width modulation (PWM) output was used from the microcontroller to generate the requisite periods. Exactly, proper pulse widths had to be determined to move the servo to the exact position for the desired speed.

To generate the PWM, the output compare (OC) feature of the µC was utilized. As background, the OC is nothing more than an 8 bit counter that counts up to 255 and back down again. With a known clock speed, the PWM is generated by storing a number that the OC looks for. When this number is spotted, the OC toggles an output pin. This is repeated on the down count, again toggling the pin.

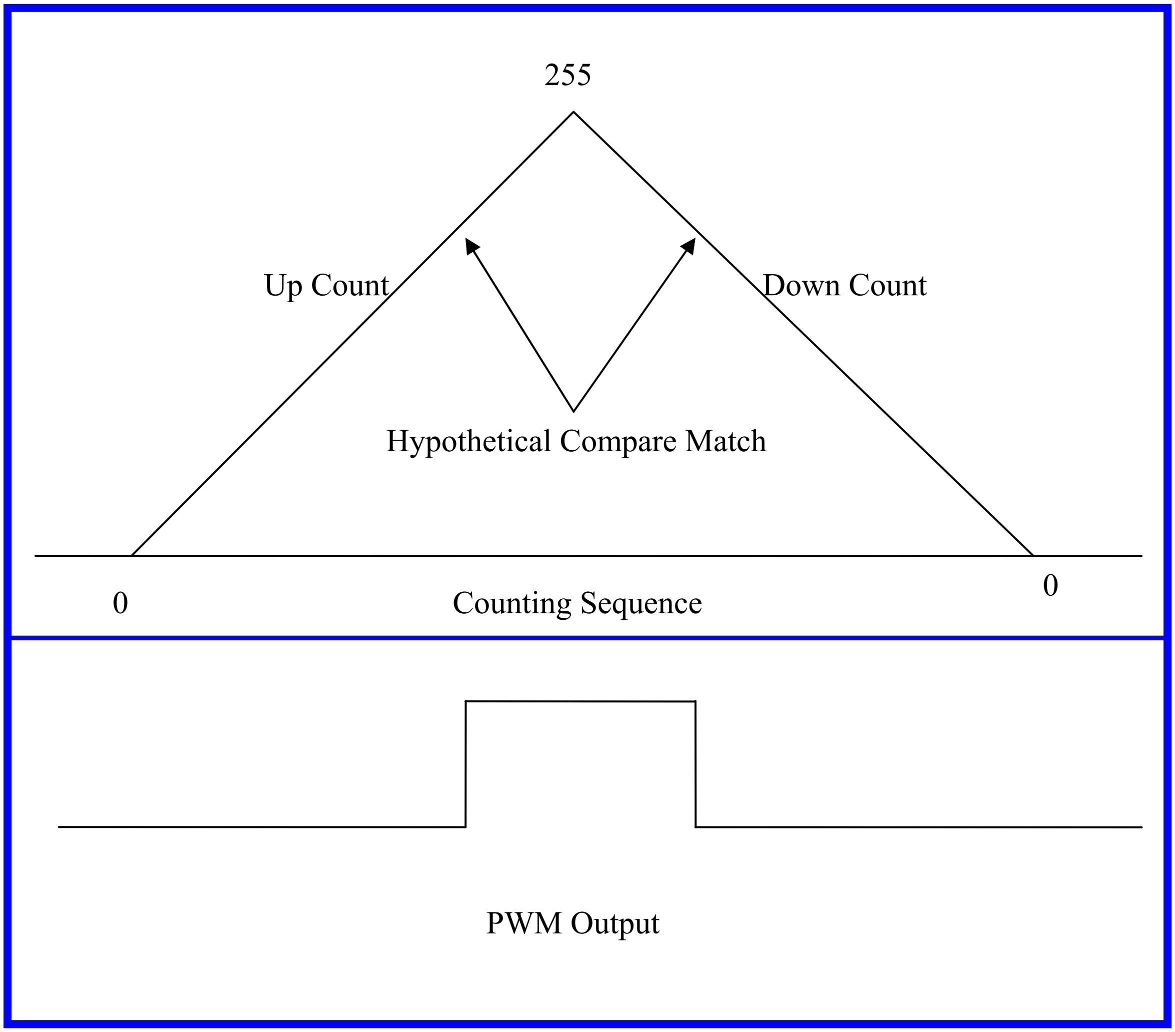


Illustration 2. PWM Basics

Speed	OC Value (hex)	Direction	OC Value (hex)	Sonar Direction	OC Value (hex)
Fwd Fast	$C9	Full Right	$A1	Look Right	$F6
Fwd Slow	$CB	Slip Right	$A3	Straight	$D9
Neutral	$CE	Straight	$A7	Look Left	$BF
Rev Slow	$D2	Slip Left	$AF		
Rev Fast	$D5	Full Right	$B2		

Table 1. Output Compare Match Values

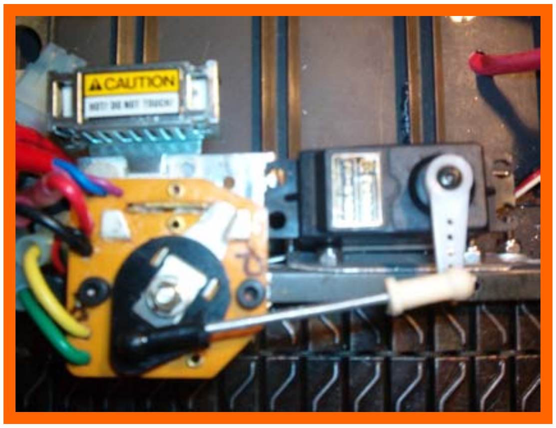


Figure 11. Speed Controller and Servo

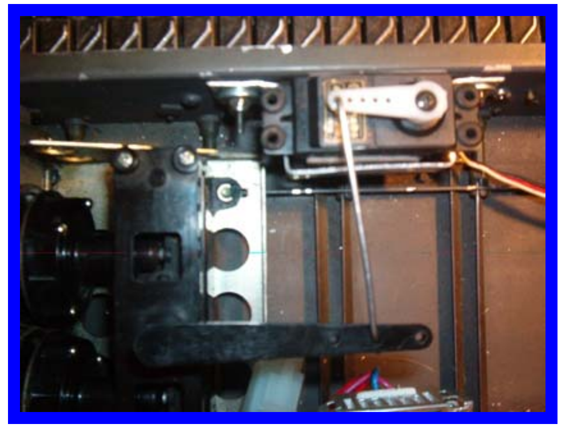


Figure 12. Dual Clutch (Direction Control) and Servo

Direction Actuation Directional control is actuated in much the same way. A servo controls a dual clutch set, one for each track. As pressure is applied by the servo on the lever arm, the clutch on that side is engage or allowed to slip. The resulting action is either full track stop on that side or reduced power. The end result is one of two turn styles: full pivot or gradual slip. The latter is more graceful.

Servo Note The servos were originally mounted to the lower tub using double sided adhesive tape. A great deal of play was introduced into the push-pull system by the flexibility of the tape. Further, no precise servo movement was attained. While acceptable n a remote control situation where human feedback is present, the servo was not providing consistent movement. The solution involved fabricating aluminum brackets to secure the servos to the lower tub walls. All play was eliminated. These brackets are evident in both Figures 11 and 12.

Mechanical Overview—*Power Supply*

Electrical power is supplied to Mr. Tool through 3 main nickel metal hydride (NiMH) battery packs. Three individual packs were used to reduce potential noise caused by the motors and motor drivers.

One, the µC pack, is composed of 12 1.2V 1800 milli-Amp hour (mAh) AA cells. Theoretical voltage is 12*1.2 or 14.4 volts. However, the battery pack is consistently above 16V, unloaded.

The second battery pack is a 7.2V, 3000 mAh remote control car pack. This pack is the main battery for the drive system only. Electrically, the motor and drive system are disconnected from the all other electronics.

Last, a 6V, 1800 mAh battery is used to provide sole current for the electromagnet. Typically, electromagnets demand high current. By incorporating its own power supply, the electromagnet will not drain current from the microcontroller and thereby possibly causing faults.

Daventech SRF08 Sonar

The Daventech SRF08 ultrasonic range finder (sonar array) uses a pulse ('ping') of sound to determine the range of up to 17 targets in an area. The SRF08 emits a ping and then waits for the first echo to return. This process takes approximately 65ms to complete.

The sonar array communicates with the host microprocessor via the Inter Integrated Circuit Bus (I2C) developed by Phillips for communicating within consumer electronics. Atmel uses this standard in the form of the Two Wire Interface (TWI).

The SRF08's main purpose in the world of Mr. Tool is obstacle avoidance from forward, left and right directions.

More detailed information and pictures are in the abbreviated Daventech Special Sensor Report located in Appendix D.

Cherry GS100701 Gear Tooth Sensor (Hall)

The GS100701's primary purpose is high speed gear sensing. Normal applications include automotive applications and machinery speed sensing. However, this hall type sensor can also be used to detect metal objects that are within close proximity to the head. In Mr. Tool, it is used to accept/reject ferrous targets.

This model is a sinking interface, i.e. it produces negative logic.

The sensor contains internal integrated circuitry that is basically an open collector bipolar junction transistor (BJT). The BJT supplies ground on the signal output wire when a ferrous (gear) target is sensed. The only external circuitry that is needed is a pull-up resistor that is determined by input voltage. The GS100701 can operate on voltages from 5 to 24 VDC.

Testing is as simple as placing a metal object in front of the sensor. A multimeter reveals that the voltage drops from 5V to approximately 0V with detection. Interfacing proves just as simplistic. The single output wire is connected to an external interrupt on the µC that is configured for falling edge trigger. The sample code "16 Bit PWM and External IRQ.asm" was used to test functionality.

More information is contained in Appendix F.

Sharp GP2D12 IR Sensor

The Sharp Electronics GP2D12 Analog IR sensor is used to detect rear obstacles. Normally, the detecting distance is between 10 and 80 cm. Mr. Tool was originally configured around a GP2D15 digital output sensor that gives logic one at a fixed detection distance of 24 cm. Unfortunately, the GP2D15 met an untimely demise due to reverse battery application. The analog version was readily available (in lieu of 'Next Day Air' charges).

A conversion was devised to change the output to a digital one so that no platform revision were needed (discussed later). Succinctly, the digital output conversion uses an LM311 comparator to compare against an output reference voltage from a set distance. Approximately 24 inches was chosen for convenience, corresponding to a voltage of 2.04V. Table 2 shows the results of near field testing. Figure 12 shows the mounted sensor.

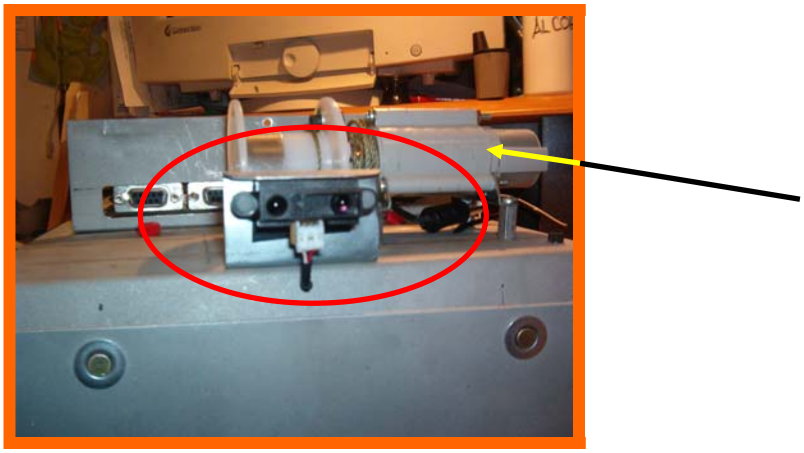


Figure 13. Sharp GP2D12 Sensor (Circle) Mount with Winch Assembly in Background (Arrow)

Arm Feedback

The first attempts at arm control involved many attempts to time the lift cycle. This proved unworthy due to the winch spool. Moreover, the exact length of the string would have to be precisely measured, as well as having a known spool speed. From there, the distance travel would be factored in . . . there are much better ways to do this.

Instead, limit switches were used. In fact, two switches were attached to the skirts. One is at full rest and the other lies at full upright. Each is tied directly to a port pin through a current limiting resistor and then to ground. Both switches are of the normally closed type. The Atmel's internal pull-ups are enabled to pull the output high when the switch is open.

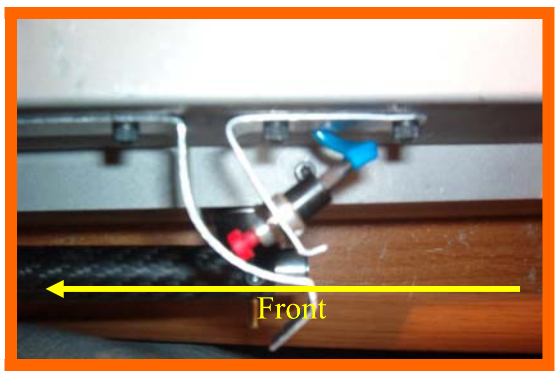


Figure 14. Upper Arm Limit Switch and Return Spring

A leaf type return spring hand rolled from aluminum is used to coax the spring back towards the rest position once the tension on the winch has been released.

Electrical and Computing Overview

Atmel ATMega323 Microcontroller

The ATMega 323 was actually the second choice for a microcontroller. The first choice was the ATMeg1a 128, however, due to technical difficulties; design was switched to the 323.

The 323 is more adequate in terms of ports and timers. Features present on the board that were utilized include the 4 timers in 8 bit PWM mode, all available external interrupts and the two wire interface or I2C bus.

Software development was on the proprietary Atmel board, the STK500. Originally, the STK501 top module with 64 pin zero insertion force (ZIF) socket was used, but it developed some problems. The STK500 is also the same board that is incorporated into Mr. Tool.

Great care was taken in the routing and termination of all wiring. Early on in development, faults and frayed wires were discovered near the shear junction of wire to connector (i.e. solder point). To remedy, heat shrink tubing (22AWG) was used as a strain relief. The result is shown in Figure 15 below. Note the absence of the typical 'bird's nest.'

Figure 15. Precision Wiring Harness and STK500

Motor Drivers

Experiments were performed on two discrete integrated circuit packages. Ideally, PWM was desired to control all electrical motors inside Mr. Tool. However, due to the high current draw of the main motor, no suitable motor driver was found for the main motor. In contrast, two drivers were tested in conjunction with the winch motor.

Texas Instruments SN754410 H-Bridge Originally, the TI H-bridge was chosen to control the winch motor. It was thought that the 1.1A capacity of this package was adequate for the motor. However, after extensive testing, the winch motor revealed a stall current of close to 1A. Although the SN754410 is rated to 1.1A, it never performed near that level. It seemed to deliver closer to .85 to .95A under load all the while generated copious amounts of heat. Also, this IC is only available in a PDIP with no included sink to alleviate heat.

National LMD18200 H-Bridge A much more robust package, the LMD18200 is available with a current capacity of 3A and is encased in a TO-220 type with included heatsink. It was tested on both the main motor and the winch motor. While it performed flawlessly on the winch motor, the LMD18200 could not keep up with the main motor and would 'thermal out,' or go into thermal protection mode due to the large amount of current demand.

The National H-bridge included many extra features not available on the Texas Instruments controller. Notably, it includes provisions for an external heatsink, single direction control pin (as opposed to two on the TI), and braking capability. First, an aluminum TO-220 style heatsink was bolted to the back with thermal grease in between the two. Next, braking was introduced by connecting the brake input to an unused port pin on the microcontroller. Use of the brake allowed for even transitions between lift and descent of the arm. The only precaution is that there must be a 1μS delay in between application of the direction pins or brake pins.

Magnet Control—Fairchild HUF76107 Power FET

Erik Sjolander's 'Butler Bot' provided the solution for the control of the electromagnet. A TTL switch was needed to activate the magnet that could handle the high current. Enter the Fairchild HUF76107 field effect transistor. Part of the *UltraFET* series, the '76107 offers a 20A, 30V capacity with 200nS switching time. The FET is directly tied to a port pin on the microcontroller and is active high. The only external circuitry is a pull down resistor to guarantee the state of the transistor in a floating input situation.

Daughter Boards

There are three daughter boards that reside underneath the upper body. The main board serves as a junction point to the entire lower circuitry such as the servos, IR, sonar, Hall, etc. It was design in Protel and milled on the IMDL T-tech CAM router. Both the motor driver and IR digital conversion board were hand made with protoboard readily available from Radio Shack.

Main Daughter Board—*Circuit Brief* The main daughter board supplies 5V regulated power to the servos, sonar, hall, and LEDs by means of a National LM1085 (3A 5V regulator). Also included are the switch inputs for both front and rear bump and arm limit switches. The port pins are directly protected by the use of in line 150Ω resistors. Pull is selectable up or down through a jumper.

Originally, the TI motor driver was to be located on this board, but motor driver was relocated off board due to router schedule time constraints (there was not enough time to route a new board). Also, this board derives its power from the microcontroller battery back with voltage inserted to separately power the magnet.

Input supply is bypassed by way of a 100μF electrolytic capacitor. Output is stabilized via a 10μF Tantalum capacitor.

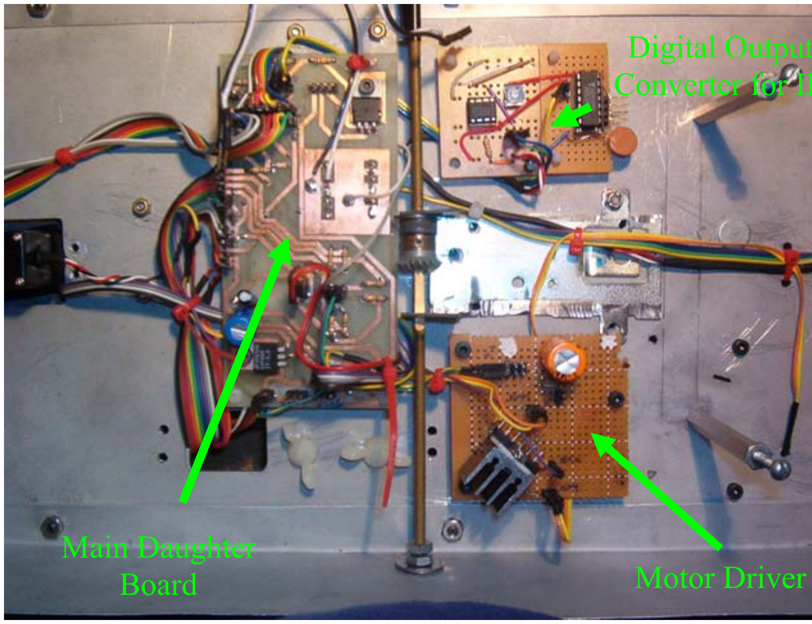


Figure 16. Daughter Boards

Motor Driver Board—*Circuit Brief* The motor driver board consist of two main parts, the LMD18200 H-Bridge and a 470µF bypass capacitor. Male headers are used as attachment points for the wire harness. A large aluminum TO-220 heatsink is attached to dissipate heat. Again, the board was constructed on protoboard and hand routed with discarded resistor leads.

GP2D12 Digital Output Conversion—*Circuit Brief* To review, the analog output of the GP2D12 was modified to put out a logic 1 at a predetermined distance. Normally, the IR sensor outputs a voltage between roughly 0 and 3V according to the distance of an object. A fixed distance was chosen and this voltage recorded and input into a comparator. The comparator weighs this input against a reference voltage and then turns on (logic one). The reference voltage can be adjusted through a 10kΩ potentiometer to represent a distance of approximately 4 to 35 inches. Mr. Tools stops if an object is closer than approximately 24 inches. A 1µF electrolytic capacitor was added between signal and ground to help reduce noise. Also, a .1µF capacitor was added to bypass the supply voltage. Board power is taken from the main daughter board.

Software

Atmel's AVR Assembly was the programming language of choice. It was chosen because of speed and ease in programming. For example, one does not have to mediate through a third party compiler such as Win*AVR*, etc.

Behaviors

Behaviors implemented include 360° obstacle avoidance through the use of pivoting sonar and IR. Also implemented are metal detection and target acquisition through the use of the Hall-effect sensor. The last behavior was arm feedback to positively control arm movement

Component Sources

1. Bump switches, LEDs, protoboard, heatsinks, batteries + chargers ***Radio Shack***
2. Electromagnet, ***www.solenoidcity.com***, $32.50
3. TI SN754410 H-Bridge, ***www.ti.com***, free sample
4. Fairchild HUF76107 Power FET, ***www.fairchildsemi.com***, free sample
5. LMD18200 H-Bridge, ***www.national.com***, free sample
6. Sharp GP2D15, GP2D12, $15 and $12 respectively, ***www.hobbyengineering.com***
7. Atmel STK500 with ATMega 32 and STK501 with ATMega128, ***www.digikey.com***, $158
8. Tamiya Planetary Gear, ***www.towerhobbies.com***, $12
9. Flakpanzer Gepard R/C tank, bought in Middle School, original price $300 (including servos)
10. Aluminum flat stock, courtesy SAE, free
11. Carbon fiber tube, courtesy SAE, free
12. Daventech SRF08 Sonar and mounting bracket, ***www.acroname.com***, $70
13. Spare RS-540S Motors, ***www.allelectronics.com***, $10
14. Hall sensor, GS100701, ***www.cherrycorp.com***, free sample

CONCLUSION

Mr. Tool was very time invasive. Most of the goals set forth at the beginning of the semester were implemented (see Behaviors above). The only goal no implemented was positive target grasp. Further, a goal was set to include a sensor that acknowledges that the magnet has the tool. This was not accomplished. All other behaviors were implemented successfully.

The unmet goal above constitutes an area of improvement. Another area would be the PWM control of the main motor. A possibility was found as a Motorola H-bridge capable of sinking or sourcing up to 5A, but there was not enough time to order samples or test. Issues that would have been dealt with include heat and increased power supply. A PWM controlled main motor would have given more precise speed control. Also, as all timer channels on the ATMega 323 were used, a larger microprocessor would have been needed with additional timer channels.

Warnings for future students would include early testing of a completed system. Mr. Tool's first full system test was days before the final demonstration. A mysterious bug prevented movement on demo day. Start early!! Also, students should make full use of many of the sample programs that may semiconductor manufactures have.

Future work would include the stouter H-Bridge for the motor and adding target acquisition acknowledgement. Also, a means to judge the size of the tool should be added. Lastly, the arm should be tool height independent. Currently, tools with a height of approximately 1.5" only are readily picked up.

Source Code

```
;-----------------------------------------------------------------
; Name:                   Main Robot.asm
; Description:    ATMega1323 Two Wire Interface (IC2) Test Program
;                         Interfaces Daventech SRF08 Range Finder to I2C
Bus

; Author:         Max Koessick
; Class:          EEL5666C, Intelligent Machine Design Lab
; Date:                   July 27, 2003
; Revision        1.a
; Changes to Date:
;                         7/27/03 First Revision

;-----------------------------------------------------------------

.nolist                                                 ; Do not include
in .lst file
.include "m323def.inc"                      ; Standard ATMega128 Include
File
.include "TWI.inc"                                ; Two Wire Interface
Error code definitions
.list
; Interrupt service vectors

.org $0000
      rjmp Reset                                  ; Reset vector
.org INT0addr
      rjmp IntV0
.org INT1addr
      rjmp IntV1
.org INT2addr
      rjmp IntV2

; ***************************************** ---------------------
; ***** Register defines for main loop ***** ---------------------
; ***************************************** ---------------------


.def        mpr              =r16        ; defines multipurpose
register
.def        MPR2        =r17        ; multipurpose register 2
.def        mpr3        =r18
.def        ECHO1L           =r19
.def        LEDreg           =r20
.def        ErrorReg    =r21
.def        Obsreg           =r22        ; Contains the object
detected flag

; ****************** ---------------------------------------------
; ***** Equates ***** ---------------------------------------------
; ****************** ---------------------------------------------
```

```
; Equate statements for SRF08 Sonar
.equ        W                  = 0               ; Write Bit
.equ        R                  = 1               ; Read Bit
.equ        SLA                = $FE       ; Slave Address of SRF08
.equ        CommandReg  = $00       ; Random address
.equ        Inches             = $50       ; Ranging Mode returns
results in inches
.equ        EchoReg2    = $02
.equ        EchoReg3    = $03

; Equate statements for Servos
.equ        LookRT             =$F6        ; Sonar directions
.equ        LookFwd            =$D9
.equ        LookLFT            =$BF
.equ        FullLFT            =$B2        ; Turning
.equ        SlipLFT            =$AF
.equ        Straight    =$A7
.equ        SlipRT             =$A3
.equ        FullRT             =$A1
.equ        StopPWM            =$FF
.equ        FwdSlow            =$d5
.equ        FwdFast            =$d5
.equ        Stop        =$CE
.equ        RevSlow            =$cb
.equ        RevFast            =$c9
.equ        Turntime    =$FFFF             ; Turning Delay
.equ        Revtime            =$FFFF            ; Reverse Delay
.equ        NoPing             =$FFFF            ; Wait for Servo to
turn
.equ        MinDist            =20


.equ        brake       = 1
.equ        ArmDir             = 0
.equ        MagOn       = 6

; *********************** ---------------------------------------
; ***** Reset Vector ***** ---------------------------------------
; *********************** ---------------------------------------

Reset:
;----------------------------------------------------------------
; ***** Setting Stackpointer ***** ------------------------------
;----------------------------------------------------------------
      ldi         MPR,low(RAMEND)               ; Set stackptr to ram
end
      out         SPL,MPR
      ldi   MPR, high(RAMEND)
      out   SPH, MPR
;----------------------------------------------------------------
; ***** Set Port Directions ***** -------------------------------
;----------------------------------------------------------------
      ser         mpr                           ; Set TEMP to $FF to...

      out   DDRA,mpr                ; LCD
```

```
      ldi         mpr,0b11111000
      out         DDRB,mpr                ; Set PORTB to output
      ldi         mpr,(1<<PB0)|(1<<PB1)
      out         PORTB,mpr               ; Enable Internal pull up for
PB0,PB1

      ser         mpr                           ; LEDs and TWI
      out         DDRC,mpr
      out         PORTC,mpr

      ldi         mpr,0b11110011          ; Set PD2 and PD3 to input
      out         DDRD,mpr                ; Set PORTD to output


;-----------------------------------------------------------------
; ***** Initialize I2C(TWI) Interface ***** ----------------------
;-----------------------------------------------------------------

; Set TWIBitRate for fclk=16Mhz

      ldi         mpr,11                        ;
100Khz=3.69MHz/(16+2*11) See Datasheet Pg202
      out         TWBR,mpr                ; Note: This system clock
does not support 400kHz


; Initialize TWCR Register
      clr         mpr
      ldi   MPR,(1<<TWEN);
      out         TWCR,MPR                ; Initialize Two Wire Control
Register

;     ldi         mpr,$01
;     out         TWAR,mpr
;-----------------------------------------------------------------
; ***** Initialize TC0,TC1A,TC1B,TC2 ***** -----------------------
;-----------------------------------------------------------------

      clr         mpr
      out         TIMSK,mpr               ; Turn Off any Timer
associated interupts


;-----Enable 16Bit PWM (Sonar Servo) Counter in 8Bit Mode---------


      ldi         mpr,0b11000001          ; Bit7:6 -> Inverted PWM
                                                ; Bit5:4 -> Disable
OC1B
                                                ; Bit3;2 -> FOC =n/a
                                                ; Bit1:0 -> 8Bit PWM
mode
      out         TCCR1A,mpr

      ldi         mpr,0b00000011          ; Bit7 -> Input Noise
Canceler Disabled
```

```
                                                ; Bit6 -> Input Capter
Edge Select n/a
                                                ; Bit5:4 -> Unsused
                                                ; Bit3 -> Clear on
Compare Match Disabled
                                                ; Bit2:0 -> Prescale =
/64
      out         TCCR1B,mpr


      sbi         PORTD,Brake             ; Set Brake bit to low PD0=0

;-----Enable 8 bit PWM (Dir and Speed) ---------------------------

      ldi         mpr,0b01110011          ; Bit7 -> FOC2 force Output
Compare = n/a
                                                ; Bit6 -> PWM0 Enables
PWM output
                                                ; Bit5:4 -> Set on
match upcount, clear on match downcount (11)
                                                ; Bit3 -> CTC0 No clear
on match
                                                ; Bit2:0 -> Prescale =
/64
      out         TCCR0,mpr               ; Enable PWM0
      out         TCCR2,mpr               ; Enable PWM2

;-----------------------------------------------------------------
; ***** Enable External Interrupts ***** -------------------------
;-----------------------------------------------------------------

      in          mpr,MCUCSR
      andi  mpr,0b10111111          ; Clear the INT2 Sense Control Bit
-> Falling Edge triggered
      out         MCUCSR,mpr

      in          mpr,MCUCR
      andi  mpr,$f0                       ; Mask Upper Bits
      ori         mpr,0b00001010          ; Set ISC1:0 Sense Control
bits [3:0] -> Falling Edge for Int0
                                                ; Low level for Int1
(IR) -> ISR must fire as long as a
                                                ; bject is detected in
the rear.
      out         MCUCR,mpr

      ldi         mpr,0b11100000          ; Enable Interrupts 1-3
      out         GICR,mpr

;-----------------------------------------------------------------

; *********************** ---------------------------------------
; ***** Main Program ***** ---------------------------------------
; *********************** ---------------------------------------
mainloop:
```

```
      ldi         mpr,FwdSlow                   ; Set default forward
speed
      out         OCR0,mpr
      ldi         mpr,Straight                  ; Set default direction
      out         OCR2,mpr
      ldi         mpr,LookFWD                   ; Set default Sonar
Direction
      out         OCR1AL,mpr

      sei

      call  LEDs                          ; Update LEDs

      call  Look                          ; For Debug
;
      sbrc  Obsreg,0                      ; If bit one is cleared from
LOOK subroutine,
                                                      ; then no
obstacle found.  Prgm will skip calling
                                                      ; Obstacle
routine
      call  Obstacle

      rjmp  mainloop




; *********************** ----------------------------------------
; ***** Subroutines ***** ----------------------------------------
; *********************** ----------------------------------------

;-----------------------------------------------------------------
;-----Look--------------------------------------------------------

Look:
; Start Error Rejection: Call ping 3 times to verify that an object is
in path
;     before branching to obstcle routing
;Ping1:

      call  Get_PING                      ; Get sonar data
      subi  ECHO1L,MinDist                ; object closer than MinDist
inches?
      brsh  No_Obs                              ; ...no? Then branch is
same or higher
;Ping2:
;     call  Get_PING                      ; Get sonar data
;     subi  ECHO1L,MinDist                ; object closer than MinDist
inches?
;     brsh  No_Obs                              ; ...no? Then branch is
same or higher
;Ping3:
;     call  Get_PING                      ; Get sonar data
```

```
;      subi  ECHO1L,MinDist                ; object closer than MinDist
inches?
;      brsh  No_Obs                              ; ...no? Then branch is
same or higher

       ldi         Obsreg,$1                     ; Found an Obstacle
       rjmp  End_look

No_Obs:
       clr         Obsreg                              ; Didn't find an
obstacle

End_look:

       ret


; -----Get_PING-----------------------------------------------------

Get_Ping:
.nolist
.include "ping.inc"
.list
;return instruction included in .inc file

;-----------------------------------------------------------------
;-----Obstacle----------------------------------------------------

Obstacle:



       cli                                       ; Disable interrupts
whil changing Output compare registers

       ldi         mpr,Stop                ; StopPWM
       out         OCR0,mpr

       ldi         mpr,LookLFT             ; Rotate Sonar Left
       out         OCR1AL,mpr

       sei                                       ; Reset Interrupts

       call  NoPingDelay             ; Wait for servo to turn

       call  Look

       sbrc  Obsreg,0                ; If bit one is cleared from LOOK
subroutine,
                                                 ; then no obstacle
found.  Prgm will skip looking
                                                 ; right and break out
       rjmp  Right

       call  Go_left                       ; ...else go left
       rjmp  End_Obstacle            ; Exit subroutine
```

```
Right:

      cli                                     ; Disable interrupts
whil changing Output compare registers
      ldi         mpr,LookRT            ; Rotate Sonar right
      out         OCR1AL,mpr

      sei                                     ; Renable Interrupts

      call  NoPingDelay             ; Wait for servo to turn
      call  NoPingDelay             ; Must travel 180 degrees

      call  Look

      sbrc  Obsreg,0                ; If bit one is cleared from LOOK
subroutine,
                                              ; then no obstacle
found.  Prgm will skip reversing
                                              ; and break out
      rjmp  Reverse

      call  Go_Right                ; ...else turn right
      rjmp  End_Obstacle            ; Exit Subroutine

Reverse:
      cli                                     ; Disable interrupts
whil changing Output compare registers

      ldi         mpr,LookFWD           ; Reset Sonar Forward
      out         OCR1AL,mpr

      call  NoPingDelay             ; Wait for servo to turn

      ldi         mpr,Straight          ; Set direction clutch
neutral
      out         OCR2,mpr

      ldi         mpr,FwdSlow           ; Set reverse speed 2
      out         OCR0,mpr

      sei                                     ; Reenable Interrupts

      call  ReverseDelay

      cli                                     ; Disable interrupts
whil changing Output compare registers

      ldi         mpr,STOP              ; Set Stop
      out         OCR0,mpr

      sei                                     ; Reenable Interrupts


      rjmp  Obstacle                ; Check left and right again for
options
```

```
End_Obstacle:
      ret


;------------------------------------------------------------------
;-----LEDs---------------------------------------------------------

LEDs:

      ret

;------------------------------------------------------------------
;-----Go_Left------------------------------------------------------

Go_Left:
;jmp        testz
      cli                                       ; Disable interrupts
whil changing Output compare registers

      ldi         mpr,LookFWD             ; Reset Sonar Forward
      out         OCR1AL,mpr

      ldi         mpr,FullLFT             ; Gradual Right turn (Set
Direction Clutch)
      out         OCR2,mpr

      ldi         mpr,FwdSlow             ; Set H-Bridge PWM
      out         OCR0,mpr

      sei                                       ; Reenable Interrupts

      call  TurnDelay               ; Wait to complete 90Deg turn

      cli                                       ; Disable interrupts
whil changing Output compare registers

      ldi         mpr,Straight            ; Go Straight (Set Direction
Clutch)
      out         OCR2,mpr

      sei                                       ; Reenable Interrupts

      ret

;------------------------------------------------------------------
;-----Go_Right-----------------------------------------------------

Go_Right:


      cli                                       ; Disable interrupts
whil changing Output compare registers

      ldi         mpr,LookFWD             ; Reset Sonar Forward
      out         OCR1AL,mpr
```

```
      ldi         mpr,FullRT             ; Gradual Right turn (Set
Direction Clutch)
      out         OCR2,mpr

      ldi         mpr,FwdSlow            ; Set Servo PWM
      out         OCR0,mpr

      sei                                      ; Reenable Interrupts

      call  TurnDelay              ; Wait to complete 90Deg turn

      cli                                      ; Disable interrupts
whil changing Output compare registers

      ldi         mpr,Straight           ; Go Straight (Set Direction
Clutch)
      out         OCR2,mpr

      sei                                      ; Reenable Interrupts

      ret

;-----------------------------------------------------------------
;-----Crawl_Reverse-----------------------------------------------

Crawl_Reverse:

      cli                                      ; Disable interrupts
whil changing Output compare registers

      ldi         mpr,LookFWD            ; Reset Sonar Forward
      out         OCR1AL,mpr

      ldi         mpr,Straight           ; Set direction clutch
neutral
      out         OCR2,mpr

      ldi         mpr,FwdSlow            ; Set Servo PWM
      out         OCR0,mpr

      sei                                      ; Reenable Interrupts

      call  TurnDelay              ; Keep going straight backwards

      cli                                      ; Disable interrupts
whil changing Output compare registers

      ldi         mpr,Stop               ; Stop
      out         OCR0,mpr

      sei                                      ; Reenable Interrupts

      ret

;-----------------------------------------------------------------
;-----Crawl_Forward-----------------------------------------------
```

```
Crawl_Forward:

      cli                                         ; Disable interrupts
whil changing Output compare registers
      ldi         mpr,LookFWD               ; Reset Sonar Forward
      out         OCR1AL,mpr

      ldi         mpr,Straight              ; Set direction clutch
neutral
      out         OCR2,mpr

      ldi         mpr,FwdSlow               ; Set Servo Speed to slow
      out         OCR0,mpr

      sei                                         ; Reenable Interrupts

      call  TurnDelay                 ; Keep going straight backwards

      cli                                         ; Disable interrupts
whil changing Output compare registers

      ldi         mpr,Stop                  ; Set H-Bridge PWM to stop
      out         OCR0,mpr

      sei                                         ; Reenable Interrupts

      ret

;-----------------------------------------------------------------
;-----TurnDelay---------------------------------------------------

TurnDelay:

      ldi         r24,low(Turntime)
      ldi   r25,high(Turntime)              ; Prepare register pair as
counter
      ldi         mpr,$10

TurnLoop:
      sbiw  r25:r24,1                       ; Subtract 1 from register
pair
      brne  Turnloop                        ; 3 cycles for these
instructions
                                                        ;   implements
.05328ms delay
      dec         mpr
      brne  turnloop

      ret
;-----------------------------------------------------------------
;-----ReverseDelay------------------------------------------------

ReverseDelay:

      ldi         r24,low(Revtime)
      ldi   r25,high(Revtime)         ; Prepare register pair as counter
```

```
ReverseLoop:
      sbiw  r25:r24,1                     ; Subtract 1 from register
pair
      brne  Reverseloop                   ; 3 cycles for these
instructions
                                                      ;   implements
.05328ms delay

      ret

;------------------------------------------------------------------
;-----NoPingDelay-------------------------------------------------
NoPingDelay:

      ldi         r24,low(NoPing)
      ldi   r25,high(NoPing)        ; Prepare register pair as counter
      ldi         mpr3,$9

NoPIngLoop:
      sbiw  r25:r24,1                     ; Subtract 1 from register
pair
      brne  NoPingloop                    ; 3 cycles for these
instructions
                                                      ;   implements
.05328ms delay
      dec         mpr3
      brne  nopingloop
;jmp  TESTz
      ret



; ***************************** ----------------------------------
; ***** Interupt Handlers ***** ----------------------------------
; ***************************** ----------------------------------

; External Interupts
IntV0:
      reti


IntV1:
      ;     ldi         errorreg,$aa
      ;     inc         errorreg

      ;     cpi         errorreg,5                  ; Check IR 5
times before acting
      ;     brne  endIntV1

            nop                                         ; Execute
ISR intructions here
            ;cli
            ;issue stop
            ;call obstacle
            ;sei
      ;     clr         errorreg                    ; reset register
reti
```

```
      IntV2:            ;Hall Interrupt->Acquires target and moves arm
      ;*****----------------------------------------------------------
-
            cli

      ; Magnet on here
      ; Start moving arm up
            sbi         PORTD,MagOn
            call  delay5s

            sbi         PORTD,ArmDir            ; Set PD0 to '1'-> Arm
Direction
      call  delay1us
      cbi         PORTD,Brake             ; Set Brake bit to low PD0=0
DISENGAGE
      call  delay1us
      ldi         mpr,$AA                       ; Test value *Servo
neutral*(sonar)
      out         OCR1BL,mpr              ; Load OCR1AL with value for
1.5 ms pulse in a T=8.8ms

WaitForUp:
      sbis  PINB,1                        ; PB1= Rear stop switch
      rjmp  WaitForUP

      call  delay5s
      sbi         PORTD,Brake             ; Engage Brake
      call  delay5s                       ; Delay to smooth arm
operation

      cbi         PORTD,MagON             ; Magnet off here
;
      cbi         PORTD,ArmDir            ; Change Directions
      call  delay1us
;
      cbi         PORTD,Brake             ; Set Brake bit to low PD0=0
DISENGAGE
      call  delay1us

      ldi         mpr,$AA                       ; Start Arm Motor
      out         OCR1BL,mpr

WaitForDown:

      sbic  PINB,0                        ; PB0=Front Arm Switch
      rjmp  WaitForDown

      sbi         PORTD,Brake             ; Engage Brake
      call  delay1us
      ldi         mpr,$FF                       ; Stop Arm Brake + PWM
= 0-> Output transistor are off
      out         OCR1BL,mpr

      sei

      reti
```

```
;*****-------------------------------------------------------------

TestZ:


      ldi         mpr,$55
      com         obsreg
      out         portA,obsreg

here:       rjmp  here

;------------------------------------------------------------------
delay1us:
      ldi         mpr,$ff
loopdelay1us:
      dec         mpr
      brne  loopdelay1us

      ret
;------------------------------------------------------------------
delay5s:

      ldi         r24,$ff
      ldi         r25,$ff
      ldi         mpr,$9

delay5sLoop:
      sbiw  r25:r24,1
      brne  delay5sLoop
      dec         mpr
      brne  delay5sLoop
      ret
;-------------------------------------------------------------------
```

```
;--------------------------------------------
;Project Name:  323 16Bit PWM Test.asm
;Description:     Test Single Channel PWM 16Bit Up/Down Counter
;Author:          Max Koessick
;Date;                  July 26, 2003
;Revision:        1.0  Working 16Bit PWM
;                       1.a  Working Ext Interupts (2:0)
;                       1.b    Added 8 bit PWMs
;                       1.c    Added IR IRQ Error Checking Algorithm

;****NOTE****

;You must disable I-bit around OC register changes or an Interrupt may
fire

;System Calculations:
;------------------------------------------
;Use 3.69MHz clock
;Use Prescaler =/64 ->57.6kHz = T=~17uS
;8bit PWM Up/Down counts to $FF->17uS*FF=4.423ms = T(PWM)/2
;@1.0ms, 4.423-1.0/2=3.923ms
;     solve(.003923=.000017x,x)->x=226=$E2 *Servo Left*
;@1.5ms, 4.423-1.5/2=3.673ms
;     solve(.003673=.000017x,x)->x=212=$D4 *Servo Neutral*
;@2.0ms, 4.423-2.0/2=3.423ms
;     solve(.003423=.000017x,x)->x=197=$C5 *Servo Right*

.nolist
.include "m323def.inc"        ; Default Include file for ATMega128
.list                               ; Do not include the "m323def.inc"
in the .lst file

;Interrupt Service Vector Addresses

.org $0000
      rjmp RESET                    ; Reset Vector
.org INT0addr
      rjmp IntV0
.org INT1addr
      rjmp IntV1
.org INT2addr
      rjmp IntV2

;-------------------------------------------
;Register Definitions
;-------------------------------------------

.def mpr          =r16        ; Temporary Register
.def mpr2         =r17
.def errorreg     =r20
;Initialization

RESET:

      clr         errorreg
;-----Setting Stackpointer----------------------------------------
```

```
      ldi         MPR,low(RAMEND)             ; Set stackptr to ram
end
      out         SPL,MPR
      ldi   MPR, high(RAMEND)
      out   SPH, MPR

;-----Set Port Directions-----------------------------------------

      ldi         mpr,0b11110000
      out         DDRD,mpr              ; Set PORTD to output

      ldi         mpr,(1<<PB3)
      out         DDRB,mpr              ; Set PORTB to output

      ser         mpr
      out         DDRC,mpr
      out         DDRA,mpr


;-----Enable 16Bit PWM (Sonar Servo) Counter in 8Bit Mode---------
      ldi         mpr,0b11110001        ; Bit7:6 -> Inverted PWM
                                              ; Bit5:4 -> Disable
OC1B
                                              ; Bit3;2 -> FOC =n/a
                                              ; Bit1:0 -> 8Bit PWM
mode
      out         TCCR1A,mpr

      ldi         mpr,0b00000011        ; Bit7 -> Input Noise
Canceler Disabled
                                              ; Bit6 -> Input Capter
Edge Select n/a
                                              ; Bit5:4 -> Unsused
                                              ; Bit3 -> Clear on
Compare Match Disabled
                                              ; Bit2:0 -> Prescale =
/64
      out         TCCR1B,mpr

;-----Enable 8 bit PWM (Dir and Speed) ---------------------------

      ldi         mpr,0b01110011        ; Bit7 -> FOC2 force Output
Compare = n/a
                                              ; Bit6 -> PWM0 Enables
PWM output
                                              ; Bit5:4 -> Set on
match upcount, clear on match downcount (11)
                                              ; Bit3 -> CTC0 No clear
on match
                                              ; Bit2:0 -> Prescale =
/64
      out         TCCR0,mpr             ; Enable PWM0
      out         TCCR2,mpr             ; Enable PWM2
;------Enable External Interupts----------------------------------

      in          mpr,MCUCSR
```

```
      andi  mpr,0b10111111          ; Clear the INT2 Sense Control Bit
-> Falling Edge triggered
      out         MCUCSR,mpr

      in          mpr,MCUCR
      andi  mpr,$f0                       ; Mask Upper Bits
      ori         mpr,0b00000010          ; Set ISC1:0 Sense Control
bits [3:0] -> Falling Edge for Int0
                                                ; Low level for Int1
(IR) -> ISR must fire as long as a
                                                ; object is detected in
the rear.
      out         MCUCR,mpr

      ldi         mpr,0b11100000          ; Enable Interrupts
      out         GICR,mpr

;-------------------------------------------------------------------


      ldi         mpr,$ce                       ; Test value *Servo
Neutral*(Speed)
      out         OCR0,mpr                ; Load OCR0 with value for
1.0 ms pulse in a T=8.8ms

      ldi         mpr,$a4                       ; Test value *Servo
Neutral*(Direction)
      out         OCR2,mpr                ; Load OCR0 with value for
1.0 ms pulse in a T=8.8ms


      ldi         mpr,$d9                       ; Test value *Servo
neutral*(sonar)
      out         OCR1AL,mpr              ; Load OCR1AL with value for
1.5 ms pulse in a T=8.8ms
      ldi         mpr,$ff                       ; Test value *Servo
neutral*(sonar)
      out         OCR1BL,mpr              ; Load OCR1AL with value for
1.5 ms pulse in a T=8.8ms


                                                ; Interrupts must be
disabled when changing output compare registers
sei

mainloop:
      ldi         mpr,$ff
      out         portc,mpr
      out         porta,mpr


      rjmp  mainloop

IntV0:
      reti

IntV1:                                                ; IR Interrupt
```

```
;	ldi		errorreg,$aa
	inc		errorreg

	cpi		errorreg,5			; Check IR 5 times
before acting
	brne	endIntV1

	nop						; Execute ISR
intructions here
	;cli
	;issue stop
	;call obstacle
	;sei
	clr		errorreg			; reset register


endIntV1:
	;call		delay
	reti

IntV2:

	ldi		mpr,$aa
	com		mpr
	out		portc,mpr
	call	delay
	reti

delay:

	ldi		r24,$ff
	ldi		r25,$ff
	ldi		mpr,$06

here:
	sbiw	r25:r24,1
	brne	here
;	dec		mpr
	;brne here

	ret
```

```
;*********************************************************************
; Ping.inc
; Max Koessick
; IMDL, Summer 2003
; Based on Atmel ATMega323 Datasheet




; Ping Sonar Routine.  Actively seeks the closest object returned as
the low byte in Echo Register 3
;***MASTER TRANSMITTER*****

      ldi                 mpr,(1<<TWINT)|(1<<TWSTA)|(1<<TWEN)
      out                 TWCR,MPR                  ; Send START condition

WAIT1:
      in                  MPR,TWCR                  ; Wait for TWINT Flag
set. This indicates that
      sbrs         MPR,TWINT                  ; the START condition has
been transmitted
      rjmp         WAIT1


      in                  MPR,TWSR                  ; Check value of TWI
Status Register.
      cpi          MPR,START                  ; If status different from
START go to ERROR
      breq         NEXT1
;     jmp                 ERROR1

;***SLAVE ADDRESS + Write***

NEXT1:
      ldi          MPR,SLA+W                  ; Load SLA+W into TWDR
Register
      out                 TWDR,MPR

      ldi          MPR,(1<<TWINT)|(1<<TWEN)
      out                 TWCR,MPR                  ; Clear TWINT bit in
TWCR to start transmission
                                                          ; of address
WAIT2:
      in                  MPR,TWCR                  ; Wait for TWINT Flag
set. This indicates that
      sbrs         MPR,TWINT                  ; SLA+W has been transmitted,
and ACK/NACK has
      rjmp         WAIT2                            ; been received

      in                  MPR,TWSR                  ; Check value of TWI
Status Register. If status
      cpi          MPR,MT_SLA_ACK             ; different from MT_SLA_ACK,
go to ERROR
      breq         NEXT2
      jmp                 ERROR2
```

```
;***Send Command Register Address Byte***

NEXT2:
      ldi         MPR,CommandReg          ; Load data (Address Byte)
into TWDR
      out         TWDR,MPR                ; Register

      ldi               MPR,(1<<TWINT)|(1<<TWEN)
      out         TWCR,MPR                ; Clear TWINT bit in TWCR to
start transmission
                                                      ; of data
WAIT3:

      in                MPR,TWCR                ; Wait for TWINT Flag
set. This indicates that
      sbrs        MPR,TWINT               ; data has been transmitted,
and ACK/NACK has
      rjmp        WAIT3                         ; been received

      in                MPR,TWSR                ; Check value of TWI
Status Register. If status
                                                      ; different from
MT_DATA_ACK, go to ERROR
      cpi         MPR,MT_DATA_ACK
      breq        NEXT4
      jmp               ERROR3

;***Send Ranging Mode Byte***

NEXT4:
      ldi         MPR,Inches              ; Load data (Data Byte) into
TWDR
                                                      ; Register
      out         TWDR,MPR

      ldi         MPR,(1<<TWINT)|(1<<TWEN)
      out         TWCR,MPR                ; Clear TWINT bit in TWCR to
start transmission
                                                      ; of data
WAIT5:

      in                MPR,TWCR                ; Wait for TWINT Flag
set. This indicates that
      sbrs        MPR,TWINT               ; data has been transmitted,
and ACK/NACK has
      rjmp        WAIT5                         ; been received

      in                MPR,TWSR                ; Check value of TWI
Status Register. If status
                                                      ; different from
MT_DATA_ACK, go to ERROR
      cpi         MPR,MT_DATA_ACK
      breq        NEXT5
      jmp               ERROR5

NEXT5:
```

```
;*****Random READ Operation*****

;Send Start Condition
NEXT7:

      call        Delay1                        ; SRF08 must wait
bewteen reading and writing
      ldi               MPR,(1<<TWINT)|(1<<TWSTA)|(1<<TWEN)
      out               TWCR,MPR                ; Send START condition
WAIT8:
      in                MPR,TWCR                ; Wait for TWINT Flag
set. This indicates that
      sbrs        MPR,TWINT               ; the START condition has
been transmitted
      rjmp        WAIT8

      in                MPR,TWSR                ; Check value of TWI
Status Register. If status
                                                      ; different from
START, go to ERROR
      cpi         MPR,rep_START
      breq        NEXT8
      jmp               ERROR6


;***SLAVE ADDRESS + Write*** Setting Address for READ

NEXT8:

      ldi         MPR,SLA+W               ; Load SLA+W into TWDR
Register
      out               TWDR,MPR

      ldi               MPR,(1<<TWINT)|(1<<TWEN);
      out               TWCR,MPR                ; Clear TWINT bit in
TWCR to start transmission
                                                      ; of address
WAIT9:
      in                MPR,TWCR                ; Wait for TWINT Flag
set. This indicates that
      sbrs        MPR,TWINT               ; SLA+W has been transmitted,
and ACK/NACK has
      rjmp        WAIT9                         ; been received

      in                MPR,TWSR                ; Check value of TWI
Status Register. If status
                                                      ; different from
MT_SLA_ACK, go to ERROR
      cpi         MPR,MT_SLA_ACK
      breq        NEXT9

      jmp               ERROR7


;***Send Echo Register 3 Address (low Byte)***Setting Address for READ
```

```
NEXT9:
      ldi         MPR,EchoReg3            ; Load data (Address Byte)
into TWDR Register
      out         TWDR,MPR

      ldi         MPR,(1<<TWINT) | (1<<TWEN)
      out         TWCR,MPR                ; Clear TWINT bit in TWCR to
start transmission
                                                ; of data
WAIT10:

      in                MPR,TWCR                ; Wait for TWINT Flag
set. This indicates that
      sbrs        MPR,TWINT               ; data has been transmitted,
and ACK/NACK has
      rjmp        WAIT10                        ; been received

      in                MPR,TWSR                ; Check value of TWI
Status Register. If status
                                                ; different from
MT_DATA_ACK, go to ERROR
      cpi         MPR,MT_DATA_ACK
      breq        NEXT10
      jmp               ERROR8

;Send Repeated Start Condition

NEXT10:

      ldi               MPR,(1<<TWINT)|(1<<TWSTA)|(1<<TWEA)|(1<<TWEN)
      out               TWCR,MPR                ; Send REP_START
condition
WAIT11:
      in                MPR,TWCR                ; Wait for TWINT Flag
set. This indicates that
      sbrs        MPR,TWINT               ; the START condition has
been transmitted
      rjmp        WAIT11

      in                MPR,TWSR                ; Check value of TWI
Status Register. If status
                                                ; different from
START, go to ERROR
      cpi         MPR,rep_START
      breq        NEXT11
      jmp               ERRORa

;***SLAVE ADDRESS+READ***

NEXT11:
      ldi         MPR,SLA+R               ; Load SLA+R into TWDR
Register
      out               TWDR,MPR

      ldi         MPR,(1<<TWINT)|(1<<TWEN)
      out               TWCR,MPR                ; Clear TWINT bit in
TWCR to start transmission
```

```
                                                          ; of SLA+R,
enable TWI and generate an ACK, TWEA=1
WAIT12:
      in                MPR,TWCR                  ; Wait for TWINT Flag
set. This indicates that
      sbrs        MPR,TWINT                 ; SLA+R has been transmitted,
and ACK/NACK has
      rjmp        WAIT12                    ; been received

      in                MPR,TWSR                  ; Check value of TWI
Status Register. If status
                                                          ; different from
MR_SLA_ACK, go to ERROR
      cpi         MPR,MR_SLA_ACK
      breq        NEXT12
      jmp               ERRORb

NEXT12:
;Get EchoRegister 3 data
      ldi               MPR,(1<<TWINT)|(1<<TWEN)
      out               TWCR,MPR                  ; Clear TWINT bit in
TWCR to start reception of
                                                          ; data. Not
setting TWEA causes NACK to be
                                                          ; returned after
reception of next data byte
                                                          ; receive last
data byte. Signal this to Slave
                                                          ; by returning
NACK
WAIT13:
      in                MPR,TWCR                  ; Wait for TWINT Flag
set. This indicates that
      sbrs        MPR,TWINT                 ; data has been received and
NACK returned
      rjmp        WAIT13

      in                MPR,TWSR                  ; Check value of TWI
Status Register. If status
      cpi         MPR,MR_DATA_NACK  ; different from MR_DATA_NACK, go
to ERROR
      breq        NEXT13
      jmp               ERRORc

NEXT13:

      in                ECHO1L,TWDR               ; Input received data
from TWDR.
      mov               mpr3,ECHO1L               ; Move ECHO1L Contents
to multipurpose register3
                                                          ;  to avoid
corruption
      com               mpr3                      ; Prepare for LED
output
      out               PORTA,mpr3                ; Put Echo Results onto
LEDs (PortA)
      out               portc,mpr3
```

```
;Issue Stop

      ldi         MPR,(1<<TWINT)|(1<<TWSTO)|(1<<TWEN)
      out               TWCR,MPR                ; Send STOP signal

END_GET_PING:
      ret                                             ; Return from
subrouting GET_PING

;***Error Detection Routine***
;Error will be presented as a or'ed pair of the step in which
; the program broke and the TWSR
ERROR1:
      ldi               ErrorReg,$01
      rjmp        output
ERROR2:
      ldi               ErrorReg,$02
      rjmp        output
ERROR3:
      ldi               ErrorReg,$03
      rjmp        output
ERROR4:
      ldi               ErrorReg,$04
      rjmp        output
ERROR5:
      ldi               ErrorReg,$05
      rjmp        output
ERROR6:
      ldi               ErrorReg,$06
      rjmp        output
ERROR7:
      ldi               ErrorReg,$07
      rjmp        output
ERROR8:
      ldi               ErrorReg,$08
      RJMP        output
ERROR9:
      ldi               ErrorReg,$09
      RJMP        output
ERRORa:
      ldi               ErrorReg,$0A
      RJMP        output
ERRORb:
      ldi               ErrorReg,$0B
      RJMP        output
ERRORc:
      ldi               ErrorReg,$0c
      RJMP        output
ERRORd:
      ldi               ErrorReg,$0d
      RJMP        output
Output:

; Load Contents of TWI Status Register and display on Port C (LEDs)
```

```
      in                 MPR,TWSR                ; Load the TWSR for
Error display
      or                 MPR,errorreg
      com                MPR                           ; Change to
active low LEDs
;     out                PORTA,errorreg

      rjmp        END_GET_PING

;----------------------------------------------------------------------
; There must be delay loop between reading and writing to the SRF08
Delay1:

      push               XH
      push        XL
      push        mpr

      ldi                XH,$00
      ldi                XL,$50
      ldi                mpr,$03

loop4:
      sbiw        XH:XL,1
      brne        loop4
      dec                mpr
      brne loop4

      pop                mpr
      pop                XL
      pop                XH
      ret
```

```
; LCD_Init.inc

; Initializes LCD for Mega323
; Max Koessick
; IMDL, Summer 2003
; Based on information from www.mil.ufl.edu/4744

LCDInit:
      push  mpr
;-----------------------------------------------------------------
      call  DELAY3ms                    ; Wait 15ms for
Initialization
      call  DELAY3ms
      call  DELAY3ms
      call  DELAY3ms
      call  DELAY3ms

;Set # Display lines, 8-bit mode and Font------------------------

      ldi   mpr,0b0000000
      out   PORTE,mpr                   ; Activate command register


      ldi   mpr,0b00110000
      out   PORTB,mpr                   ; Function Set to 8-bit
operation


      ldi   mpr,0b01000000                    ; Activate LCD Enable
      out   PORTE,mpr

      ldi   mpr,0b00000000
      out   PORTE,mpr                   ; Deactivate LCD Enable

      call  delay4_1ms

      ldi   mpr,0b01000000                    ; Activate LCD Enable
      out   PORTE,mpr

      ldi   mpr,0b00000000
      out   PORTE,mpr                   ; Deactivate LCD Enable

      call  delay100us

      ldi   mpr,0b01000000                    ; Activate LCD Enable
      out   PORTE,mpr

      ldi   mpr,0b00000000
      out   PORTE,mpr                   ; Deactivate LCD Enable

      call  delay4_1ms

      ldi   mpr,0b01000000                    ; Activate LCD Enable
      out   PORTE,mpr

      ldi   mpr,0b00000000
```

```
      out   PORTE,mpr                    ; Deactivate LCD Enable
;Set Number of Lines and Pitch------------------------------------

      ldi   mpr,0b0000000
      out   PORTE,mpr                    ; Activate command register

      ldi   mpr,0b00111000
      out   PORTB,mpr                    ; Function Set to 2 lines and
5x8 pitch

      ldi   mpr,0b01000000                     ; Activate LCD Enable
      out   PORTE,mpr

      ldi   mpr,0b00000000
      out   PORTE,mpr                    ; Deactivate LCD Enable

      call  delay40us

;Display, Cursor, and Blink Off-----------------------------------

      ldi   mpr,0b0000000
      out   PORTE,mpr                    ; Activate command register

      ldi   mpr,0b00001000
      out   PORTB,mpr                    ; Turn them off!

      ldi   mpr,0b01000000                     ; Activate LCD Enable
      out   PORTE,mpr

      ldi   mpr,0b00000000
      out   PORTE,mpr                    ; Deactivate LCD Enable

      call  delay40us

;Clear Screen, Cursor Home-----------------------------------

      ldi   mpr,0b0000000
      out   PORTE,mpr                    ; Activate command register

      ldi   mpr,0b00000001
      out   PORTB,mpr                    ; Do it!

      ldi   mpr,0b01000000                     ; Activate LCD Enable
      out   PORTE,mpr

      ldi   mpr,0b00000000
      out   PORTE,mpr                    ; Deactivate LCD Enable

      call  delay1_64ms

;Inc Cursor Right, No shift-----------------------------------

      ldi   mpr,0b0000000
      out   PORTE,mpr                    ; Activate command register

      ldi   mpr,0b00000110
      out   PORTB,mpr                    ; Do It!
```

```
      ldi   mpr,0b01000000                      ; Activate LCD Enable
      out   PORTE,mpr

      ldi   mpr,0b00000000
      out   PORTE,mpr                     ; Deactivate LCD Enable

      call  delay40us

;Display, Cursor, and Blink Off-----------------------------------

      ldi   mpr,0b0000000
      out   PORTE,mpr                     ; Activate command register

      ldi   mpr,0b00001111
      out   PORTB,mpr                     ; Turn them on!

      ldi   mpr,0b01000000                      ; Activate LCD Enable
      out   PORTE,mpr

      ldi   mpr,0b00000000
      out   PORTE,mpr                     ; Deactivate LCD Enable

      call  delay40us

      pop   mpr
      ret
;----------------------------------------------------------------------
DELAY3ms:

      push  XL
      push  XH                            ; Save registers in
Subroutine
      ldi   XL,$FF
      ldi   XH,$BB                              ; 0xBBFF=3.007ms @
16MHz
LOOP_3:
      sbiw  XH:XL,1
      brne  LOOP_3

      pop   XH
      pop   XL                            ; Restore Registers

      ret                                 ; Return from subroutine
;----------------------------------------------------------------------
DELAY4_1ms:

      push  XL
      push  XH                            ; Save registers in
Subroutine
      ldi   XL,$FF
      ldi   XH,$ff                              ; 0xFFFF=4.09ms @ 16MHz
LOOP4_1:
      sbiw  XH:XL,1
      brne  LOOP4_1

      pop   XH
```

```
      pop   XL                            ; Restore Registers

      ret                                 ; Return from subroutine
;----------------------------------------------------------------------
DELAY40us:

      push  XL
      push  XH                            ; Save registers in
Subroutine
      ldi   XL,$8F
      ldi   XH,$02                              ; 0x028f=40.9us @ 16MHz
LOOP40:
      sbiw  XH:XL,1
      brne  LOOP40

      pop   XH
      pop   XL                            ; Restore Registers

      ret                                 ; Return from subroutine

;----------------------------------------------------------------------

DELAY100us:

      push  XL
      push  XH                            ; Save registers in
Subroutine
      ldi   XL,$4F
      ldi   XH,$06                              ; 0x064F=100.9us @
16MHz
LOOP100us:
      sbiw  XH:XL,1
      brne  LOOP100us

      pop   XH
      pop   XL                            ; Restore Registers

      ret                                 ; Return from subroutine

;----------------------------------------------------------------------

DELAY1_64ms:

      push  XL
      push  XH                            ; Save registers in
Subroutine
      ldi   XL,$FF
      ldi   XH,$66                              ; 0x66FF=1.64ms @ 16MHz
LOOP1_64ms:
      sbiw  XH:XL,1
      brne  LOOP1_64ms

      pop   XH
      pop   XL                            ; Restore Registers

      ret                                 ; Return from subroutine
```

```
;-----------------------------------------------------------------
; Name:                  MicroChip323.asm
; Description:     ATMega323 Two Wire Interface (IC2) Test Program
;                        Interfaces Microchip 24AA256K Memory to IC2 Bus

; Author:          Max Koessick
; Class:           EEL5666C, Intelligent Machine Design Lab
; Date:                  June 28, 2003
; Revision         1.a
; Changes to Date:
;                        7/2/03 First Revision
;                        7/6/03 Working
;-----------------------------------------------------------------

.nolist                                                 ; Do not include
in .lst file
.include "m323def.inc"                      ; Standard ATMega323 Include
File
.include "TWI.inc"                                ; Two Wire Interface
Error code definitions
.list
; Interrupt service vectors

.org $0000
      rjmp Reset                                  ; Reset vector

;-----------------------------------------------------------------
; Register defines for main loop
;-----------------------------------------------------------------


.def        mpr         =r16                ; defines multipurpose
register
.def        mpr2  =r17              ; multipurpose register 2
.def        ECHOL =r18
.def        ECHOH =r19
.def        ErrorReg=r20
.def        mpr3  =r21


; Equate statements
.equ        W                 = 0                 ; Write Bit
.equ        R                 = 1                 ; Read Bit
.equ        SLA               = $A0         ; Slave Address of 24AA256
.equ        Addr        = $ff         ; Random address
.equ        AddrHigh    = $00         ; SRF08 Command Register
.equ        Data        = $ef
;-----------------------------------------------------------------
; Reset vector
;-----------------------------------------------------------------

Reset:
;-----Setting Stackpointer----------------------------------------
      ldi         MPR,low(RAMEND)               ; Set stackptr to ram
end
      out         SPL,MPR
      ldi   MPR, high(RAMEND)
```

```
      out   SPH, MPR

;-----Set Port Directions----------------------------------------
      ser         mpr                                ; Set TEMP to $FF
to...
      out   DDRB,mpr
;----------------------------------------------------------------

      clr   ErrorReg                     ; For Debug purposes


; Set TWIBitRate for fclk=3.69Mhz

      ldi               mpr,11
      ;100Khz=3.69MHz/(16+2*12) See Datasheet Pg202
      out               TWBR,mpr


; Initialize TWCR Register

      ldi         MPR,(1<<TWEN);
      out               TWCR,MPR               ; Initialize TW Control
Register

      ldi               mpr,$01
      out               TWAR,mpr

      sei                                            ; set interrupts
active

;***MASTER TRANSMITTER*****

      ldi         MPR,(1<<TWINT)|(1<<TWSTA)|(1<<TWEN)
      out               TWCR,MPR               ; Send START condition

WAIT1:
      in                MPR,TWCR               ; Wait for TWINT Flag
set. This indicates that
      sbrs        MPR,TWINT              ; the START condition has
been transmitted
      rjmp        WAIT1


      in                MPR,TWSR               ; Check value of TWI
Status Register.
      cpi         MPR,START              ; If status different from
START go to ERROR
      breq        NEXT1
      jmp               ERROR1

;***SLAVE ADDRESS + Write***

NEXT1:
      ldi         MPR,SLA+W              ; Load SLA+W into TWDR
Register
      out               TWDR,MPR
```

```
      ldi         MPR,(1<<TWINT)|(1<<TWEN)
      out               TWCR,MPR                ; Clear TWINT bit in
TWCR to start transmission
                                                      ; of address
WAIT2:
      in                MPR,TWCR                ; Wait for TWINT Flag
set. This indicates that
      sbrs        MPR,TWINT               ; SLA+W has been transmitted,
and ACK/NACK has
      rjmp        WAIT2                         ; been received

      in                MPR,TWSR                ; Check value of TWI
Status Register. If status
      cpi         MPR,MT_SLA_ACK          ; different from MT_SLA_ACK,
go to ERROR
      breq        NEXT2
      jmp               ERROR2

;***Send Address Byte***

NEXT2:

      ldi         MPR,Addr                ; Load data (Address Byte)
into TWDR
      out         TWDR,MPR                ; Register

      ldi               MPR,(1<<TWINT)|(1<<TWEN)
      out         TWCR,MPR                ; Clear TWINT bit in TWCR to
start transmission
                                                      ; of data
WAIT3:

      in                MPR,TWCR                ; Wait for TWINT Flag
set. This indicates that
      sbrs        MPR,TWINT               ; data has been transmitted,
and ACK/NACK has
      rjmp        WAIT3                         ; been received

      in                MPR,TWSR                ; Check value of TWI
Status Register. If status
      cpi         MPR,MT_DATA_ACK         ; different from MT_DATA_ACK,
go to ERROR
      breq        NEXT4
      jmp               ERROR3

;***Send Data Byte***

NEXT4:
      ldi         MPR,Data                ; Load data (Data Byte) into
TWDR
      out         TWDR,MPR                ; Register

      ldi               MPR,(1<<TWINT)|(1<<TWEN)
      out         TWCR,MPR                ; Clear TWINT bit in TWCR to
start transmission
                                                      ; of data
WAIT5:
```

```
      in                  MPR,TWCR                  ; Wait for TWINT Flag
set. This indicates that
      sbrs        MPR,TWINT                   ; data has been transmitted,
and ACK/NACK has
      rjmp        WAIT5                             ; been received

      in                  MPR,TWSR                  ; Check value of TWI
Status Register. If status
                                                          ; different from
MT_DATA_ACK, go to ERROR
      cpi         MPR,MT_DATA_ACK
      breq        NEXT5
      jmp                 ERROR5
;Send Stop Condition-24AA256 Writes to memory after Stop condition
NEXT5:
      ldi         mpr,(1<<TWINT)|(1<<TWSTO)|(1<<TWEN)
      out                 TWCR,mpr

check:
      in                  mpr,TWCR
      andi        mpr,0b00010000
      brne        check

;     call        delay65ms

;*****Random READ Operation*****

;Send Start Condition
NEXT7:
      ldi                 MPR,(1<<TWINT)|(1<<TWSTA)|(1<<TWEN)
      out                 TWCR,MPR                  ; Send START condition
WAIT8:
      in                  MPR,TWCR                  ; Wait for TWINT Flag
set. This indicates that
      sbrs        MPR,TWINT                   ; the START condition has
been transmitted
      rjmp        WAIT8

      in                  MPR,TWSR                  ; Check value of TWI
Status Register. If status
                                                          ; different from
START, go to ERROR
      cpi         MPR,START
      breq        NEXT8
      JMP                 ERROR6


;***SLAVE ADDRESS + Write*** Setting Address for READ

NEXT8:

      ldi         MPR,SLA+W                   ; Load SLA+W into TWDR
Register
      out                 TWDR,MPR

      ldi                 MPR,(1<<TWINT)|(1<<TWEN);
```

```
	out		TWCR,MPR		; Clear TWINT bit in 
TWCR to start transmission
						; of address
WAIT9:
	in		MPR,TWCR		; Wait for TWINT Flag 
set. This indicates that
	sbrs	MPR,TWINT		; SLA+W has been transmitted, 
and ACK/NACK has
	rjmp	WAIT9			; been received

	in		MPR,TWSR		; Check value of TWI 
Status Register. If status
	cpi		MPR,MT_SLA_ACK		; different from MT_SLA_ACK, 
go to ERROR
	breq	NEXT9
	jmp		ERROR7


;***Send Address High Byte***Setting Address for READ

NEXT9:
	ldi		MPR,Addr		; Load data (Address Byte) 
into TWDR
	out		TWDR,MPR		; Register

	ldi		MPR,(1<<TWINT)|(1<<TWEN)
	out		TWCR,MPR		; Clear TWINT bit in TWCR to 
start transmission
						; of data
WAIT10:
	in		MPR,TWCR		; Wait for TWINT Flag 
set. This indicates that
	sbrs	MPR,TWINT		; data has been transmitted, 
and ACK/NACK has
	rjmp	WAIT10			; been received

	in		MPR,TWSR		; Check value of TWI 
Status Register. If status
	cpi		MPR,MT_DATA_ACK	; different from MT_DATA_ACK, go to 
ERROR
	breq	NEXT10
	jmp		ERROR8

;***Send Repeated Start Condition***
NEXT10:
	ldi		MPR,(1<<TWINT)|(1<<TWSTA)|(1<<TWEN)
	out		TWCR,MPR		; Send REP_START 
condition

WAIT11:
	in		MPR,TWCR		; Wait for TWINT Flag 
set. This indicates that
	sbrs	MPR,TWINT		; the START condition has 
been transmitted
	rjmp	WAIT11
```

```
	in		MPR,TWSR		; Check value of TWI
Status Register. If status
	cpi	MPR,rep_START		; different from START, go to
ERROR
	breq	NEXT11
	JMP		ERRORa

;***SLAVE ADDRESS+READ*** (Random Read)

NEXT11:
	ldi	MPR,SLA+R		; Load SLA+W into TWDR
Register
	out		TWDR,MPR

	ldi	MPR,(1<<TWINT)|(1<<TWEN)
	out		TWCR,MPR		; Clear TWINT bit in
TWCR to start transmission
							; of SLA+R,
enable TWI and generate an ACK, TWEA=1
WAIT12:
	in		MPR,TWCR		; Wait for TWINT Flag
set. This indicates that
	sbrs	MPR,TWINT		; SLA+R has been transmitted,
and ACK/NACK has
	rjmp	WAIT12			; been received

	in		MPR,TWSR		; Check value of TWI
Status Register. If status
	cpi	MPR,MR_SLA_ACK		; different from MR_SLA_ACK,
go to ERROR
	breq	NEXT12
	jmp		ERRORb

NEXT12:
;Get last data Byte
	ldi		MPR,(1<<TWINT)|(1<<TWEN)
	out		TWCR,MPR		; Clear TWINT bit in
TWCR to start reception of
							; data. Not
setting TWEA causes NACK to be
							; returned after
reception of next data byte
							; receive last
data byte. Signal this to Slave
							; by returning
NACK
WAIT13:
	in		MPR,TWCR		; Wait for TWINT Flag
set. This indicates that
	sbrs	MPR,TWINT		; data has been received and
NACK returned
	rjmp	WAIT13

	in		MPR,TWSR		; Check value of TWI
Status Register. If status
	cpi	MPR,MR_DATA_NACK	; different from MR_DATA_NACK, go
to ERROR
```

```
      breq        NEXT13
      jmp               ERRORc

NEXT13:
      in                ECHOL,TWDR              ; Input received data
from TWDR.
      com               ECHOL                   ; Invert to put onto
LEDs
      out               PORTB,ECHOL
;Issue Stop

      ldi         MPR,(1<<TWINT)|(1<<TWSTO)|(1<<TWEN)
      out               TWCR,MPR                ; Send STOP signal

MAINLOOP:

      rjmp        mainloop

ERROR1:
      ldi               ErrorReg,$01
      rjmp        output
ERROR2:
      ldi               ErrorReg,$02
      rjmp        output
ERROR3:
      ldi               ErrorReg,$03
      rjmp        output
ERROR4:
      ldi               ErrorReg,$04
      rjmp        output
ERROR5:
      ldi               ErrorReg,$05
      rjmp        output
ERROR6:
      ldi               ErrorReg,$06
      rjmp        output
ERROR7:
      ldi               ErrorReg,$07
      rjmp        output
ERROR8:
      ldi               ErrorReg,$08
      RJMP        output
ERROR9:
      ldi               ErrorReg,$09
      RJMP        output
ERRORa:
      ldi               ErrorReg,$0A
      RJMP        output
ERRORb:
      ldi               ErrorReg,$0B
      RJMP        output
ERRORc:
      ldi               ErrorReg,$0c
      RJMP        output
ERRORd:
      ldi               ErrorReg,$0d
      RJMP        output
```

```
Output:

; Load Contents of TWI Status Register and display on Port C (LEDs)

      in                 mpr2,TWCR               ; Load the TWSR for
Error display
      or                 mpr2,errorreg
      com                mpr2                    ; Change to active low
LEDs
      out                PORTB,mpr2
LOOP1:
      rjmp        loop1
; *** 65ms delay while Sonar process data
;----------------------------------------------------------------------
Delay65ms:

      push               XH
      push        XL
      push        mpr2

      ldi                XH,$ff
      ldi                XL,$00
      ldi                mpr2,$00

loop:
      sbiw        XH:XL,1
      brne        loop

      pop         mpr2

      pop                XL
      pop                XH

      ret

Test:
      ldi         mpr3,$aa
      out         PORTB,mpr3
loop2:
      rjmp  loop2
      ret

test2:
      in          mpr3,twsr
      com         mpr3
      out         PORTB,mpr3
loop3:
      rjmp  loop3
```

```
;-----------------------------------------------------------------
; Name:                 Starting Wait Loop.asm
; Description:    Implements Starting Loop for Robot Demo.
;                       Wait until either PinE6 or PinE7 is pressed
before
;                             program sequence starts

; Author:         Max Koessick
; Class:          EEL5666C, Intelligent Machine Design Lab
; Date:                 July 8, 2003
; Revision        1.a (completed and 100% Functional)

; PE6 and PE7 are connected to normally closed switches.
; Internal Pullups are enabled and a high true signal is wanted
; Program stays in wait loop until PE6 or PE7 goes high
; Signaling that a bump switch has been tapped
;-----------------------------------------------------------------

.nolist
.include "m323def.inc"
.list

; Interrupt service vectors

.org $0000
      rjmp Reset                                ; Reset vector

;-----------------------------------------------------------------
; Register defines for main loop
;-----------------------------------------------------------------

.def        mpr         =r16              ; defines multipurpose
register

;-----------------------------------------------------------------
; Reset vector
;-----------------------------------------------------------------

Reset:
;-----Setting Stackpointer-----------------------------------------
      ldi         MPR,low(RAMEND)               ; Set stackptr to ram
end
      out         SPL,MPR
      ldi   MPR, high(RAMEND)
      out   SPH, MPR

;-----Set Port Directions-----------------------------------------
      ldi         mpr,0b11110011                ; Set PE6 and PE7 to
input
      out         DDRD,mpr
      ldi         mpr,(1<<PD2)|(1<<PD3)
      out         PortD,mpr                     ; Set Pullups on Input

      ser         mpr
      out         DDRA,mpr                      ; for testing
      out         PortA,mpr                     ; lights off
;-----------------------------------------------------------------
```

```
WaitToStart:
      in          mpr,PIND                       ; read Port E
      andi  mpr,$80                              ; mask lower bits
      sbrc  mpr,7                          ; skip if bit in register set
      rjmp  Start                          ; ...if not, break out
      in          mpr,PIND                       ; read Port E
      andi  mpr,$40                              ; mask bit 6
      sbrc  mpr,6                          ; skip if bit in register set

      rjmp  Start                          ; ...if not, break out
      rjmp  WaitToStart                    ; keep waiting

Start:
      clr         mpr
      out         PortA,mpr                      ; Turn LEDs on

Mainloop:
      rjmp  mainloop
```

```
      ;------------------------------------------
;Project Name:  323 Arm and Magnet.asm
;Description:      Test H-Bridge control of arm and Main motor plus
;                        Power FET/Magnet ops
;Author:           Max Koessick
;Date;                   July 26, 2003
;Revision:         1.0  Working 16Bit PWM
;                        1.a  Working Ext Interupts (2:0)
;                        1.b    Added 8 bit PWMs
;                        1.c  Fixed Intermittent IRQ firing
;                        1.d  Final Version
;                                    Arm working correctly
;                                    1) Turn On Magnet
;                                    2) Raises Arm until feedback switch
is pressed
;                                    3) Delay
;                                    4) Turn Off Magnet
;                                    6) Lower Arm Until Fedback switch
is pressed
;------------------------------------------
;Use 3.69MHz clock
;Use Prescaler =/64 ->57.6kHz = T=~17uS
;8bit PWM Up/Down counts to $FF->17uS*FF=4.423ms = T(PWM)/2
;@1.0ms, 4.423-1.0/2=3.923ms
;     solve(.003923=.000017x,x)->x=226=$E2 *Servo Left*
;@1.5ms, 4.423-1.5/2=3.673ms
;     solve(.003673=.000017x,x)->x=212=$D4 *Servo Neutral*
;@2.0ms, 4.423-2.0/2=3.423ms
;     solve(.003423=.000017x,x)->x=197=$C5 *Servo Right*

.nolist
.include "m323def.inc"        ; Default Include file for ATMega128
.list                               ; Do not include the "m323def.inc"
in the .lst file

;Interrupt Service Vector Addresses

.org $0000
      rjmp RESET                    ; Reset Vector
.org INT0addr
      rjmp IntV0
.org INT1addr
      rjmp IntV1
.org INT2addr
      rjmp IntV2

;-------------------------------------------
;Register Definitions
;-------------------------------------------

.def mpr          =r16        ; Temporary Register
.def oldsd        =r17        ; Old Speed Register
.def newspd       =r18        ; New Speed Register
.def mpr2         =r19
```

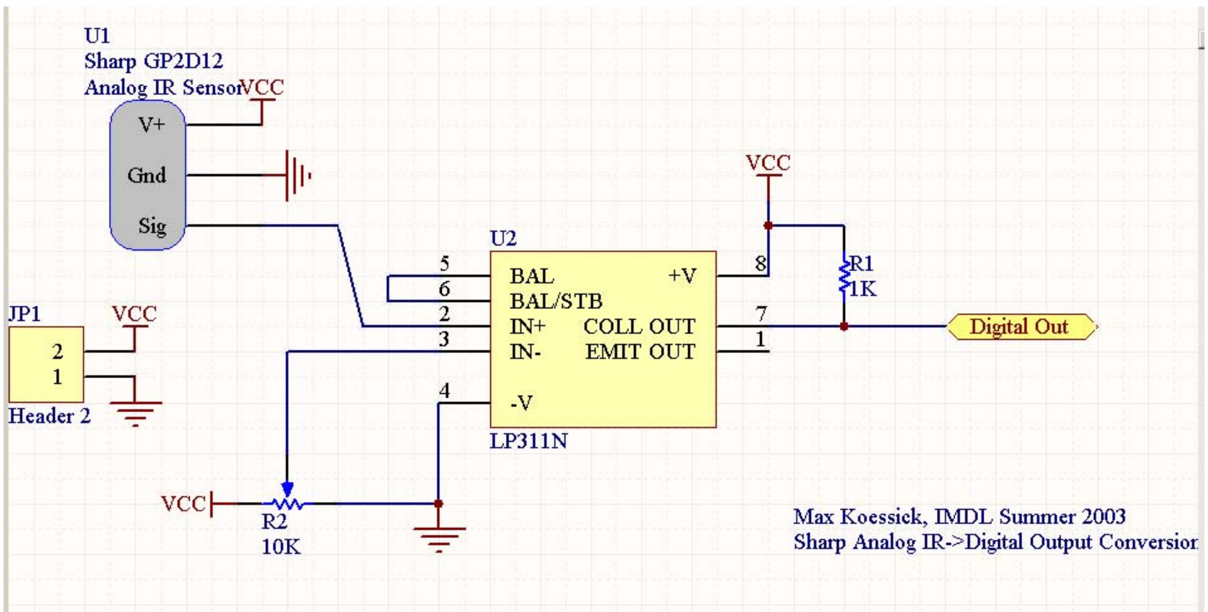
```
.equ brake        = 1
.equ ArmDir       = 0
.equ MagOn        = 6
;Initialization




RESET:

;-----Setting Stackpointer----------------------------------------
      ldi         MPR,low(RAMEND)               ; Set stackptr to ram
end
      out         SPL,MPR
      ldi   MPR, high(RAMEND)
      out   SPH, MPR

;-----Set Port Directions-----------------------------------------

      ldi         mpr,0b11110011          ; Set PD2 and PD3 to input
      out         DDRD,mpr                ; Set PORTD to output

      ldi         mpr,0b11111000
      out         DDRB,mpr                ; Set PORTB to output
      ldi         mpr,(1<<PB0)|(1<<PB1)
      out         PORTB,mpr               ; Enable Internal pull up for
PB0,PB1

      ser         mpr
      out         DDRC,mpr
      out         DDRA,mpr
      out         PORTC,mpr
      out         PORTA,mpr               ; LEDs off

;-----Enable 16Bit PWM (Sonar Servo -A) and Arm Motor (OCR1B) Counter
in 8Bit Mode---------


      ldi         mpr,0b11110001          ; Bit7:6 -> Inverted PWM
                                                ; Bit5:4 -> Disable
OC1B
                                                ; Bit3;2 -> FOC =n/a
                                                ; Bit1:0 -> 8Bit PWM
mode
      out         TCCR1A,mpr

      ldi         mpr,0b00000011          ; Bit7 -> Input Noise
Canceler Disabled
                                                ; Bit6 -> Input Capter
Edge Select n/a
                                                ; Bit5:4 -> Unsused
                                                ; Bit3 -> Clear on
Compare Match Disabled
                                                ; Bit2:0 -> Prescale =
/64
      out         TCCR1B,mpr
```

```
      sbi         PORTD,Brake             ; Set Brake bit to low PD0=0

;-----Enable 8 bit PWM (Dir and Speed) ---------------------------



;     ldi         mpr,$d4                       ; Test value *Servo
Neutral*
;     out         OCR0,mpr                ; Load OCR0 with value for
1.0 ms pulse in a T=8.8ms
;     out         OCR2,mpr                ; Sets servos to neutral at
program startup

      ldi         mpr,0b01110011          ; Bit7 -> FOC2 force Output
Compare = n/a
                                                ; Bit6 -> PWM0 Enables
PWM output
                                                ; Bit5:4 -> Set on
match upcount, clear on match downcount (11)
                                                ; Bit3 -> CTC0 No clear
on match
                                                ; Bit2:0 -> Prescale =
/64
      out         TCCR0,mpr               ; Enable PWM0
      out         TCCR2,mpr               ; Enable PWM2
;------Enable External Interupts---------------------------------

      in          mpr,MCUCSR
      andi  mpr,0b10111111          ; Clear the INT2 Sense Control Bit
-> Falling Edge triggered
      out         MCUCSR,mpr

      in          mpr,MCUCR
      andi  mpr,$f0                       ; Mask Upper Bits
      ori         mpr,0b00000010          ; Set ISC1:0 Sense Control
bits [3:0] -> Falling Edge for Int0
                                                ; Low level for Int1
(IR) -> ISR must fire as long as a
                                                ; object is detected in
the rear.
      out         MCUCR,mpr

      ldi         mpr,0b11100000          ; Enable Interrupts
      out         GICR,mpr


;-----------------------------------------------------------------

mainloop:

;******* when this code is a subroutine, clear the I-bit here ******

;     cli

; Magnet on here
; Start moving arm up
```

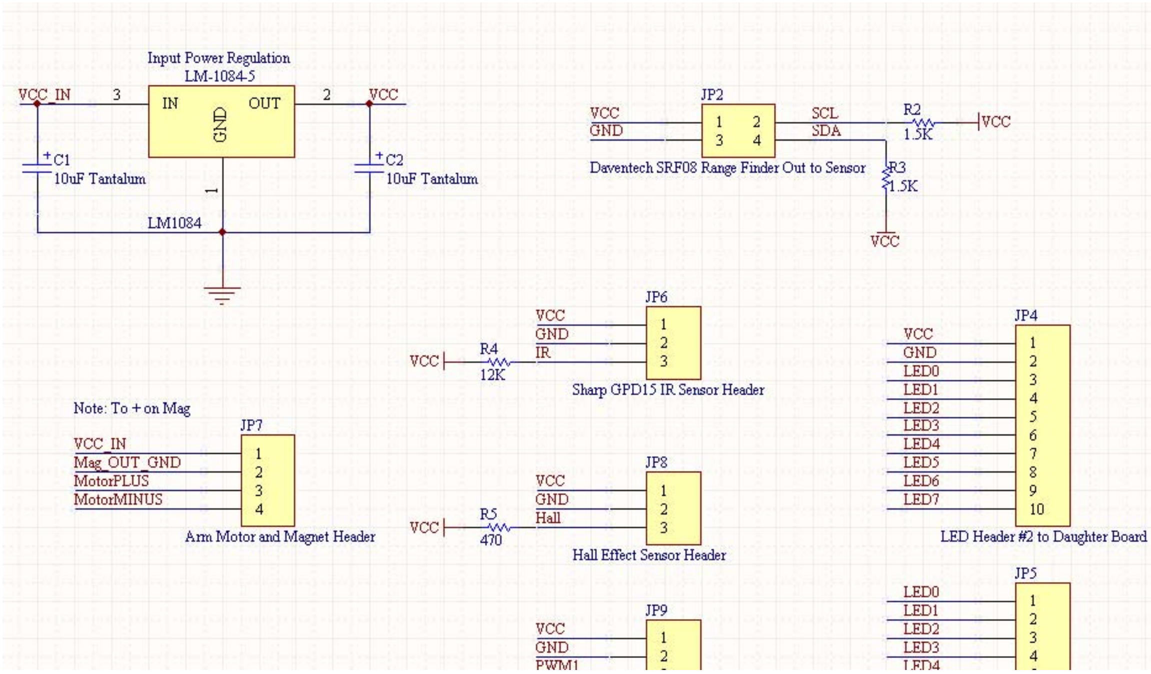
```
      sbi         PORTD,MagOn
      call  delay5s

      sbi         PORTD,ArmDir            ; Set PD0 to '1'-> Arm
Direction
      call  delay1us
      cbi         PORTD,Brake             ; Set Brake bit to low PD0=0
DISENGAGE
      call  delay1us
      ldi         mpr,$aa                 ; Test value *Servo
neutral*(sonar)
      out         OCR1BL,mpr              ; Load OCR1AL with value for
1.5 ms pulse in a T=8.8ms

WaitForUp:
      sbis  PINB,1                        ; PB1= Rear stop switch
      rjmp  WaitForUP

;     call  delay5s
      sbi         PORTD,Brake             ; Engage Brake
      call  delay5s                       ; Delay to smooth arm
operation

      cbi         PORTD,MagON             ; Magnet off here
;
      cbi         PORTD,ArmDir            ; Change Directions
      call  delay1us
;
      cbi         PORTD,Brake             ; Set Brake bit to low PD0=0
DISENGAGE
      call  delay1us

;     ldi         mpr,$AA                       ; Start Arm Motor
;     out         OCR1BL,mpr

WaitForDown:

      sbic  PINB,0                        ; PB0=Front Arm Switch
      rjmp  WaitForDown

      sbi         PORTD,Brake             ; Engage Brake
      call  delay1us
      ldi         mpr,$FF                       ; Stop Arm Brake + PWM
= 0-> Output transistor are off
      out         OCR1BL,mpr

      sei                                       ; Reenable I-Bit

mloop:
;Exit subroutine here
      rjmp  mloop

IntV0:

      reti

IntV1:
```

```
      reti

IntV2:
      reti

;------------------------------
delay1us:
      ldi         mpr,$ff
loopdelay1us:
      dec         mpr
      brne  loopdelay1us

      ret
;------------------------------
delay5s:

      ldi         r24,$ff
      ldi         r25,$00
;     ldi         mpr,$3

delay5sLoop:
      sbiw  r25:r24,1
      brne  delay5sLoop
;     dec         mpr
;     brne  delay5sLoop
      ret
;------------------------------DISENGAGE
Test:

      LDI         MPR,$aA
      OUT         PORTa,MPR

      rjmp  end
end:
      ret
```

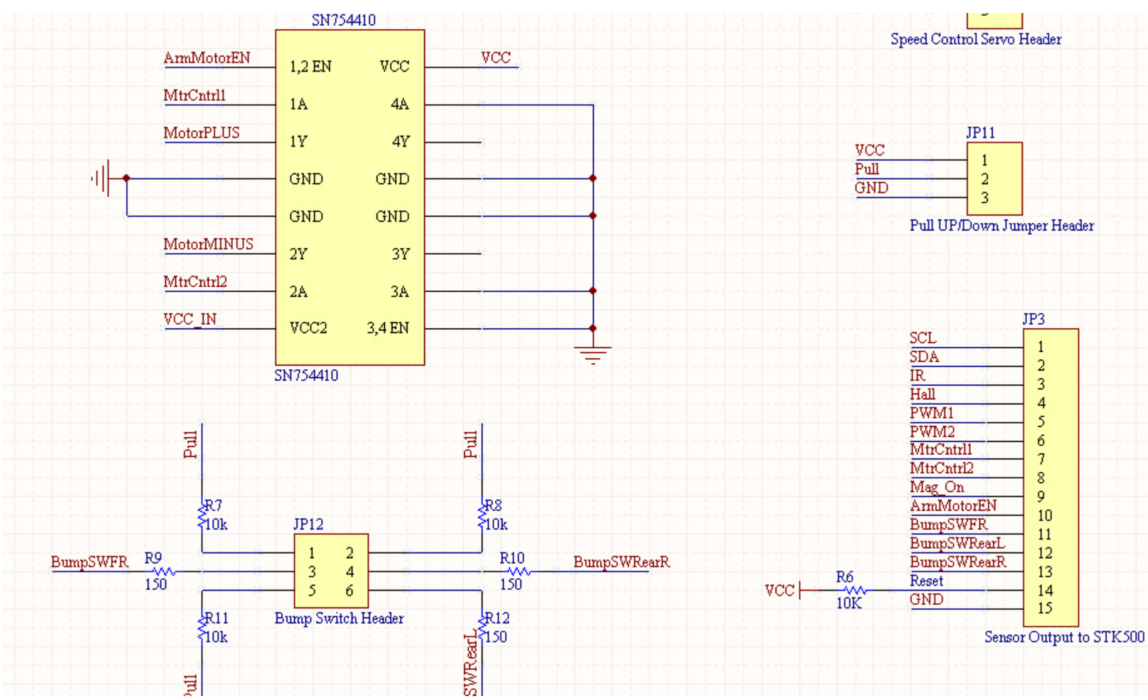
Circuit Schematics



Appendix B.1 GP2D12 Digital Conversion 1

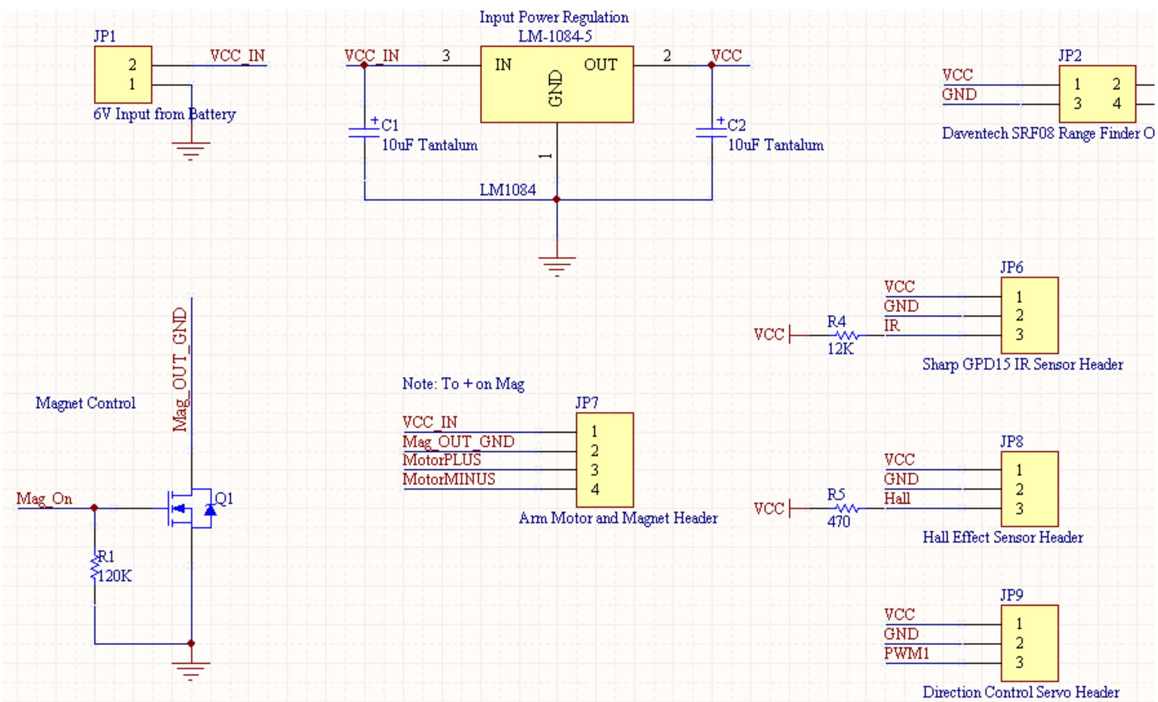


Appendix B.2 Main Daughter Board 1



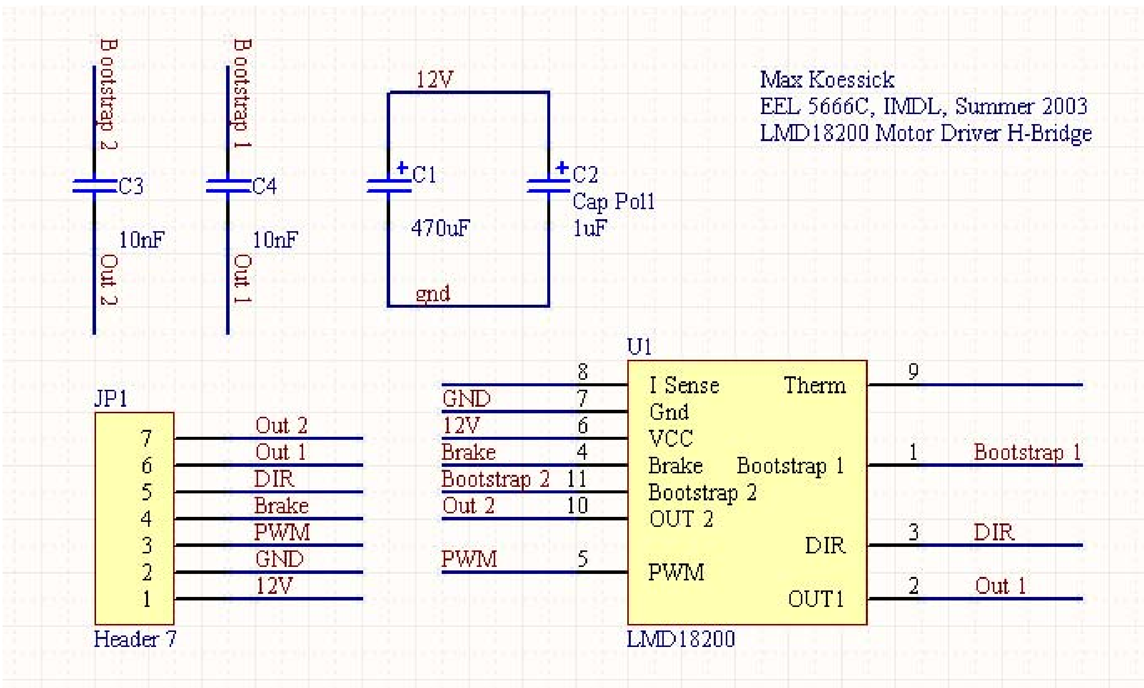
Appendix B.2 Main Daughter Board 2

Appendix B:
Circuit Schematics



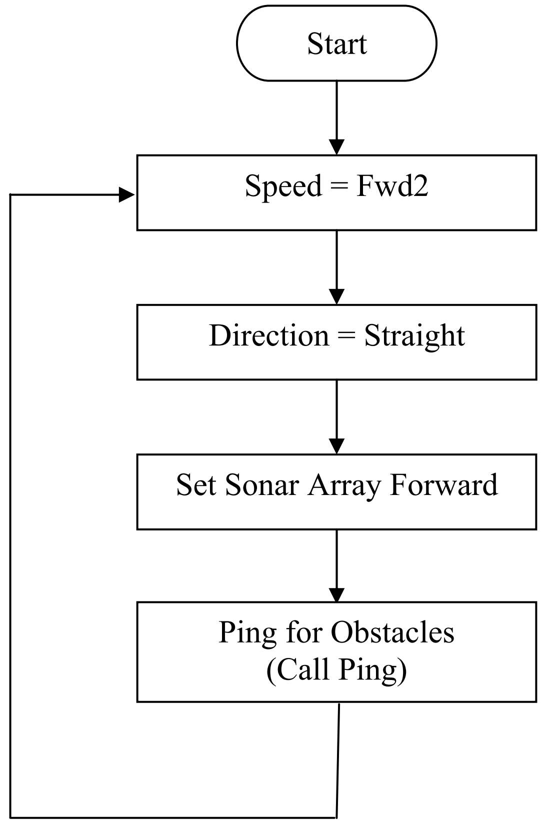
Appendix B.2 Main Daughter Board 3

Circuit Schematics

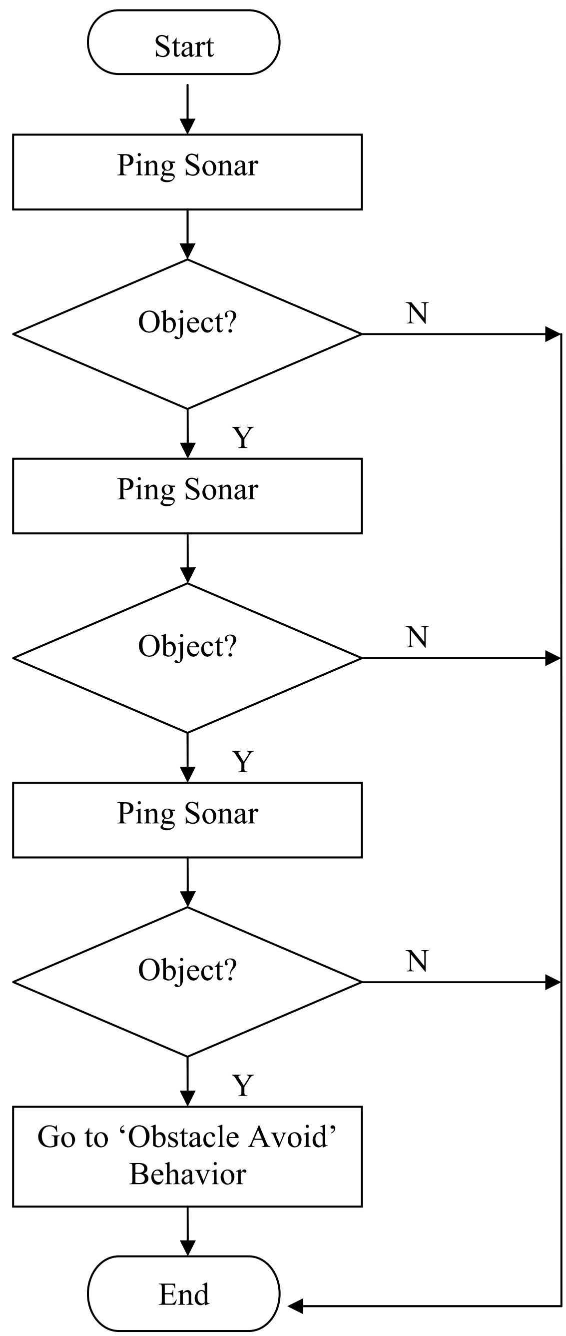


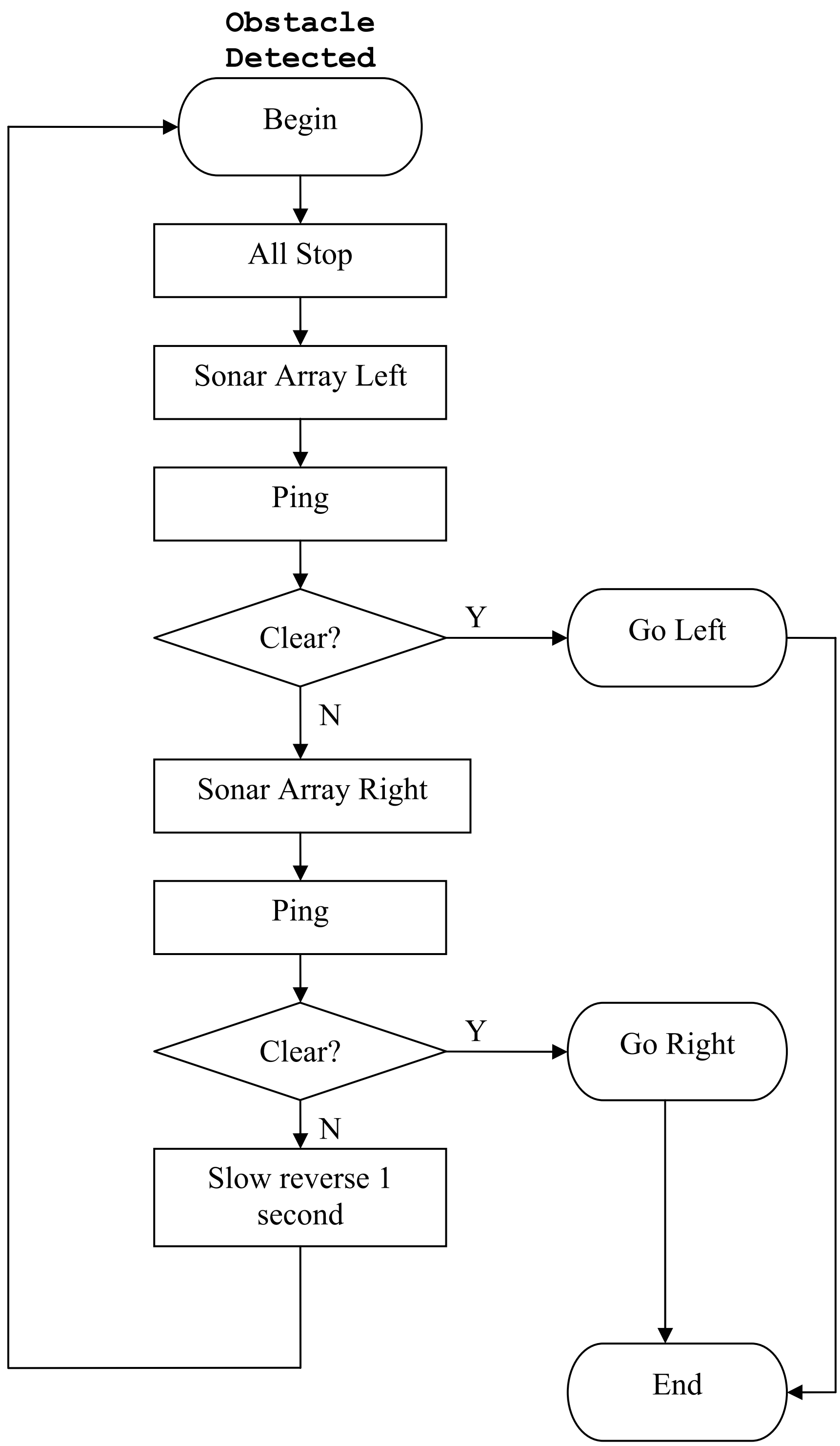
Appendix B.3 LMD18200 Motor Driver 1

Main

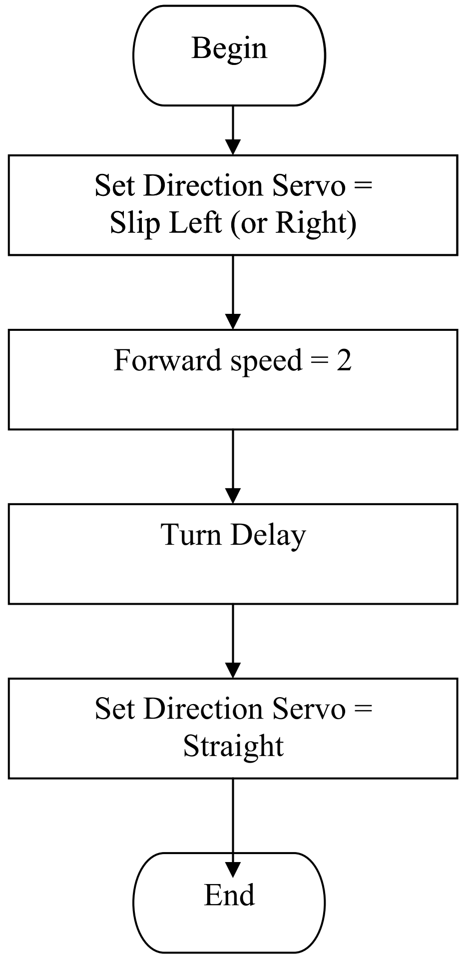

Start
Speed = Fwd2
Direction = Straight
Set Sonar Array Forward
Ping for Obstacles
(Call Ping)

Ping

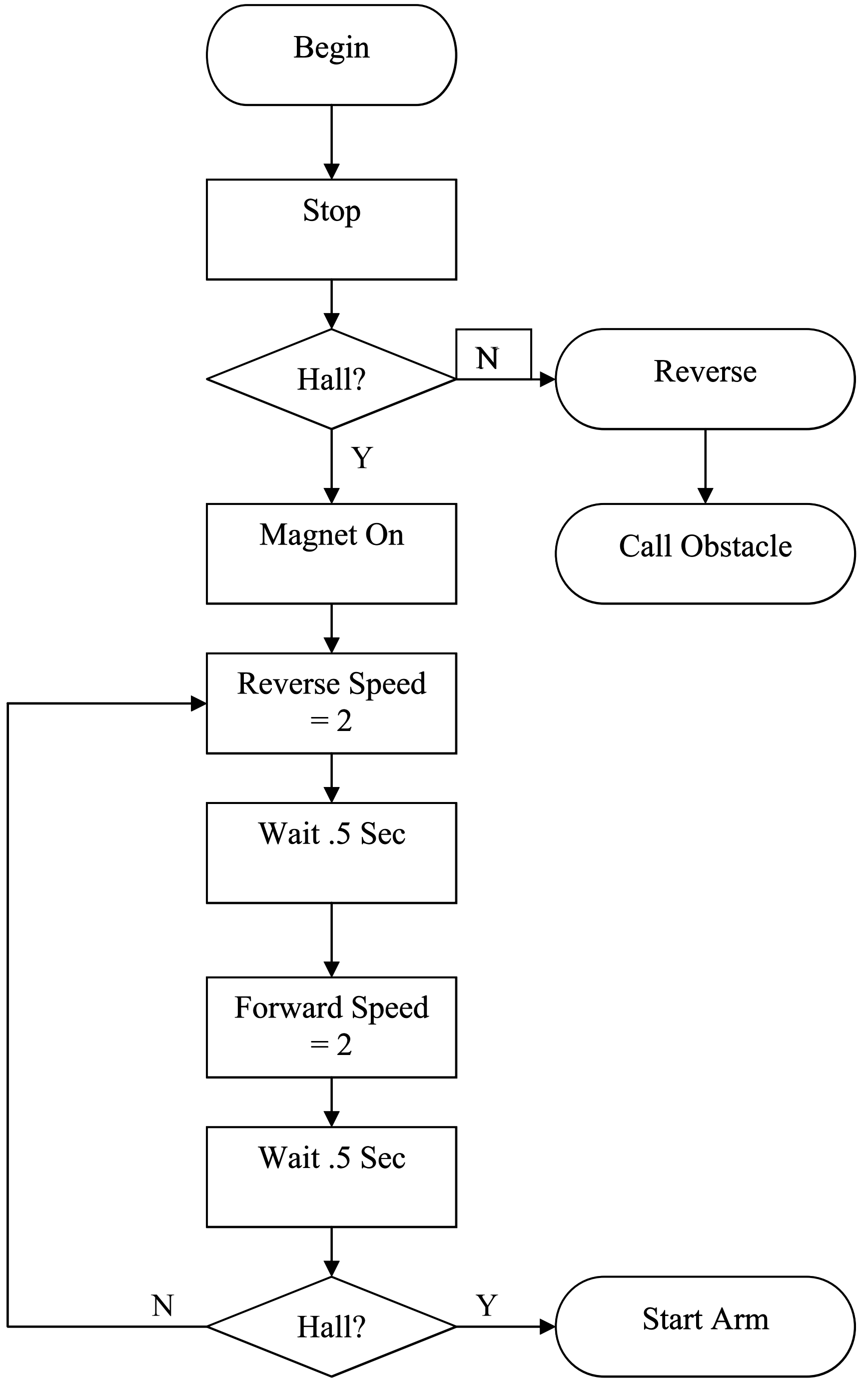

Start
Ping Sonar
Object?
N
Y
Ping Sonar
Object?
N
Y
Ping Sonar
Object?
N
Y
Go to 'Obstacle Avoid' Behavior
End


Obstacle
Detected
Begin
All Stop
Sonar Array Left
Ping
Clear?
Y
Go Left
N
Sonar Array Right
Ping
Clear?
Y
Go Right
N
Slow reverse 1 second
End

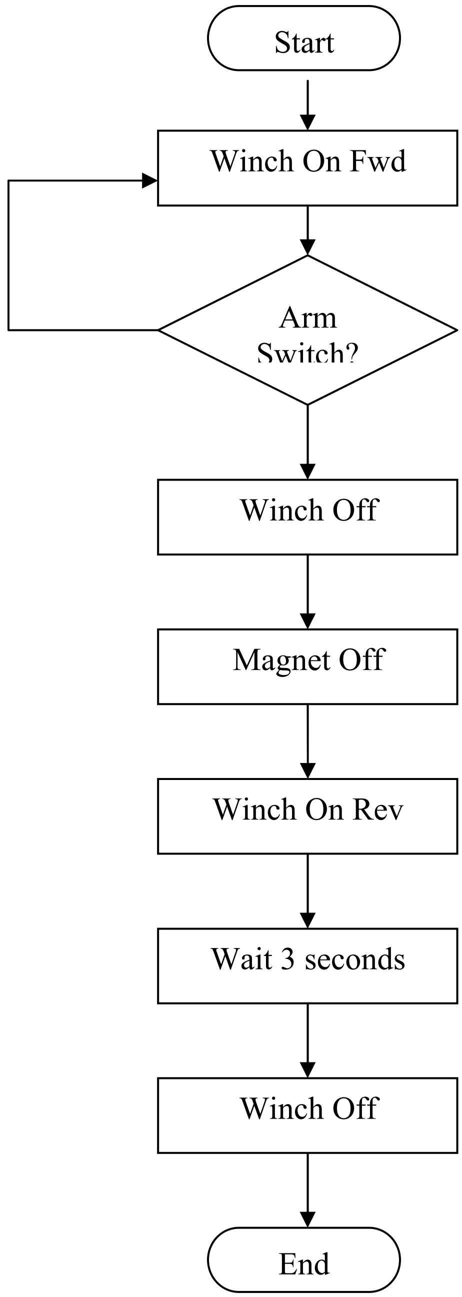
Go Left (or Right)

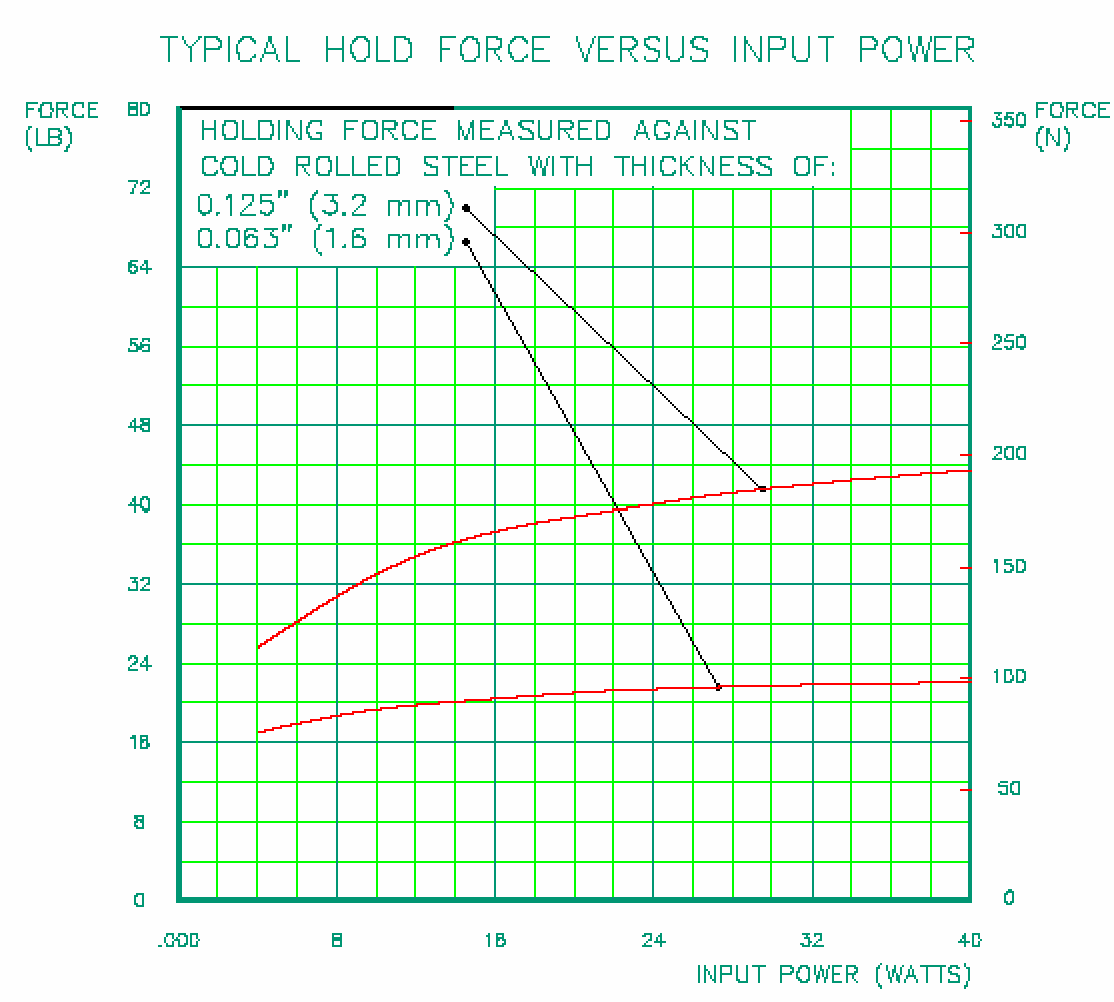


Possible Target Interrupt Request



Arm


Start
Winch On Fwd
Arm Switch?
Winch Off
Magnet Off
Winch On Rev
Wait 3 seconds
Winch Off
End



(Graph Courtesy of Solenoid City)

Special Sensor Report: Daventech SRF08

Introduction

Sensor Synopsis

The Daventech SRF08 ultrasonic range finder (sonar array) uses a pulse ('ping') of sound to determine the range of up to 17 targets in an area. The SRF08 emits a ping and then waits for the first echo to return. This process takes approximately 65ms to complete.

The sonar array communicates with the host microprocessor via the Inter Integrated Circuit Bus (I2C) developed by Phillips for communicating within consumer electronics. Atmel uses this standard in the form of the Two Wire Interface (TWI).

Project Overview

ShopBot is an autonomous vehicle that will navigate a garage floor. It will pick up any tools that it finds, i.e. sockets, etc . . . The robot will wander the floor in a random pattern until it comes in contact with a target. It uses a combination of IR and a Hall Effect proximity sensor to determine target validity. A valid target is simply a ferrous object.

Sensor Integration and Purpose

The SRF08's main purpose in the world of ShopBot is obstacle avoidance from forward, left and right directions.

Under forward movement, the sonar will constantly ping until it detects an object that is less than 36" away. This alert will cause ShopBot to slow down. If it is a tool, it will pass under the sonar as ShopBot advances. However, if this is a wall, the target will keep registering as an obstacle and at 9", ShopBot will change directions.

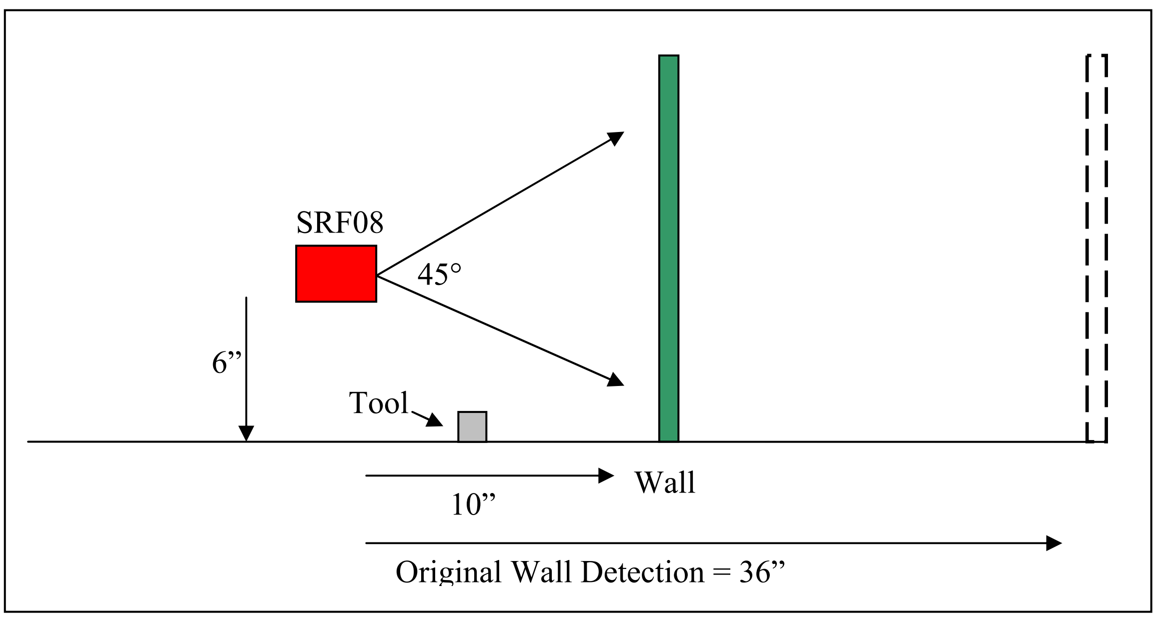


Figure 1. Tool/Wall Detection Scheme

Special Sensor Report: Daventech SRF08

Figure 2 is an illustration provided by Daventech. The beam diffusion illustrates that at 1 foot range, there is approximately a 45° spread. This is used to calculate the distance at which an average 1" tall tool will slip 'underneath the radar.'

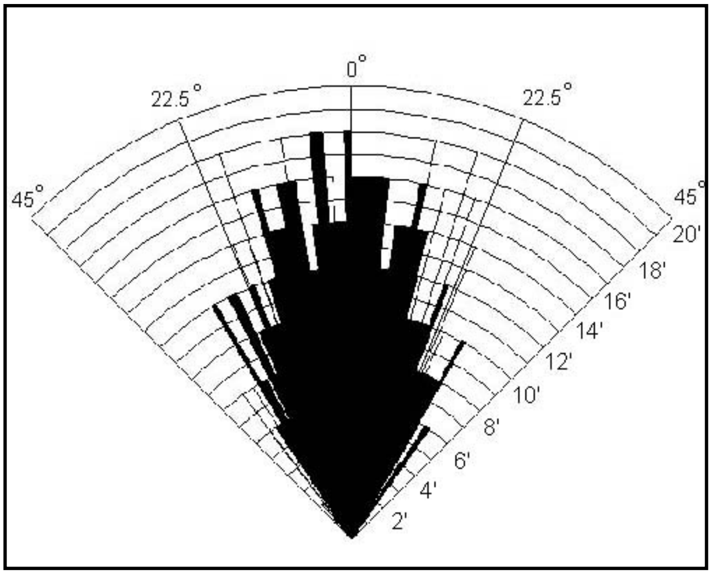


Figure 2. SRF08 Beam Pattern

The SRF08 is 6" above ground. Therefore, using the Pythagorean Theorem (with the hypotenuse = 1'), the third leg of the triangle that constitutes the ground plane would be approximately 10" (refer to Figure 1).

Lastly, since this is a tank with one discrete drive motor, it can only turn by stopping one set of tracks. It cannot rotate in place. Therefore, object detection is necessary to either left or right directions when a change in heading is required. To meet this requirement, the SRF08 is mounted on a servo that can rotate ±90° to aid in side obstacle detection.

Figure 3. SRF08 Mounting Location

Special Sensor Report: Daventech SRF08

Testing

The first obstacle to overcome in implementation was the mastering of the I2C bus. This was realized in assembly code. Due to sensor mounting location, there are several echo rejection criteria that must be met (see Figure 3).

Forward Looking

In forward looking scenarios, the SRF08 tended to pick up echoes from the robot platform itself. To prove this, an experiment was set up where the first object detected would be forced. Further, the platform was put on the edge of a chair and aimed at a wall. This way, the first object detected could be predicted with reasonable certainty.

Any reading closer than 6″ would be rejected as the part of the platform. Specifically, the front bumper and arm are within the 45° beam diffusion. Figure 4 depicts the experiment. With nothing above or below, it is reasonable that the first objects detected will be the platform and then the wall, in that order. By rejecting the first echo register (the closest object), a reading of 24″ was returned in the next echo register. Actual distance was approximately 24′.

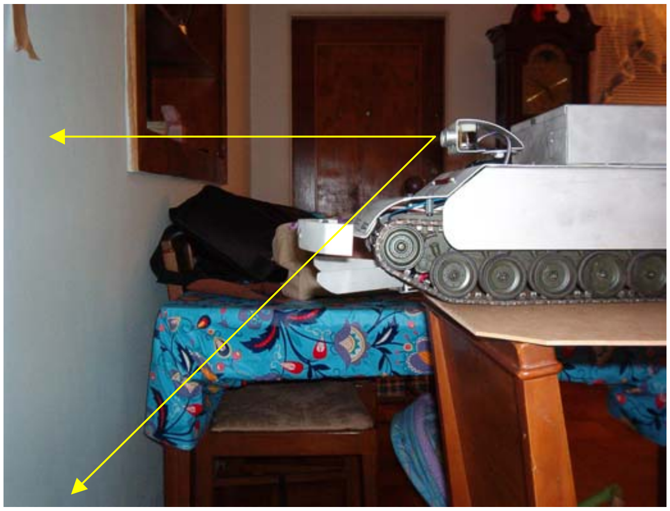


Figure 4. Forward Looking Sonar Ping Experiment

Side Looking

A similar experiment was setup to test side looking effectiveness. This time, however, both possible surfaces of corruption (top of platform and side of processor housing) are parallel to the sound waves and shouldn't theoretically interfere. However, this was not the case.

When turning to the side, the servo could not turn parallel both angles each time. Moreover, readings were returned that would be from objects under 1-2″. Therefore, again, the first readings were thrown out.

Special Sensor Report: Daventech SRF08

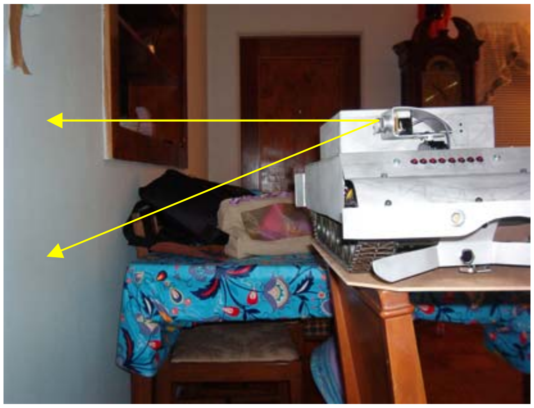


Figure 5. Side Ping Experiment

Figure 6. Rotated Sonar Array

Software Examples are found in the previous software section

Mr. Tool was originally called 'ShopBot.'

Special Sensor Report: Electromagnet

Description

Solenoid City's E-20-100 is a light duty electromagnet. In Mr. Tool, it is used to grasp ferrous tools and move them into a basket. Implementation is fairly simple in that the only circuitry needed is a TTL switch that can handle the high current needed to activate the electromagnet. Figure 1 depicts a drawing the magnet. A 10-32 thread is provided in the top for mounting purposes.

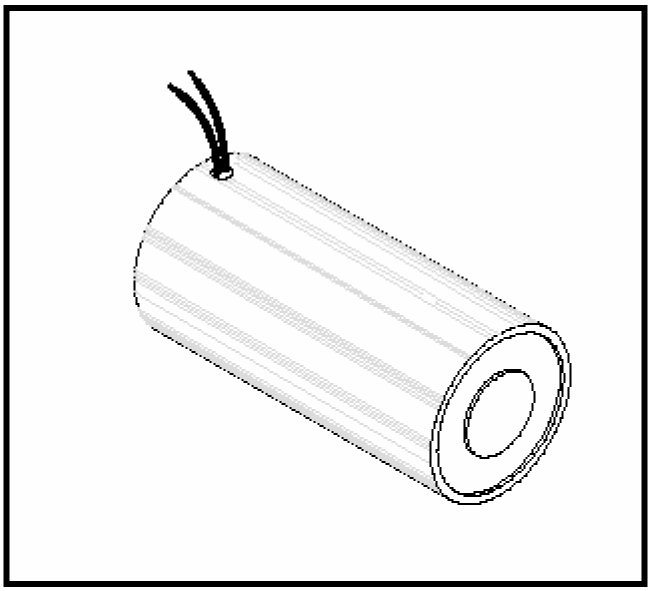


Figure 17. Solenoid City's E Series Electromagnet (Courtesy Solenoid City)

Advantages and Disadvantages

In a nutshell, this is the easiest way to pick up a ferrous object. Solenoid City's simple magnet is much easier to implement that any sort of robotic hand or grabber. This one advantage far outweighs the two disadvantages of weight and power consumption.

The E-20-100 is very robust at 5.3 ounces. The robot platform that incorporates this particular model must be capable of moving it. Moreover, plywood platforms would be questionable. The second disadvantage is power consumption. From Figure 2, at a typical 4-12V robot platform, the magnet consumes from typically .5A at 4 Watts to 1.5A at 12 Watts (assuming an average 8V system). Therefore, power supplies and switches must be chosen to accommodate this demand.

Special Sensor Report: Electromagnet

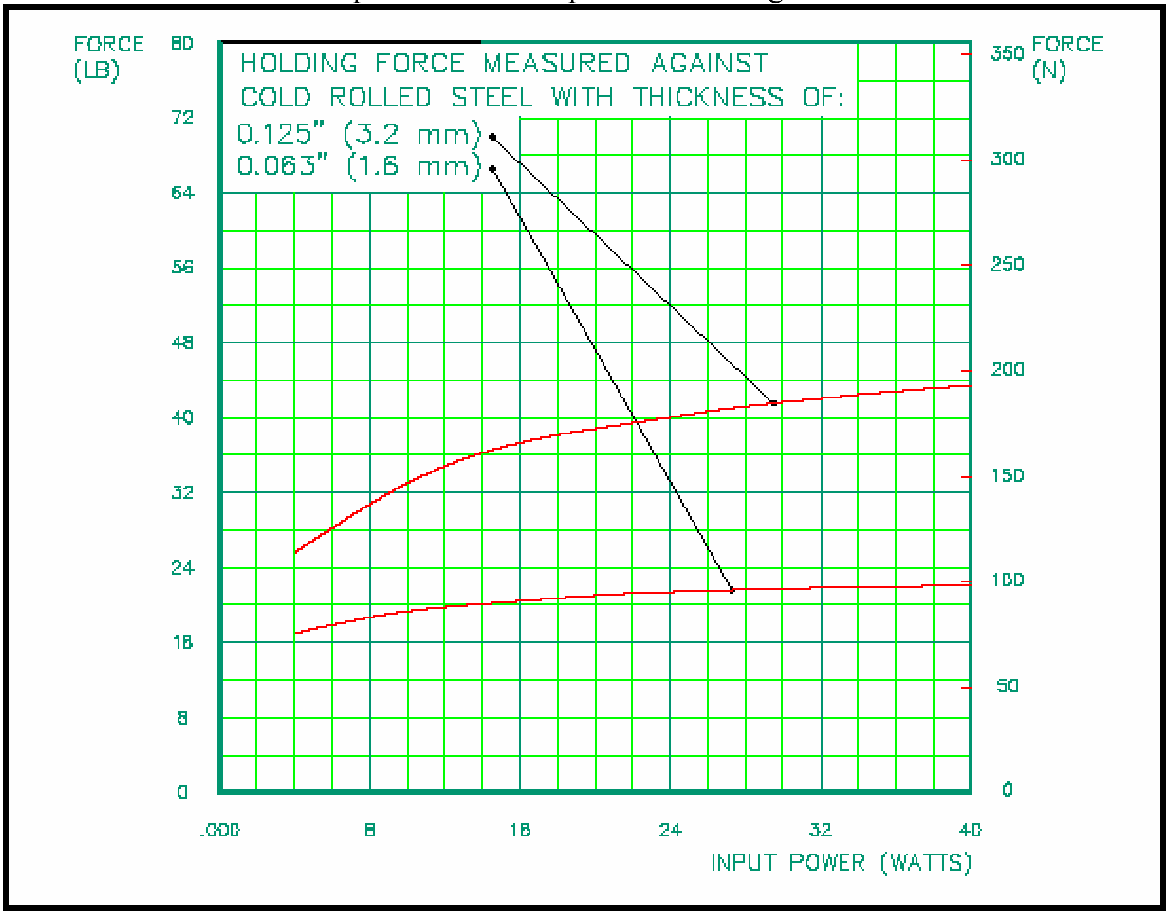


Figure 18. Power Consumption vs. Holding Force (Courtesy Solenoid City)

Interface

Figure 3 shows the typical interface. As stated earlier, a high power capacity switch is needed to control the current to the magnet. In this case, a Fairchild HUF76107 Power FET was chosen because of its high handling capacity. It is capable of loads up to 20A and 30V. These criteria exceed the needs of the electromagnet.

The gate is activated by standard TTL signals, therefore making the design positive logic. The FET can be directly connected any port pin on a microprocessor that supply TTL levels on output ports. When the gate is driven high, the Power FET supplies ground closing the circuit and energizing the magnet's core.

The 120kΩ pull down resistor is added to ensure an off state in the event of a floating input.

Special Sensor Report: Electromagnet

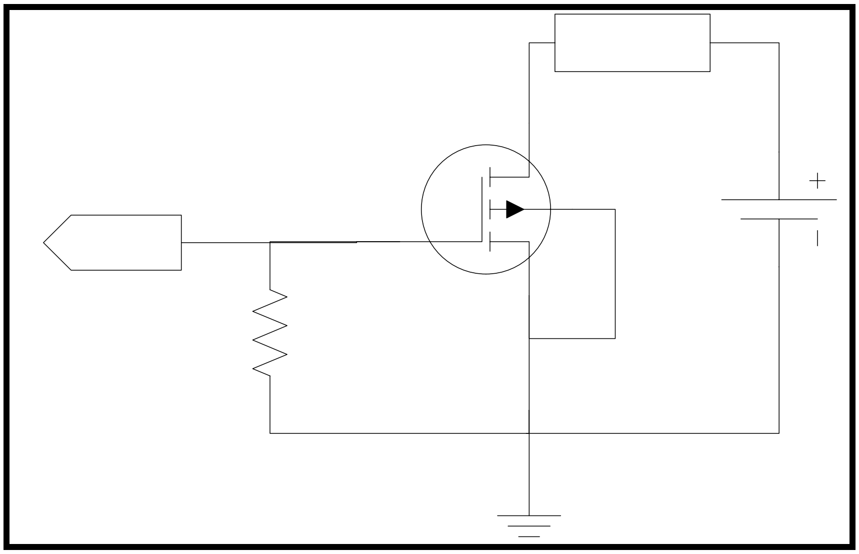


Figure 19. Interface Circuit

Availability and Cost

The E-20-100 can be easily purchased online through www.solenoidcity.com for a price of $35 plus shipping. Other magnets are available to fit most applications.

Sources:

"E-20-100.pdf" Datasheet, www.solenoidcity.com

Input from u

Special Sensor Report: Hall Sensor

Description

The GS100701's primary purpose is high speed gear sensing. Normal applications include automotive applications and machinery speed sensing. However, this hall type sensor can also be used to detect metal objects that are within close proximity to the head. In Mr. Tool, it is used to accept/reject ferrous targets.

This model is a sinking interface, i.e. negative logic.

The sensor contains internal integrated circuitry that is basically an open collector bipolar junction transistor. The BJT supplies ground on the signal output wire when a ferrous (gear) target is sensed. The only external circuitry that is needed is a pull-up resistor that is determined by input voltage. The GS100701 can operate on voltages from 5 to 24 VDC.

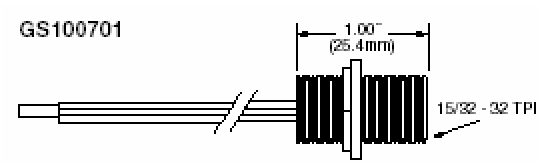


Figure 20. GS100701 Gear Tooth Sensor (Courtesy Cherry Sensor)

Advantages and Disadvantages

Advantages include easy integration into any existing design. All that is required is a simple pull up resistor. Table 1 describes possible resistor values

Volts dc	5	9	12	15	24
Ohms	470	820	1.2K	1.5K	2.2K

Table 1. Resistor Values

The main disadvantage is in the metal detection application. Any metal has to be close (<5 mm) before a logic one is output on the signal wire

Interface

Figure 2 shows the typical interface. No other external circuitry is needed.

Special Sensor Report: Hall Sensor

Normal software approach would include polling or the use of external interrupts. Mr. Tool uses the previous, so no relevant software is available. Once an object is detected using an alternate means (IR/Photo Transistor), the GS100701 is used to determine whether the object is ferrous or not.

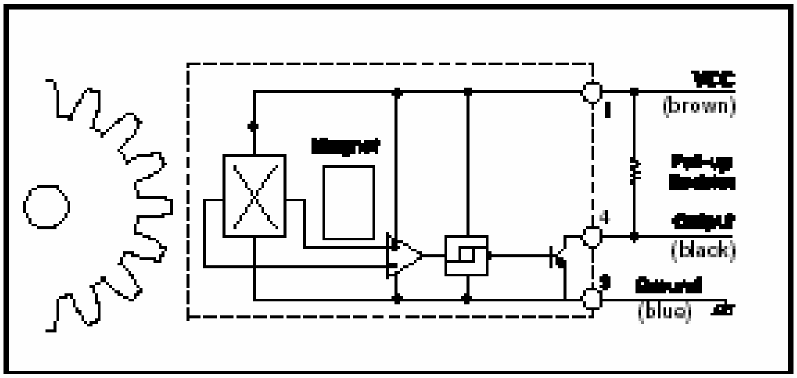


Figure 2. Interface Circuit

Availability and Cost

The GS100701 can be easily acquired online through www.cherrycorp.com as a free sample. If not, the cost is approximately $32 and it is available from major distributors like Digikey and Newark.

Sources:

"Cherry GS Sensors.pdf" Datasheet, www.cherrycorp.com