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

Joshua Phillips

Graffiti-Bot

4-25-2001

Table of Contents

Abstract

This project will focus on moving a robot in a straight line and then extracting some behaviors from this ability. A closed loop system for accurate movement will be implemented in Graffiti-bot. This system must allow for movement in a straight line and precise turns. These behaviors will then allow graffiti-bot to write messages in block letters. This report will discuss all aspects of the robot but will concentrate on the closed loop system for accurate movement.

Introduction

Graffiti-bot will be a simple symmetrical rectangular box platform. Since one of the overall goals of the robot will be to move in a straight line, the location of the wheels, motors, and sensors will be important. Good symmetry about the robot will help reduce but not eliminate the tendency of most platforms to stray from their original path. For example, wheels of unequal radius (even if not noticeable by the human eye) will eventually cause the robot to veer off course and the driving motors will seldom react exactly the same given the same input. When all of these factors are taken into account, the need for a feedback system is quite apparent. The standard behavior of obstacle avoidance will also be implemented. It is anticipated that the feedback system introduced above will eliminate some of the problems with typical avoidance systems.

Integrated System

The brain for Graffiti-bot will be implemented with a Microchip PIC16F877 Microcontroller. Several sensors will be used to gather information about the robots environment. Some of these sensors will be infrared sensors, “bump” switches, and a PS/2 optical mouse for accurate movement. The microcontroller will be used to gather and interpret the information from the sensors about the environment and update the robots behaviors accordingly. A Liquid Crystal Display (LCD) will be mounted on Graffiti-bot and used for troubleshooting and general information display. All of these components will be

Mobile Platform

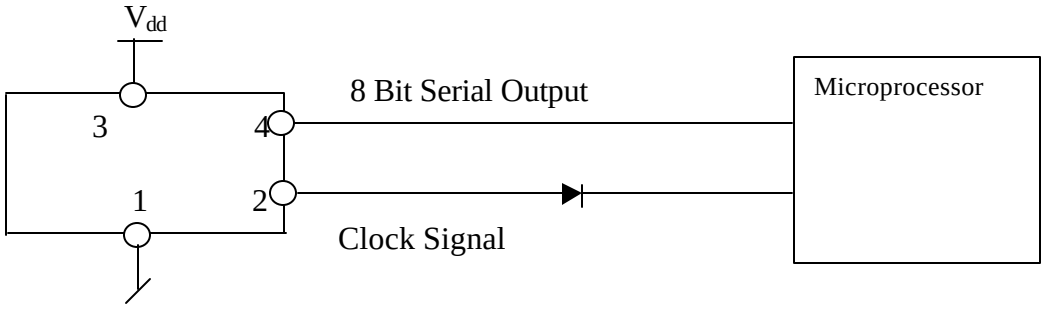
The physical platform for the robot is made of balsa wood and cutout on the “T-tech” machine in lab. The basic structure of the robot will be a simple rectangular box. Graffiti-bot is a two-wheeled robot. The wheels will be rear mounted and the front of the robot will rest on the PS/2 mouse. Accurate measurements must be made, as the mounting of the PS/2 mouse is critical. The motors should also be mounted as symmetrically as possible to aid in moving straight. The wheels are three-inch diameter and were obtained from Mekatronix.

Actuation

Graffiti-bot will use two Nema size 17 stepper motors to drive the wheels. The motors were obtained from Jameco part number 155432. The stepper motors will make navigation and steering corrections easier as a fairly accurate measurement of the number of revolutions can be obtained by counting the number of pulses sent to each motor. This can then be combined with the circumference of each wheel to yield the distance each wheel traveled. To interface the motors to the microprocessor, Allegro Microsystems part number UCN5804B BiMos II Unipolar Stepper-Motor Driver is used. These I.C.'s make an easy 3 wire interface for controlling the motors. One general-purpose output pin for each control signal: motor direction, output enable, and one step input.

Sensors

The main sensors used for collision avoidance are two Sharp part number GP2D02 High Sensitive Distance Measuring Sensors. These each contain an I.R. transmitter, receiver, and a lens that can be changed to alter the range of the device. The Sharp GP2D02 can sample about once every 72 milliseconds and sends it data via a synchronous serial stream. The interface with the I.R. sensor is a fairly simple four-wire interface but it should be noted that the user must supply the clock and that the clock pin on the sensor is an open drain input. Also, the clock line is really a control line and must do more than simply supply a clock signal. For example, to instruct the sensor to take a measurement the clock pin must be held low for 70 milliseconds. One circuit that works well and is simple is shown below.



The diode only allows the microprocessor to pull the open drain clock pin low. The data sheet for the Sharp GP2D02 is included in the appendix.

The main sensor for gathering data about the relative movement of the robot is a Logitech PS/2 optical mouse part number 830386-0000. Using an optical mouse is advantageous because there are no moving parts and it never needs cleaning. Also the resolution of an optical mouse is usually better than a standard ball mouse. As the PS/2 protocol is rather lengthy I will discuss only the information necessary to implement the

PS/2 mouse into a feedback system on a mobile robot. For the remainder of this discussion of the PS/2 protocol and interface the term “host” shall refer to the unit querying the mouse and the term “device” shall refer to the mouse itself.

PS/2 Protocol

The PS/2 protocol is a four-wire interface consisting of power, ground, clock, and data. The clock and data lines are bi-directional open collector signals. Clock and data normally float high and are pulled low by either device or host. The data and clock line are NEVER asserted high by the host or device. The information is transmitted via an 11 bit serial stream consisting of one start bit (low), eight data bits (LSB first), one odd parity bit, and one stop bit (high). Three to four of these 11 bit streams are usually combined into one packet depending on the mode of operation and information requested from the device. Communication can occur from host to device or from device to host. In either case the mouse always provides the clock signal. The data packet format, two types of communication, the electrical interface, and the device initialization are described in detail below.

Data Packet Format

The following packet is the format used in default mode. Other mice may include options to add extra packets for more buttons or features but default mode is fine for this project.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Y ovfl	X ovfl	Y sign	X sign	1	0	R Button	L Button
Byte 2	X7	X6	X5	X4	X3	X2	X1	X0
Byte 3	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0

- Y ovfl : overflow flag for the Y movement accumulator (Byte 3).
  - 1 = an overflow occurred.
  - 0 = an overflow did not occur.
- X ovfl : overflow flag for the X movement accumulator (Byte 2).
  - 1 = an overflow occurred.
  - 0 = an overflow did not occur.
- Y sign : sign bit for Byte 3
  - 1 = Byte 3 is negative (mouse is moving backwards).
  - 0 = Byte 3 is zero or positive (mouse is moving forwards).
- X sign : sign bit for Byte 2
  - 1 = Byte 2 is negative (mouse is moving left).
  - 0 = Byte 2 is zero or positive (mouse is moving right).
- R Button : Right mouse button status bit
  - 1 = right mouse button is pressed.
  - 0 = right mouse button is not pressed.
- L Button : Left mouse button status bit
  - 1 = left mouse button is pressed.

    - 0 = left mouse button is not pressed.

Byte 2 is the amount of movement in the x direction (left or right) the mouse has detected since the last transmission. Byte 3 is the amount of movement in the y direction (forwards or backwards) the mouse has detected since the last transmission.

Host to Device Communication

- To change the mode of the device to "host request to send" mode the host must follow the following procedure:
    - Hold the clock line low for at least 100 microseconds (longer is ok).
    - Pull the data line low.
    - Release the clock line.
- Once the device is in "host request to send" mode it will pulse the clock low 11 times.
- The host changes data when the clock is low.
- The device latches data when the clock is high.
- After sensing a valid stop bit the device will pull data low and pulse the clock line low one more time to indicate that a framing error did not occur.
- If a framing error did occur (invalid stop or parity bit) the device will send the "resend" command (0hFE) to the host.
- If the data transfer was successful the device will respond by sending an acknowledge byte (0hFA) followed by any data the host command requires.

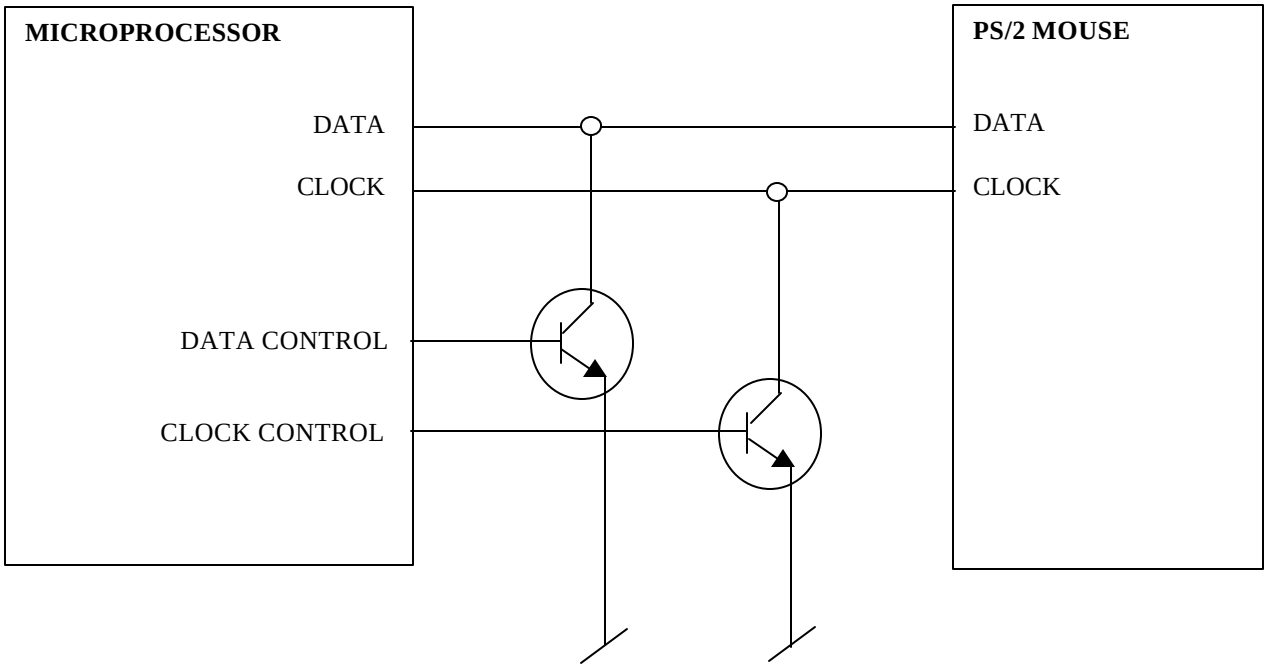
Device to Host Communication

- When the device has information to send to the host it checks the state of the clock and data line.
    - If Data and Clock are both high the bus state is idle and the device will transmit the data.
    - If the Clock line is low the Host is inhibiting transmission and the device will continue to accumulate data and wait for the host to release the clock line.
    - If Clock is high and Data is low the host has requested "host request to send" mode is ready to transmit data to the device.
- If the bus is idle the device will transmit the data by pulsing the clock line low once and then high 11 more times while changing the data line appropriately.
- The device changes the data when the clock is high.
- The host should latch the data when the clock is low.
- If invalid data is detected (bad parity bit) the host can issue the Resend command (0hFE).

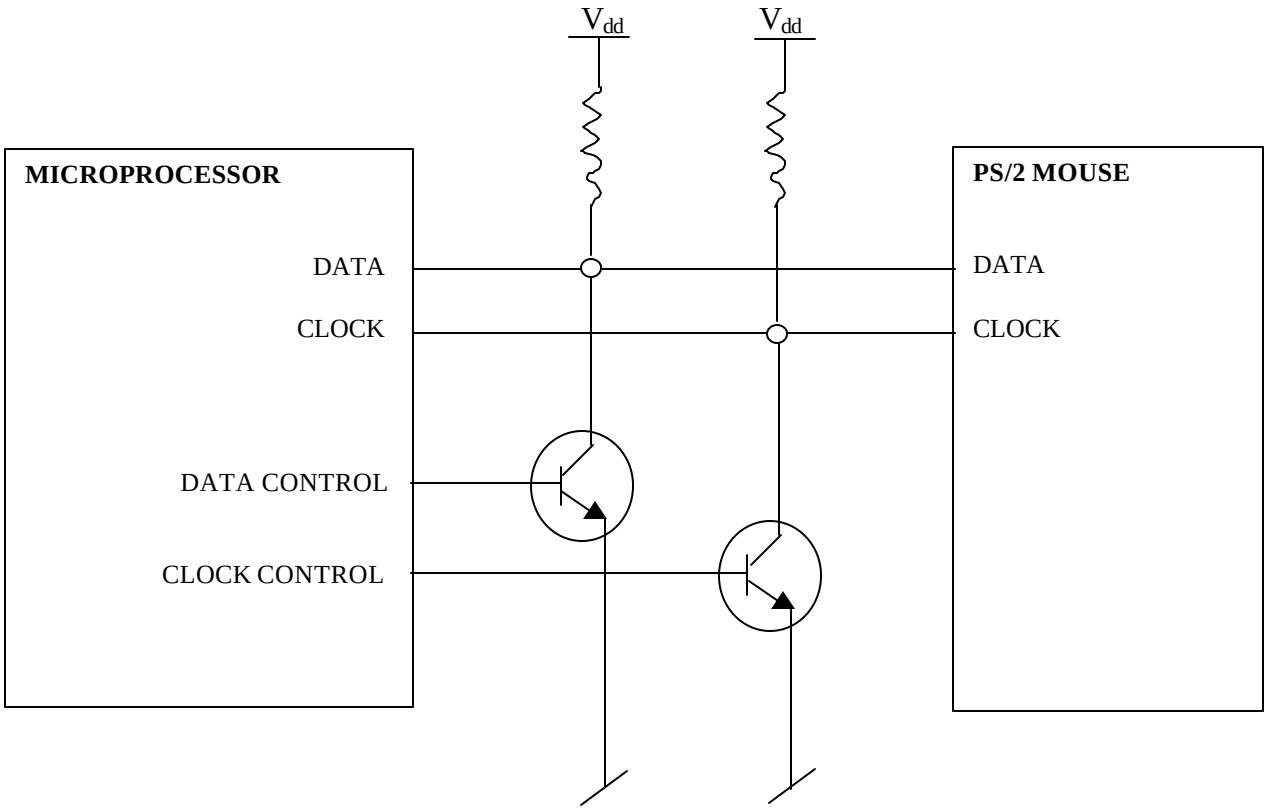
Electrical Interface

As mentioned above the clock and data lines are bi-directional open collector signals. One interface (the one I used) that works well is shown below. This setup requires two general-purpose input pins with internal weak pull-up resistors, two general-purpose

output pins, and two NPN switching transistors. The output pins turn on the switching transistors to pull the appropriate line (clock or data) low. The internal weak pull-up resistors on the microprocessor allow the line to float high when the transistors are off.



If internal weak pull-up resistors are not available then external pull-up resistors (5 to 10 kOhms) can be added as follows:



Device Initialization

The PS/2 protocol allows several modes of operation for a mouse such as Polling Mode or Continuous Stream Mode. I decided to use a controlled continuous stream mode. A few initializations must be made upon powering up or hot-plugging the mouse, as it will remain in disabled mode until given an enable code by the user. Before sending the enable code and setting the mode the host must wait about for the device to complete its power on self-test and calibration. This typically takes between 300 and 1000 milliseconds. Once the self-test and calibration is complete the device will transmit the code 0hAA followed by 0h00. Once this "ready" code is received an enable command (0hF4) must be sent. This will enable the mouse to transmit data in its default mode. The default mode is continuous stream mode (which is the mode I will be using) so no other initialization is needed. A full list of commands and modes is contained in a sample of

pages from a Synaptics PS/2 touch pad data sheet in the appendix of this document. The full document will be included on the floppy disk accompanying this report This document is a good resource for adjusting the resolution of the mouse and troubleshooting.

Useful Hints for Using the PS/2 Mouse

- When enabled, the PS/2 mouse sends a packet of data upon an event. For example, if the mouse is not being moved and no buttons have been pressed or released, the mouse will not transmit any data. Once the mouse detects movement or a button changes state the mouse will transmit the data if the bus is idle (clock and data lines both high). This is important if you are going to poll the device clock during transmission. If the mouse does not change state the clock will remain high permanently and you will be polling for a very long time. A timeout routine should be included if you plan to poll the device clock line.
- If the bus is not idle and the mouse has data to send, the mouse will continue accumulating x and y movement data until the bus is idle. This could result in an overflow of the x or y movement byte if transmission is inhibited by the host for too long. For this reason the overflow bit should always be checked when using the mouse x and y movement data.
- In order not to lose packets, the host should inhibit transmission by pulling the clock line low anytime the host is not monitoring the bus. For example, if the microprocessor is reading the I.R. data and the mouse is moving, the mouse will transmit its data packet if the bus is idle. The processor would miss this unless the clock line was held low in which case the movement would just be added into the next packet of data. (An ideal way to solve this problem is to have a dedicated system reading the mouse continuously, but I went for a one-chip design.)
- Remember the mouse always supplies the clock, once enabled the mouse will always send data packets (event driven) if the bus is idle.
- Any command sent by the host will cause the motion accumulators to be cleared.

Bump Switches

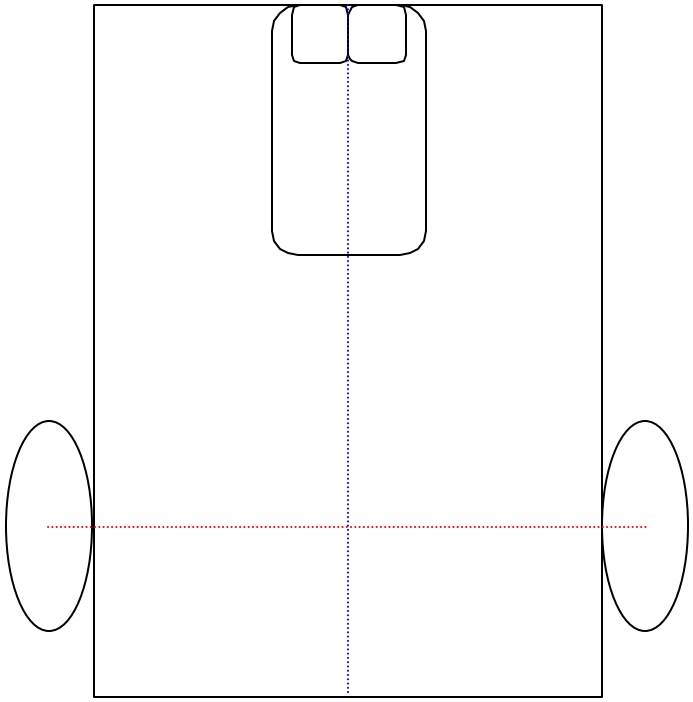
The last sensor used in Graffiti-bot is a normally open single pull single throw switch. The switch is attached to a bumper that runs along portions of the exterior that would be prone to bumping into objects or walls should the other sensors fail. These switches connect a general-purpose input pin to either ground or V_{dd} depending on the state of the switch. Polling the different input pins allows graffiti-bot to determine which side of the robot has collided with an object.

Behaviors

All of the sensors generate stimuli for the robot to respond to. These responses to the sensor input can then be abstracted to behaviors. The first behavior of Graffiti-bot will be obstacle avoidance. The robot must be able to protect itself from injury and must also ensure it does not injure other objects. This behavior is accomplished via the Sharp I.R.

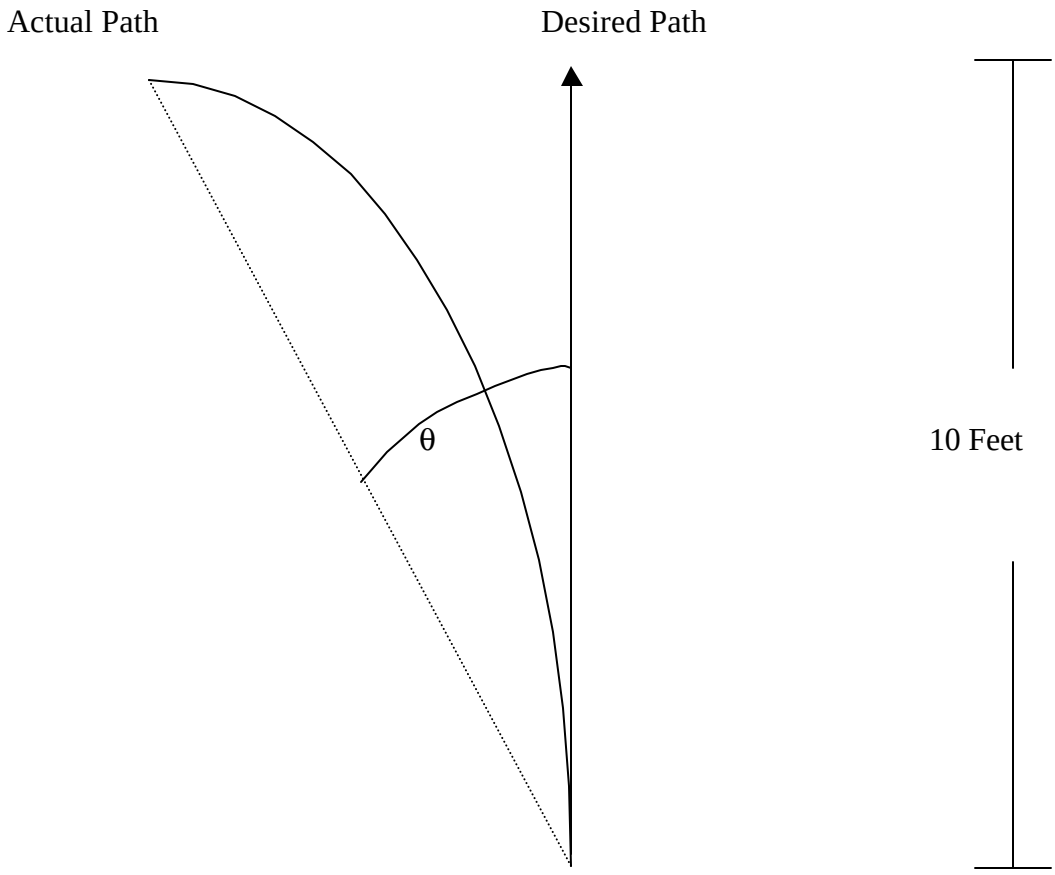
sensors. Taking samples from the I.R. once every 70 milliseconds, the I.R. data is tested against a fixed value (or distance). When the object is closer than the preset threshold, the robot will choose a random course of action (except, of course, continuing toward the object).

Another behavior that Graffiti-bot is capable of is moving in a straight line. This is accomplished using the PS/2 mouse. The X accumulator is sampled about 100 times a second along with the "X sign" bit to tell the direction and magnitude of deviation from a straight line. Depending on the direction and magnitude, the appropriate wheel speed is reduced or increased as necessary until the robot moves straight. It is important to align the mouse accurately as a misaligned mouse will yield false readings and cause the robot to move in circles or other undesired motion. The mouse should be mounted with the bisecting line along the vertical of the mouse aligned with the bisecting line along the vertical of the robot as seen from a birds eye view in the following diagram:

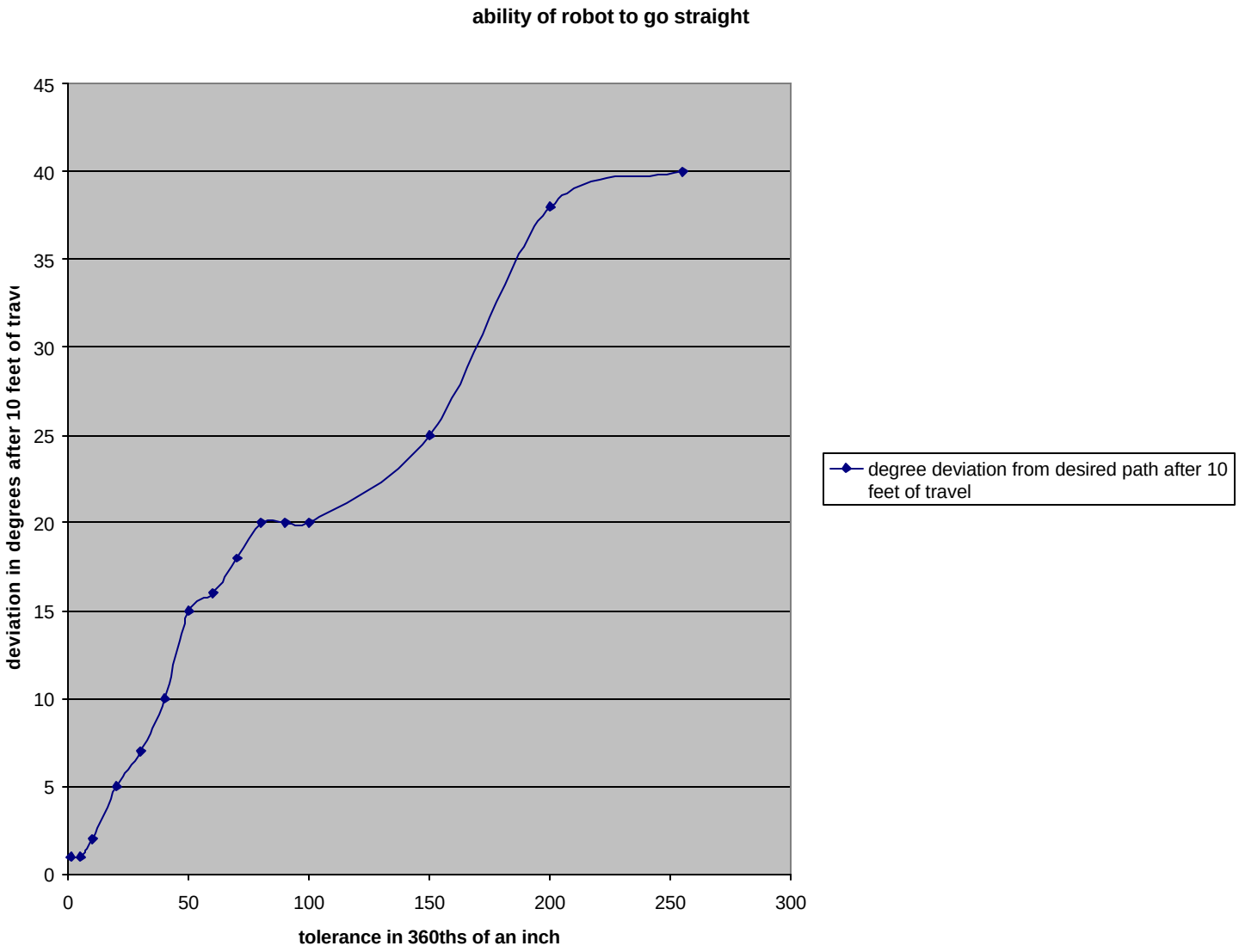


Note: The mouse can be mounted anywhere along the blue line. The preferable location has the center of the mouse "eye" located half the distance of the red line above the intersection of the red and blue lines.

To fine-tune the control system, an error much larger than would be experienced during normal operation was forced into the system. The feedback system variables were changed incrementally and the deviation (measured in degrees) from the desired path after 10 ft was recorded. The test would be similar to the following diagram:



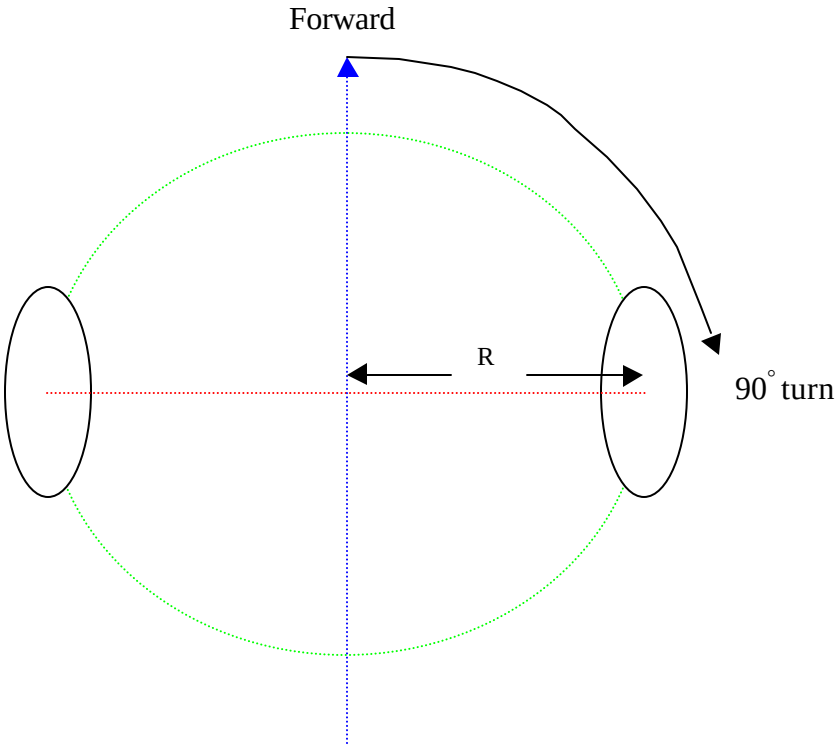
After several tests the following graph was formed:



Note: The point at zero is invalid as it causes oscillation and other unwanted behavior.

The final result of the experiment was that the optimal threshold for comparison with the X magnitude was between 0h06 and 0h07. This value will increase or decrease depending on the alignment of the mouse, symmetry in the mounting of the motors, wheel alignment and wheel similarity. Using a final result of seven in the threshold register, the robot was able to travel 10 feet with a deviation of one degree or less from the desired path.

The last behavior Graffiti-bot exhibits is message writing. By attaching a solenoid-controlled pen to the robot platform simple text can be written in block letters. The ability to make an accurate 90-degree turn is essential to make block letters. This can easily accomplished with the stepper motors and checked with the mouse. Using the diagram below the correct number of steps needed to turn the robot 90-degrees about the wheel center can be calculated.



After the 90 degree turn each wheel has traveled a distance of R*π/2 along the dotted green line. The number of steps is then calculated as:

Number of steps = (R * π/2) / (number of degrees per step * wheel radius)

The robot will have a difficult time making an exact 90-degree turn due to the discreetness of the stepper motors. The 1.8 degree per step motors on Graffiti-bot were able to make an 89.5 degree turn. This is accurate enough for the purposes of this project, however, after several turns the accumulated error begins to show. This can be rectified by turning 90.5 degrees half of the time and turning 89.5 degrees the other half of the time. With these details implemented, Graffiti-bot successfully writes messages.

Conclusion

Graffiti-bot is a success. The ability to drive straight was successful and provided the means to develop the other behaviors. However, the method of integration used amongst the sensors and peripheral devices could be improved. Using a single processor to control every device on the robot is space saving but complicates the software development. Using a dedicated system for each sensor or peripheral device removes some of the burden from the main processor and makes software development much easier. If done again, I wouldn't hesitate to adopt this philosophy.

```
;       Graffiti-bot Code for IMDL spring 2001 Prof Arroyo by Joshua Phillips
LIST    P=PIC16F877     ;
        include "p16f877.inc"  ;
;*****************************************************************************
;       LABELS
;*****************************************************************************
#define  IRFDAT PORTB,0x07     ;
#define  IRFCLK PORTB,0x06     ;
#define  MSDATA PORTB,0x05     ;
#define MSCLOCK PORTB,0x04     ;
#define  MSCCTRL        PORTB,0x03     ;
#define  MSDCTRL        PORTB,0x02     ;
#define  IRRDAT PORTB,0x01     ;
#define  IRRCLK PORTB,0x00     ;
#define  MOTORL PORTA,0x00     ;
#define  MOTORR PORTA,0x01     ;
#define MOTL_OE PORTA,0x02     ;
#define MOTR_OE PORTA,0x03     ;
#define MOTR_DR PORTC,0x07     ;
#define MOTL_DR PORTA,0x05     ;
#define RS_LQD  PORTC,0x00     ;
#define RW_LQD  PORTC,0x01     ;
#define E_LQD   PORTC,0x02     ;
#define BUMP_F  PORTC,0x06     ;
#define  BUMP_L PORTC,0x05     ;
#define  BUMP_R PORTC,0x04     ;
#define MARKER  PORTC,0x03     ;
#define LQD     PORTD          ;
#define BUSYFL  PORTD,0x07     ;
#define  Y_OVER BUTTONS,0x07   ;
#define X_OVER  BUTTONS,0x06   ;
#define Y_SIGN  BUTTONS,0x05   ;
#define X_SIGN  BUTTONS,0x04   ;
#define R_BUT   BUTTONS,0x01   ;
#define L_BUT   BUTTONS,0x00   ;
COUNT   equ     0x24           ;
COUNT1  equ     0x25           ;
COUNT2  equ     0x26           ;
MSINFO  equ     0x27           ; no init needed
PARITY  equ     0x28           ; no init needed
LQDDATA equ     0x29           ; no init needed
REVTEMP equ     0x2A           ; no init needed
REVTMP2 equ     0x2B           ; no init needed
REVTMP3 equ     0x2C           ; no init needed
REVTMP4 equ     0x2D           ; no init needed
TIMEOUT equ     0x2F           ; no init needed
INCNT   equ     0x30           ; no init needed
OUTCNT  equ     0x31           ; no init needed
RWT     equ     0x32           ; Period for Right wheel pulse (8 Bit)
LWTH    equ     0x33           ; Period for Left wheel upper byte
LWTL    equ     0x34           ; Period for left wheel lower byte
TM2CNT  equ     0x35           ; 70 ms timer in TIMER2 interrupt routine (IR)
IRDATA  equ     0x36           ; no init needed
IRTEST  equ     0x37           ; flag for IR routines (indicates a measurement has been
                                 started)
IR      equ     0x38           ; flag for main routine to validate data / flag for IR
                                 routines to choose front or rear IR
IRRDATA equ     0x39           ; no init needed
LW_CNT  equ     0x3A           ; counters for mouse to be read once every LW_CNT times the
                                 wheel is turned
RW_CNT  equ     0x3B           ; counters for LCD to be updated once every RW_CNT times
                                 the wheel is turned
STCOUNT equ     0x3C           ; timeout counter for mouse routine
IDLE_MS equ     0x3D           ; flag for main routine to recognize a timeout, no init
                                 needed
BUTTONS equ     0x3E           ; mouse data
X_MAG   equ     0x3F           ; mouse data
Y_MAG   equ     0x40           ; mouse data
RAND    equ     0x41           ; a random number
TOUT_2  equ     0x42           ; another timeout counter
TOUT_3  equ     0x43           ; another timeout counter
```

```
PULSES  equ     0x44            ; for navigation
DIST    equ     0x45            ; for navigation
NAV_OK  equ     0x46            ; for nav init to 0x00
SECONDS equ     0x47            ;
ODELAY  equ     0x48            ;
IDELAY  equ     0x49            ;
NDELAY  equ     0x4A            ;
MS_CNTR equ     0x4B            ; init to 3
MS_CNTL equ     0x4C            ; init to 3
RADIUS  equ     0x4D            ;
RPULSES equ     0x4E            ;
LPULSES equ     0x4F            ;
RWT_TMP equ     0x50            ;
LWTHTMP equ     0x51            ;
LWTLTMP equ     0x52            ;
ACCRCNT equ     0x53            ;
ACCLCNT equ     0x54            ;
TOPSPD  equ     0x55            ;
MAX_ACC equ     0x56            ;
MS_USE  equ     0x57            ;
MALIGNB equ     0x58            ;
MALIGNF equ     0x59            ;
FIRST_F equ     0x5A            ;
W_TEMP  equ     0x70            ; temporary register for w available in all banks
S_TEMP  equ     0x71            ; temp reg for the status reg (all banks)
        org     0x00            ;       Set RESET vector
        goto    INIT            ;       to beginning of program.
        org     0x04            ;       set INTERRUPT vector to
        goto    INTRUPT         ;       beginning of INTERRUPT service routine
        org     0x05            ;       start of program
INIT    bcf     STATUS,RP0      ;       select bank 0
        bcf     STATUS,RP1      ;       select bank 0
;*****************************************************************************
;       INTIALIZE REGISTERS / PORTS / INTERUPTS
;*****************************************************************************
;*****************************************************************************
;       INTERRUPT CONTROL REGISTER
;       initialize to:
;       GLOBAL INTERRUPT ENABLE              : DISABLED     0
;       PERIPHERAL INTERRUPT ENABLE   : ENABLED      1
;       TIMER0 OVERFLOW INTERRUPT     : ENABLED      1
;       EXTERNAL INTERRUPT PIN        : DISABLED     0
;       PORTB INTERRUPT ON CHANGE     : DISABLED     0
;       LOWER 3 BITS ARE FLAGS               : DONT CARE    xxx
;*****************************************************************************
        movlw   0x60            ;       initialize interrupt control reister
        movwf   INTCON          ;
;*****************************************************************************
;*****************************************************************************
;       OPTION REGISTER
;       initialize to:
;       PORTB INTERNAL PULLUP RESISTORS      : ENABLED      0
;       EXTERNAL INTERRUPT EDGE SELECT       : RISING EDGE  1
;       TIMER0 CLOCK SOURCE SELECT    : Fosc/4       0
;       TIMER0 SOURCE EDGE SELECT     : low to high  0
;       PRESCALER ASSIGNMENT          : TIMER0       0
;       LOWER 3 BITS ARE PRESCALER    : 1/256        111
;*****************************************************************************
        bsf     STATUS,RP0      ;       select bank 1
        bcf     STATUS,RP1      ;       select bank 1
        movlw   0x47            ;
        movwf   OPTION_REG      ;       initialize option register
;*****************************************************************************
;       PERIPHERAL INTERRUPT REGISTER 1
;       initialize to:
;       PARALLEL SLAVE PORT R/W INTERRUPT ENABLE     : DISABLED     0
;       A/D CONVERTER INTERRUPT ENABLE                       : DISABLED     0
;       USART RECIEVE INTERRUPT ENABLE                       : DISABLED     0
;       USART TRANSMIT INTERRUPT ENABLE                      : DISABLED     0
;       SYNCHRONOUS SERIAL PORT INTERRUPT ENABLE     : DISABLED     0
;       CAPTURE/COMPARE 1 INTERRUPT ENABLE           : DISABLED     0
```

```
;       TIMER2 TO PR2 MATCH INTERRUPT                : ENABLED      1
;       TIMER1 OVERFLOW INTERRUPT ENABLE             : ENABLED      1
;*******************************************************************************
        movlw   0x03    ;
        movwf   PIE1    ; initialize PIE1 register
;*******************************************************************************
;       PERIPHERAL INTERRUPT REGISTER 2
;       initialize to:
;       BIT 7-5                                 : ALWAYS 000
;       EEPROM WRITE INTERRUPT ENABLE           : DISABLED 0
;       BUS COLLISION INTERRUPT ENABLE                  : DISABLED 0
;       BIT 2-1                                 : ALWAYS 00
;       CAPTURE COMPARE 2 INTERRUPT ENABLE      : DISABLED 0
;*******************************************************************************
        movlw   0x00;
        movwf   PIE2; initialize PIE2 register
;*******************************************************************************
;       PORT A (6 bits wide)
;       Analog input 0 thru 4 are on this port along with
;       TIMER0's external clock input pin
;       A0 will be the left motor signal
;       A1 will be the right motor signal
;       A2 will be the motor driver ouput enable signal (active low)
;       A4 and A5 will be motor direction pins
;       All other pins are inputs
;*******************************************************************************
        movlw   0x00            ;
        movwf   TRISA           ; data direction register for PORTA
;*******************************************************************************
;       PORT B (8 bits wide)
;       General purpose in-out.
;       Bit 0 and 1 will control the rear IR.
;       Bit 2 will control the mouse data line.
;       Bit 3 will control the mouse clock line.
;       Bit 4 will be used as mouse clock.
;       Bit 5 will be used as mouse data.
;       Bit 6 will be the IR front clock.
;       Bit 7 will be the IR front data.
;       All pins will be used as inputs.
;*******************************************************************************
        movlw   0xB2            ;
        movwf   TRISB           ; data direction register for PORTB
;*******************************************************************************
;       PORT C (8 bits wide)
;       General purpose in-out, PWM output, SPI, USART, CAPTURE 1&2
;       PORT C PINS 0,1,2 will be control lines for the LQD
;       PORT C PIN 7 will be MOTOR LEFT DIRECTION
;*******************************************************************************
        movlw   0x70            ; 0111 0000
        movwf   TRISC           ; data direction register for PORTC
;*******************************************************************************
;       PORT D (8 bits wide)
;       Port D can be used as a parallel slave port or general in-out
;       This port will send data to the LQD
;       all pins are outputs
;*******************************************************************************
        movlw   0x00            ; 0000 0000
        movwf   TRISD           ; data direction register for PORTD
;*******************************************************************************
;       PORT E (3 bits wide)
;       Port E can be used as general purpose in-out, Analog input 7 thru 5,
;       or as the control bits for parallel slave port mode.
;*******************************************************************************
        movlw   0x07            ;
        movwf   TRISE           ; data direction register for PORTE
;*******************************************************************************
;       ANALOG-TO-DIGITAL REGISTER 1
;       This register selects the port configurations for analog or digital
;       input and selects the values for Vref+
;       This register also right/left justifies the A/D result register
;       The result is 10 bits wide in a 16 bit wide register.
```

```
;       The pins will be used as follows:
;       PORT E A/D pins = Digital I/O
;       PORT A  PIN0    = Digital I/O
;               PIN1    = Digital I/O
;               PIN2    = Digital I/O
;               PIN3    = Digital I/O
;               PIN4    = Digital I/O
;               PIN5    = Digital I/O
;*******************************************************************************
;
        movlw   0x06            ;
        movwf   ADCON1          ; config A/D
;*******************************************************************************
;       ANALOG-TO-DIGITAL REGISTER 0
;       This register is mainly used to start and stop the conversions
;       and select which analog input is to be used for the next conversion.
;       note:   The required pause before the next acquisition can begin is
;               2*the value selected in bits 7 and 6 - in this case Fosc/32
;               would yield about .8us @ 20Mhz
;*******************************************************************************
;
        bcf     STATUS,RP0      ;
        bcf     STATUS,RP1      ; SELECT BANK 0
        bsf     ADCON0,0x07     ;
        bcf     ADCON0,0x06     ; conversion clock set to Fosc/32
        bcf     ADCON0,0x05     ;
        bcf     ADCON0,0x04     ;
        bcf     ADCON0,0x03     ; select analog input 0
        bcf     ADCON0,0x00     ; turn off the A/D converter
;*******************************************************************************
;       TIMER2  Setup
;*******************************************************************************
;
        movlw   0x7F            ; prescale of 16
        movwf   T2CON           ; post scale of 16
;*******************************************************************************
;       TIMER1 SETUP
;       Timer1 will be used to generate the pulses for the motors
;       prescale = 1:1
;*******************************************************************************
;
        movlw   0x01            ;       0011 0101
        movwf   T1CON           ;
;*******************************************************************************
;       MAIN PROGRAM
;*******************************************************************************
;
        call    MS_TX           ; Initialize PS/2 Mouse and write data to LQD
        call    LQDINIT         ; initialize Liquid Crystal Display
        bsf     IRFCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        clrf    IRTEST          ;
        clrf    IR              ;
        clrf    X_MAG           ;
        clrf    Y_MAG           ;
        clrf    BUTTONS         ;
        bsf     MOTL_OE         ;
        bsf     MOTR_OE         ;
        bcf     MOTL_DR         ; init to move forward
        bsf     MOTR_DR         ;
        movlw   0x00            ;
        movwf   LWTH            ;
        movwf   LWTL            ;
        movwf   RWT             ;
        movlw   0x06            ;
        movwf   TM2CNT          ; 70 ms timer for IR routine
        movlw   0x03            ;
        movwf   MS_CNTL         ;
        movwf   MS_CNTR         ;
        clrf    RWT_TMP         ;
        clrf    LWTHTMP         ;
        clrf    LWTLTMP         ;
        movlw   0x07            ;
        movwf   ACCRCNT         ;
        movwf   ACCLCNT         ;
        movlw   0xFF            ;
```

```
        movwf   TOPSPD          ;
        movlw   0x03            ;
        movwf   MAX_ACC         ;
        bcf     MS_USE,0x00     ; (mouse is disabled)
        movlw   0x10            ;
        movwf   SECONDS         ;
        movlw   0x38            ;
        movwf   MALIGNB         ;
        movlw   0x3B            ;
        movwf   MALIGNF         ;
        bsf     MARKER          ; lift up pen
;-------------------------------------------------------------------------------------------
----------
; X_MAG AND Y_MAG are 9 bit signed 2's compliment numbers
; the 9th bit is the Y_sign or X_sign bit below in the button register
;-------------------------------------------------------------------------------------------
----------
;BUTTONS REGISTER
;
;BIT    7               6               5       4       3       2       1               0
;
;NAME   Y_OVER          X_OVER          Y_SIGN  X_SIGN  1       0       R_BUT           L_BUT
;
;IF 1   overflow        overflow        DOWN    LEFT    1       0       PRESSED         PRESSED
;
;IF 0   no overflow     no overflow     UP      RIGHT   1       0       NOT PRESSED     NOT
PRESSED
;
; (LEFT MOTOR 0 = forward, 1 = reverse) (RIGHT MOTOR 0 = reverse, 1 = reverse)
;-------------------------------------------------------------------------------------------
----------
WAIT4ME goto    HERE            ;
        clrf    BUTTONS         ;
        call    MS_INFO         ; wait until the left mouse button is pushed before
        btfsc   BUTTONS,0x00    ; doing anything.
        goto    LEFTMB          ;
        btfsc   BUTTONS,0x01    ;
        goto    RIGHTMB         ;
        goto    WAIT4ME         ;
LEFTMB  bsf     INTCON,GIE      ; enable unmasked interrupts
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        bcf     MOTL_OE         ;
        bcf     MOTR_OE         ;
HERE    bsf     INTCON,GIE      ; enable unmasked interrupts
        bcf     MOTL_OE         ;
        bcf     MOTR_OE         ;
        bcf     MS_USE,0x00     ; disable mouse
        movlw   0x00            ;
        movwf   LWTH            ;
        movwf   LWTL            ;
        movwf   RWT             ;
        btfss   MOTL_DR         ; check direction of LEFT MOTOR
        goto    L_FOR           ;
        goto    L_REV           ;
L_FOR   btfsc   MOTR_DR         ; LEFT MOTOR is going forwards check RIGHT MOTOR
        goto    FORWARD         ; BOTH MOTORS FORWARD
        goto    R_TURN          ; RIGHT TURN
L_REV   btfsc   MOTR_DR         ; LEFT MOTOR is going backwards check RIGHT MOTOR
        goto    L_TURN          ; LEFT TURN
        goto    REVERSE         ; REVERSE
;-------------------------------------------------------------------------------
FORWARD bcf     IR,0x00         ; choose front IR
        btfss   BUMP_F          ;
        goto    OPT_3           ; if front collision go backward
        btfss   BUMP_L          ; if left collision go backward
        goto    OPT_3           ;
        btfss   BUMP_R          ; if right collision go backward
        goto    OPT_3           ;
        btfsc   X_OVER          ; if an overflow occured do nothing
        goto    STRT            ;
        btfsc   X_SIGN          ;
```

```
        goto    MOVINGL         ; robot is moving left / increase left motor speed?
        goto    MOVINGR         ; robot is moving right or straight / increase right motor
                                  speed?
MOVINGR movlw   0x33            ; check X_MAG from mouse if less than w-reg dont do
                                  anything
        subwf   X_MAG,0x00      ; if greater than w-reg adjust motor times to correct
                                  steering
        btfss   STATUS,0x00     ;
        goto    STRT            ;
        movlw   0x01            ;
        addwf   RWT,0x01        ;
        clrf    X_MAG           ;
        goto    STRT            ;
MOVINGL movlw   0xFF            ; take 2's compliment
        xorwf   X_MAG,0x01      ; flip the bits
        incf    X_MAG,0x01      ; add 1
        movlw   0x33            ; check X_MAG from mouse if less than w-reg dont do
                                  anything
        subwf   X_MAG,0x00      ; if greater than w-reg adjust motor times to correct
                                  steering
        btfss   STATUS,0x00     ;
        goto    STRT            ;
        movlw   0x01            ;
        addwf   LWTL,0x01       ;
        clrf    X_MAG           ;
        goto    STRT            ;
STRT    btfsc   IR,0x07         ; check if new data being aquired
        goto    $ - 1           ;
        btfss   IR,0x07         ; check if data is valid
        goto    $ - 1           ;
        movlw   0x83            ;
        subwf   IRDATA,0x00     ; check if object is near
        btfss   STATUS,0x00     ;
        goto    HERE            ;
        movf    TMR0,0x00       ; get a random number
        movwf   RAND            ; store random number
        sublw   0x55            ; check for a range of 0 to 85
        btfsc   STATUS,0x00     ; check if a borrow occured
        goto    OPT_1           ; if between 0 and 85 choose option 1
        movf    RAND,0x00       ; get same number again
        sublw   0xAA            ; check for a range of 85 to 170
        btfsc   STATUS,0x00     ; check if a borrow occured
        goto    OPT_2           ; if between 85 and 170 choose option 2
        goto    OPT_3           ; if between 170 and 255 choose option 3
OPT_1   bcf     MOTL_DR         ; turn right
        bcf     MOTR_DR         ;
        call    DELAY           ;
        call    DELAY           ;
        goto    HERE            ;
OPT_2   bsf     MOTL_DR         ; turn left
        bsf     MOTR_DR         ;
        call    DELAY           ;
        call    DELAY           ;
        goto    HERE            ;
OPT_3   bsf     MOTL_DR         ; move in reverse mode
        bcf     MOTR_DR         ;
        call    DELAY           ;
        call    DELAY           ;
        goto    HERE            ;
REVERSE bsf     IR,0x00         ; choose rear IR
        btfsc   IR,0x07         ; check if new data being aquired
        goto    $ - 1           ;
        btfss   IR,0x07         ; check if data is valid
        goto    $ - 1           ;
        movlw   0x83            ;
        subwf   IRRDATA,0x00    ; check if object is near
        btfss   STATUS,0x00     ;
        goto    HERE            ;
        movf    TMR0,0x00       ; get a random number
        movwf   RAND            ; store random number
        sublw   0x55            ; check for a range of 0 to 85
```

```
        btfsc   STATUS,0x00     ; check if a borrow occured
        goto    OPT_R1          ; if between 0 and 85 choose option 1
        movf    RAND,0x00       ; get same number again
        sublw   0xAA            ; check for a range of 85 to 170
        btfsc   STATUS,0x00     ; check if a borrow occured
        goto    OPT_R2          ; if between 85 and 170 choose option 2
        goto    OPT_R3          ; if between 170 and 255 choose option 3
OPT_R1  bcf     MOTL_DR         ; turn right
        bcf     MOTR_DR         ;
        call    DELAY           ;
        call    DELAY           ;
        goto    HERE            ;
OPT_R2  bsf     MOTL_DR         ; turn left
        bsf     MOTR_DR         ;
        call    DELAY           ;
        call    DELAY           ;
        goto    HERE            ;
OPT_R3  bcf     MOTL_DR         ; move forward
        bsf     MOTR_DR         ;
        call    DELAY           ;
        call    DELAY           ;
        goto    HERE            ;
R_TURN  bcf     IR,0x00         ; choose front IR
        btfss   BUMP_L          ; if left collision go backward
        goto    OPT_3           ;
        btfss   BUMP_R          ; if right collision go backward
        goto    OPT_3           ;
        btfsc   IR,0x07         ; check if new data being aquired
        goto    $ - 1           ;
        btfss   IR,0x07         ; check if data is valid
        goto    $ - 1           ;
        movlw   0x83            ;
        subwf   IRDATA,0x00     ; check if object is near
        btfsc   STATUS,0x00     ;
        goto    HERE            ;
        bcf     MOTL_DR         ; move forward
        bsf     MOTR_DR         ;
        call    DELAY           ;
        call    DELAY           ;
        goto    HERE            ;
L_TURN  bcf     IR,0x00         ; choose front IR
        btfss   BUMP_L          ; if left collision go backward
        goto    OPT_3           ;
        btfss   BUMP_R          ; if right collision go backward
        goto    OPT_3           ;
        btfsc   IR,0x07         ; check if new data being aquired
        goto    $ - 1           ;
        btfss   IR,0x07         ; check if data is valid
        goto    $ - 1           ;
        movlw   0x83            ;
        subwf   IRDATA,0x00     ; check if object is near
        btfsc   STATUS,0x00     ;
        goto    HERE            ;
        bcf     MOTL_DR         ; move forward
        bsf     MOTR_DR         ;
        call    DELAY           ;
        call    DELAY           ;
        goto    HERE            ;
;-------------------------------------------------------------------------------
;       THIS IS THE GRAFFITTI BEHAVIOR (robot will write GO GATORS)
;-------------------------------------------------------------------------------
RIGHTMB call    LETTERG         ;
        call    LETTERO         ;
        call    BLANK_S         ;
        call    LETTERG         ;
        call    LETTERA         ;
        call    LETTERT         ;
        call    LETTERO         ;
        call    LETTERR         ;
        call    LETTERS         ;
        goto    WAIT4ME         ;
```

```
;*******************************************************************************
;       INTERRUPT ROUTINE
;*******************************************************************************
INTRUPT movwf   W_TEMP          ; save w reg contents
        btfss   PIR1,TMR1IF     ; Check if Timer 1 interrupt
        goto    NEXT1           ;
        bcf     PIR1,TMR1IF     ; clear flag
LEFT_M  btfss   MOTORL          ; check if left motor pulse is high or low
        goto    MAKELH          ; if low make high
        goto    MAKELL          ; if high make low
MAKELH  bsf     MOTORL          ;
        btfsc   NAV_OK,0x00     ;
        decf    LPULSES,0x01    ;
        goto    LSETT           ; set pulse width
MAKELL  bcf     MOTORL          ;
        goto    LSETT           ; set pulse width
LSETT   bsf     NAV_OK,0x01     ; tell ACC routine which motor is being updated
        call    ACC             ;
        movf    LWTHTMP,0x00    ;
        movwf   TMR1H           ;
        movf    LWTLTMP,0x00    ;
        movwf   TMR1L           ;
        goto    INT_END         ;
;-------------------------------------------------------------------------------------------
---------
NEXT1   btfss   INTCON,0x02     ; Check if Timer 0 interrupt
        goto    NEXT2           ;
        bcf     INTCON,0x02     ; clear flag
RIGHT_M btfss   MOTORR          ; check if right motor pulse is high or low
        goto    MAKERH          ; if low make high
        goto    MAKERL          ; if high make low
MAKERH  bsf     MOTORR          ;
        btfsc   NAV_OK,0x00     ;
        decf    RPULSES,0x01    ;
        goto    RSETT           ; set pulse width
MAKERL  bcf     MOTORR          ;
        goto    RSETT           ; set pulse width
RSETT   bcf     NAV_OK,0x01     ; tell ACC routine which motor is being updated
        call    ACC             ;
        movf    RWT_TMP,0x00    ;
        movwf   TMR0            ;
        goto    INT_END         ;
;-------------------------------------------------------------------------------------------
---------
NEXT2   btfss   PIR1,TMR2IF     ; Check if Timer 2 interrupt
        goto    INT_END         ;
        bcf     PIR1,TMR2IF     ; clear flag
        btfss   MS_USE,0x00     ;
        goto    IR_RD           ;
        decfsz  MS_CNTR,0x01    ;
        goto    IR_RD           ;
        movlw   0x04            ;
        movwf   MS_CNTR         ;
        call    MS_INFO         ;
        call    X_CURS          ;
        movf    X_MAG,0x00      ;
        movwf   REVTEMP         ;
        call    REV_ASC         ;
        movf    REVTMP3,0x00    ;
        movwf   LQDDATA         ;
        call    LQD_SND         ;
        movf    REVTMP4,0x00    ;
        movwf   LQDDATA         ;
        call    LQD_SND         ;
        call    Y_CURS          ;
        movf    Y_MAG,0x00      ;
        movwf   REVTEMP         ;
        call    REV_ASC         ;
        movf    REVTMP3,0x00    ;
        movwf   LQDDATA         ;
        call    LQD_SND         ;
```

```
        movf    REVTMP4,0x00    ;
        movwf   LQDDATA         ;
        call    LQD_SND         ;
        goto    INT_END         ;
IR_RD   btfsc   IR,0x00         ; check is front IR (0) or rear IR (1) should be checked
        goto    REARIR          ;
        goto    FRONTIR         ;
FRONTIR bcf     IR,0x07         ; for main loop to wait for valid data
        btfsc   IRTEST,0x00     ; check if measurement has been initiated
        goto    TESTOK          ; if yes go check if measurement is finished else proceed
                                  with init
        decfsz  TM2CNT,0x01     ; wait for 70 ms before reading I.R.
        goto    INT_END         ;
        movlw   0x06            ; reset 70 ms counter
        movwf   TM2CNT          ;
        bcf     IRFCLK          ; make clock signal low to initiate a measurement
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        btfsc   IRFDAT          ; check for data signal to go low (ack that a measurement
                                  is in progress)
        goto    $ - 1           ;
        bsf     IRTEST,0x00     ; set flag to indicate a measurement has been started
        goto    INT_END         ;
TESTOK  btfss   IRFDAT          ; check for measurement complete
        goto    INT_END         ; else end interrupt so other routines may continue
                                  processing
        bsf     IRFCLK          ; start bit
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        call    DELAY           ;
        bcf     IRFCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        call    DELAY           ;
IR7     bsf     IRFCLK          ; bit 7
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        btfss   IRFDAT          ;
        goto    IRLOW7          ;
        goto    IRHIGH7         ;
IRLOW7  bcf     IRDATA,0x07     ;
        goto    IR6             ;
IRHIGH7 bsf     IRDATA,0x07     ;
        goto    IR6             ;
IR6     call    DELAY           ; bit 6
        bcf     IRFCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        call    DELAY           ;
        bsf     IRFCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        btfss   IRFDAT          ;
        goto    IRLOW6          ;
        goto    IRHIGH6         ;
IRLOW6  bcf     IRDATA,0x06     ;
        goto    IR5             ;
IRHIGH6 bsf     IRDATA,0x06     ;
        goto    IR5             ;
IR5     call    DELAY           ; bit 5
        bcf     IRFCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        call    DELAY           ;
        bsf     IRFCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        btfss   IRFDAT          ;
        goto    IRLOW5          ;
        goto    IRHIGH5         ;
IRLOW5  bcf     IRDATA,0x05     ;
        goto    IR4             ;
IRHIGH5 bsf     IRDATA,0x05     ;
        goto    IR4             ;
IR4     call    DELAY           ; bit 4
        bcf     IRFCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        call    DELAY           ;
        bsf     IRFCLK          ;
```

```
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        btfss   IRFDAT          ;
        goto    IRLOW4          ;
        goto    IRHIGH4         ;
IRLOW4  bcf     IRDATA,0x04     ;
        goto    IR3             ;
IRHIGH4 bsf     IRDATA,0x04     ;
        goto    IR3             ;
IR3     call    DELAY           ; bit 3
        bcf     IRFCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        call    DELAY           ;
        bsf     IRFCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        btfss   IRFDAT          ;
        goto    IRLOW3          ;
        goto    IRHIGH3         ;
IRLOW3  bcf     IRDATA,0x03     ;
        goto    IR2             ;
IRHIGH3 bsf     IRDATA,0x03     ;
        goto    IR2             ;
IR2     call    DELAY           ; bit 2
        bcf     IRFCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        call    DELAY           ;
        bsf     IRFCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        btfss   IRFDAT          ;
        goto    IRLOW2          ;
        goto    IRHIGH2         ;
IRLOW2  bcf     IRDATA,0x02     ;
        goto    IR1             ;
IRHIGH2 bsf     IRDATA,0x02     ;
        goto    IR1             ;
IR1     call    DELAY           ; bit 1
        bcf     IRFCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        call    DELAY           ;
        bsf     IRFCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        btfss   IRFDAT          ;
        goto    IRLOW1          ;
        goto    IRHIGH1         ;
IRLOW1  bcf     IRDATA,0x01     ;
        goto    IR0             ;
IRHIGH1 bsf     IRDATA,0x01     ;
        goto    IR0             ;
IR0     call    DELAY           ; bit 0
        bcf     IRFCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        call    DELAY           ;
        bsf     IRFCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        btfss   IRFDAT          ;
        goto    IRLOW0          ;
        goto    IRHIGH0         ;
IRLOW0  bcf     IRDATA,0x00     ;
        goto    IR_END          ;
IRHIGH0 bsf     IRDATA,0x00     ;
        goto    IR_END          ;
IR_END  bcf     IRTEST,0x00     ; reset flag to indicate a measurement has not been started
        bsf     IR,0x07         ; valid data
        goto    INT_END         ;
;-------------------------------------------------------------------------------------
---------
REARIR  bcf     IR,0x07         ; for main loop to wait for valid data
        btfsc   IRTEST,0x01     ; check if measurement has been initiated
        goto    TESTOKR         ; if yes go check if measurement is finished else proceed
                                  with init
        decfsz  TM2CNT,0x01     ; wait for 70 ms before reading I.R.
        goto    INT_END         ;
```

```
        movlw   0x06            ; reset 70 ms counter
        movwf   TM2CNT          ;
        bcf     IRRCLK          ; make clock signal low to initiate a measurement
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        btfsc   IRRDAT          ; check for data signal to go low (ack that a measurement
                                  is in progress)
        goto    $ - 1           ;
        bsf     IRTEST,0x01     ; set flag to indicate a measurement has been started
        goto    INT_END         ;
TESTOKR btfss   IRRDAT          ; check for measurement complete
        goto    INT_END         ; else end interrupt so other routines may continue
                                  processing
        bsf     IRRCLK          ; start bit
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        call    DELAY           ;
        bcf     IRRCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        call    DELAY           ;
IRR7    bsf     IRRCLK          ; bit 7
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        btfss   IRRDAT          ;
        goto    LOW7            ;
        goto    HIGH7           ;
LOW7    bcf     IRRDATA,0x07    ;
        goto    IRR6            ;
HIGH7   bsf     IRRDATA,0x07    ;
        goto    IRR6            ;
IRR6    call    DELAY           ; bit 6
        bcf     IRRCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        call    DELAY           ;
        bsf     IRRCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        btfss   IRRDAT          ;
        goto    LOW6            ;
        goto    HIGH6           ;
LOW6    bcf     IRRDATA,0x06    ;
        goto    IRR5            ;
HIGH6   bsf     IRRDATA,0x06    ;
        goto    IRR5            ;
IRR5    call    DELAY           ; bit 5
        bcf     IRRCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        call    DELAY           ;
        bsf     IRRCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        btfss   IRRDAT          ;
        goto    LOW5            ;
        goto    HIGH5           ;
LOW5    bcf     IRRDATA,0x05    ;
        goto    IRR4            ;
HIGH5   bsf     IRRDATA,0x05    ;
        goto    IRR4            ;
IRR4    call    DELAY           ; bit 4
        bcf     IRRCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        call    DELAY           ;
        bsf     IRRCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        btfss   IRRDAT          ;
        goto    LOW4            ;
        goto    HIGH4           ;
LOW4    bcf     IRRDATA,0x04    ;
        goto    IRR3            ;
HIGH4   bsf     IRRDATA,0x04    ;
        goto    IRR3            ;
IRR3    call    DELAY           ; bit 3
        bcf     IRRCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        call    DELAY           ;
        bsf     IRRCLK          ;
```

```
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        btfss   IRRDAT          ;
        goto    LOW3            ;
        goto    HIGH3           ;
LOW3    bcf     IRRDATA,0x03    ;
        goto    IRR2            ;
HIGH3   bsf     IRRDATA,0x03    ;
        goto    IRR2            ;
IRR2    call    DELAY           ; bit 2
        bcf     IRRCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        call    DELAY           ;
        bsf     IRRCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        btfss   IRRDAT          ;
        goto    LOW2            ;
        goto    HIGH2           ;
LOW2    bcf     IRRDATA,0x02    ;
        goto    IRR1            ;
HIGH2   bsf     IRRDATA,0x02    ;
        goto    IRR1            ;
IRR1    call    DELAY           ; bit 1
        bcf     IRRCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        call    DELAY           ;
        bsf     IRRCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        btfss   IRRDAT          ;
        goto    LOW1            ;
        goto    HIGH1           ;
LOW1    bcf     IRRDATA,0x01    ;
        goto    IRR0            ;
HIGH1   bsf     IRRDATA,0x01    ;
        goto    IRR0            ;
IRR0    call    DELAY           ; bit 0
        bcf     IRRCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        call    DELAY           ;
        bsf     IRRCLK          ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        btfss   IRRDAT          ;
        goto    LOW0            ;
        goto    HIGH0           ;
LOW0    bcf     IRRDATA,0x00    ;
        goto    IRR_END         ;
HIGH0   bsf     IRRDATA,0x00    ;
        goto    IRR_END         ;
IRR_END bcf     IRTEST,0x01     ; reset flag to indicate a measurement has not been started
        bsf     IR,0x07         ; for main loop to wait for valid data
        goto    INT_END         ;
;-------------------------------------------------------------------------------------
---------
INT_END movf    W_TEMP,0x00     ; restore w reg contents
        retfie                  ;
;*************************************************************************
;
;       SUBROUTINES
;
;*************************************************************************
;-------------------------------------------------------------------------
;       READ 1 bit of data from mouse
;       Data is sent LSB first.
;-------------------------------------------------------------------------
MS_BIT  btfss   MSCLOCK         ; make sure clock is high
        goto    $ - 1           ;
        btfsc   MSCLOCK         ; wait for falling edge
        goto    $ - 1           ;
        btfss   MSDATA          ; check if data is '1' or '0'
        goto    BIT_0           ;
        bsf     STATUS,0x00     ;
        goto    BIT_Rx          ;
```

```
BIT_0   bcf     STATUS,0x00     ;
BIT_Rx  rrf     MSINFO,0x01     ; rotate new bit into temp reg
        return                  ;
;-------------------------------------------------------------------------------
;       INFORM MOUSE OF INCOMING Tx and Then Initialize the mouse
;       Holds mouse clock line low for at least 100 uSec
;       At an .0000002 sec per instruction that takes 500 instruction cycles.
;-------------------------------------------------------------------------------
MS_TX   movlw   0xA6            ; 166
        movwf   COUNT1          ;
        bsf     MSCCTRL         ; pull mouse clock line low for >= 100 uSec
        decfsz  COUNT1,0x01     ; worth one instruction cycle until count1 = 0
        goto    $ - 1           ; worth two instruction cycles
                                ; thus 166 * 3 = 498 Instruction cycles
                                ; plus 1 for the decfsz on the last count
                                ; plus the bsf below = 500 I.C.'s
MS_TXR  bsf     MSDCTRL         ; pull mouse data line low
        bcf     MSCCTRL         ; release the mouse clock line
        btfsc   MSCLOCK         ; wait for mouse to pull clock low
        goto    $ - 1           ;
        bsf     MSDCTRL         ; Start Bit = 0
        btfss   MSCLOCK         ; wait for mouse to pull clock high
        goto    $ - 1           ;
        btfsc   MSCLOCK         ; wait for mouse to pull clock low
        goto    $ - 1           ;
        bsf     MSDCTRL         ; Bit 0 = 0
        btfss   MSCLOCK         ; wait for mouse to pull clock high
        goto    $ - 1           ;
        btfsc   MSCLOCK         ; wait for mouse to pull clock low
        goto    $ - 1           ;
        bsf     MSDCTRL         ; Bit 1 = 0
        btfss   MSCLOCK         ; wait for mouse to pull clock high
        goto    $ - 1           ;
        btfsc   MSCLOCK         ; wait for mouse to pull clock low
        goto    $ - 1           ;
        bcf     MSDCTRL         ; Bit 2 = 1
        btfss   MSCLOCK         ; wait for mouse to pull clock high
        goto    $ - 1           ;
        btfsc   MSCLOCK         ; wait for mouse to pull clock low
        goto    $ - 1           ;
        bsf     MSDCTRL         ; Bit 3 = 0
        btfss   MSCLOCK         ; wait for mouse to pull clock high
        goto    $ - 1           ;
        btfsc   MSCLOCK         ; wait for mouse to pull clock low
        goto    $ - 1           ;
        bcf     MSDCTRL         ; Bit 4 = 1
        btfss   MSCLOCK         ; wait for mouse to pull clock high
        goto    $ - 1           ;
        btfsc   MSCLOCK         ; wait for mouse to pull clock low
        goto    $ - 1           ;
        bcf     MSDCTRL         ; Bit 5 = 1
        btfss   MSCLOCK         ; wait for mouse to pull clock high
        goto    $ - 1           ;
        btfsc   MSCLOCK         ; wait for mouse to pull clock low
        goto    $ - 1           ;
        bcf     MSDCTRL         ; Bit 6 = 1
        btfss   MSCLOCK         ; wait for mouse to pull clock high
        goto    $ - 1           ;
        btfsc   MSCLOCK         ; wait for mouse to pull clock low
        goto    $ - 1           ;
        bcf     MSDCTRL         ; Bit 7 = 1
        btfss   MSCLOCK         ; wait for mouse to pull clock high
        goto    $ - 1           ;
        btfsc   MSCLOCK         ; wait for mouse to pull clock low
        goto    $ - 1           ;
        bsf     MSDCTRL         ; Parity Bit = 0
        btfss   MSCLOCK         ; wait for mouse to pull clock high
        goto    $ - 1           ;
        btfsc   MSCLOCK         ; wait for mouse to pull clock low
        goto    $ - 1           ;
        bcf     MSDCTRL         ; Stop Bit = 1
```

```
        btfss   MSCLOCK         ; wait for mouse to pull clock high
        goto    $ - 1           ;
        btfsc   MSDATA          ; wait for mouse to pull data low
        goto    $ - 1           ;
        btfss   MSDATA          ; wait for mouse to release data
        goto    $ - 1           ;
        bcf     MSCCTRL         ; enable mouse tx
        bcf     MSDCTRL         ;
        movlw   0x08            ; Initialize counter for 8 bits
        movwf   COUNT2          ;
        btfsc   MSCLOCK         ; wait for mouse to bring clock low
        goto    $ - 1           ;
        btfsc   MSDATA          ; wait for mouse to bring data low
        goto    $ - 1           ;
READB2  call    MS_BIT          ; get one bit
        decfsz  COUNT2,0x01     ; decrement counter
        goto    READB2          ; get next bit
        btfss   MSCLOCK         ; make sure clock is high
        goto    $ - 1           ;                               PARITY
        btfsc   MSCLOCK         ; wait for falling edge
        goto    $ - 1           ;
        btfss   MSCLOCK         ; make sure clock is high
        goto    $ - 1           ;                               STOP
        btfsc   MSCLOCK         ; wait for falling edge
        goto    $ - 1           ;
        btfss   MSCLOCK         ; wait for clock to float high again
        goto    $ - 1           ;
        bsf     MSCCTRL         ; pull mouse clock line low (disable transmission)
        return                  ;
;-------------------------------------------------------------------------------
;       READ ONE PACKET OF DATA FROM MOUSE
;-------------------------------------------------------------------------------
MS_INFO bcf     MSCCTRL         ; enable mouse tx
        bcf     MSDCTRL         ;
        movlw   0xFF            ; This rountine may try to find the beginning of a mouse
                                  data packet 255 times
        movwf   INCNT           ; If 255 tries all fail the routine times out and exits
                                  leaving the previous values intact
TIMEIN  decfsz  INCNT,0x01      ;
        goto    TOPL            ;
        goto    T_OUT           ; here is the timeout exit point
TOPL    movlw   0x7F            ; 127 * .2usec per instruction * about 6 instructions = 152
                                  usec period of clock inactivity
        movwf   OUTCNT          ; if the mouse clock is not inactive for this period of
                                  time the rountine has tried to read
OUTERL  decfsz  OUTCNT,0x01     ; mouse data in the middle of a packet (who knows what bit
                                  is on) so try again until a timeout occurs
        goto    KLP             ;
        goto    LOOK            ;
KLP     btfss   MSCLOCK         ; here is where the clock is checked as long as the clock
                                  remains in its inactive state (high)
        goto    TIMEIN          ; for the duration of the counter (OUTCNT) then the mouse
                                  is between packets and the next clock
        goto    OUTERL          ; edge (falling) will be the start bit of the first byte of
                                  data in a new packet.
LOOK    movlw   0x08            ; Initialize counter for 8 bits
        movwf   COUNT2          ;
        movlw   0xFF            ;
        movwf   TOUT_3          ;
CIL3    decfsz  TOUT_3,0x01     ;
        goto    CIL4            ;
        goto    T_OUT           ;
CIL4    movlw   0xFF            ;
        movwf   TOUT_2          ;
CHECCK0 btfsc   MSCLOCK         ; wait for mouse to bring clock low
        goto    CIL1            ;
        goto    CIL2            ;
CIL1    decfsz  TOUT_2,0x01     ;
        goto    CHECCK0         ;
        goto    CIL3            ;
CIL2    btfsc   MSDATA          ; wait for mouse to bring data low
```

```
        goto    $ - 1           ;
READB   call    MS_BIT          ; get one bit
        decfsz  COUNT2,0x01     ; decrement counter
        goto    READB           ; get next bit
        btfss   MSCLOCK         ; make sure clock is high
        goto    $ - 1           ;                               PARITY
        btfsc   MSCLOCK         ; wait for falling edge
        goto    $ - 1           ;
        btfss   MSCLOCK         ; make sure clock is high
        goto    $ - 1           ;                               STOP
        btfsc   MSCLOCK         ; wait for falling edge
        goto    $ - 1           ;
        btfss   MSCLOCK         ; wait for clock to float high again
        goto    $ - 1           ;
        btfss   MSINFO,0x02     ;
        goto    OK_01           ;
        goto    RD_END          ;
OK_01   btfsc   MSINFO,0x03     ;
        goto    OK_02           ;
        goto    RD_END          ;
OK_02   movf    MSINFO,0x00     ;
INFOOK  movwf   BUTTONS         ;
        movlw   0x08            ; Initialize counter for 8 bits
        movwf   COUNT2          ;
CHECCK1 btfsc   MSCLOCK         ; wait for mouse to bring data low
        goto    $ - 1           ; (get start bit)
        btfsc   MSDATA          ; wait for mouse to bring clock low
        goto    $ - 1           ;
READX   call    MS_BIT          ; get one bit
        decfsz  COUNT2,0x01     ; decrement counter
        goto    READX           ; get next bit
        btfss   MSCLOCK         ; make sure clock is high
        goto    $ - 1           ;                               PARITY
        btfsc   MSCLOCK         ; wait for falling edge
        goto    $ - 1           ;
        btfss   MSCLOCK         ; make sure clock is high
        goto    $ - 1           ;                               STOP
        btfsc   MSCLOCK         ; wait for falling edge
        goto    $ - 1           ;
        btfss   MSCLOCK         ; wait for clock to float high again
        goto    $ - 1           ;
        movf    MSINFO,0x00     ;
        movwf   X_MAG           ;
        movlw   0x08            ; Initialize counter for 8 bits
        movwf   COUNT2          ;
CHECCK2 btfsc   MSCLOCK         ; wait for mouse to bring data low
        goto    $ - 1           ; (get start bit)
        btfsc   MSDATA          ; wait for mouse to bring clock low
        goto    $ - 1           ;
READM   call    MS_BIT          ; get one bit
        decfsz  COUNT2,0x01     ; decrement counter
        goto    READM           ; get next bit
        btfss   MSCLOCK         ; make sure clock is high
        goto    $ - 1           ;                               PARITY
        btfsc   MSCLOCK         ; wait for falling edge
        goto    $ - 1           ;
        btfss   MSCLOCK         ; make sure clock is high
        goto    $ - 1           ;                               STOP
        btfsc   MSCLOCK         ; wait for falling edge
        goto    $ - 1           ;
        movf    MSINFO,0x00     ;
        movwf   Y_MAG           ;
        goto    RD_END          ;
T_OUT   clrf    BUTTONS         ;
RD_END  bsf     MSCCTRL         ;
        return                  ;
;-------------------------------------------------------------------------------
X_CURS  bcf     E_LQD           ;  set cursor address
        bcf     RW_LQD          ;
        bcf     RS_LQD          ;
        movlw   0x85            ;
```

```
        movwf   LQD             ;
        bsf     E_LQD           ;
        nop                     ;
        nop                     ;
        bcf     E_LQD           ;
        call    CHK_BSY         ;
        nop                     ;
        nop                     ;
        return                  ;
;-------------------------------------------------------------------------------
Y_CURS  bcf     E_LQD           ;  set cursor address
        bcf     RW_LQD          ;
        bcf     RS_LQD          ;
        movlw   0xC5            ;
        movwf   LQD             ;
        bsf     E_LQD           ;
        nop                     ;
        nop                     ;
        bcf     E_LQD           ;
        call    CHK_BSY         ;
        nop                     ;
        nop                     ;
        return                  ;
;-------------------------------------------------------------------------------
;       Send a character to the LQD
;       Character to be sent should be in register "LQDDATA"
;-------------------------------------------------------------------------------
LQD_SNDbcf      E_LQD           ;
        bcf     RW_LQD          ;
        bsf     RS_LQD          ;
        movf    LQDDATA,0x00    ;
        movwf   LQD             ; send char info to data bus
        bsf     E_LQD           ; enable instruction
        nop                     ;
        nop                     ;
        bcf     E_LQD           ;
        call    CHK_BSY         ;
        return                  ;
;-------------------------------------------------------------------------------
;       LQDINIT .........DUH!
;-------------------------------------------------------------------------------
LQDINITcall     CHK_BSY         ;
        bcf     E_LQD           ;
        bcf     RW_LQD          ;
        bcf     RS_LQD          ;
        movlw   0x38            ;
        movwf   LQD             ;
        bsf     E_LQD           ;
        nop                     ;
        bcf     E_LQD           ;
        call    CHK_BSY         ;
        nop                     ;
        nop                     ;
        bcf     E_LQD           ;
        bcf     RW_LQD          ;
        bcf     RS_LQD          ;
        movlw   0x0E            ;
        movwf   LQD             ;
        bsf     E_LQD           ;
        nop                     ;
        bcf     E_LQD           ;
        call    CHK_BSY         ;
        nop                     ;
        nop                     ;
        bcf     E_LQD           ;
        bcf     RW_LQD          ;
        bcf     RS_LQD          ;
        movlw   0x06            ;
        movwf   LQD             ;
        bsf     E_LQD           ;
        nop                     ;
```

```
        bcf     E_LQD           ;
        call    CHK_BSY         ;
        nop                     ;
        nop                     ;
        bcf     E_LQD           ;
        bcf     RW_LQD          ;
        bcf     RS_LQD          ;
        movlw   0x01            ;
        movwf   LQD             ;
        bsf     E_LQD           ;
        nop                     ;
        bcf     E_LQD           ;
        call    CHK_BSY         ;
        nop                     ;
        nop                     ;
        movlw   0x58            ;
        movwf   LQDDATA         ;
        call    LQD_SND         ;
        movlw   0x20            ;
        movwf   LQDDATA         ;
        call    LQD_SND         ;
        movlw   0x3D            ;
        movwf   LQDDATA         ;
        call    LQD_SND         ;
        movlw   0x20            ;
        movwf   LQDDATA         ;
        call    LQD_SND         ;
        bcf     E_LQD           ;  set cursor address
        bcf     RW_LQD          ;
        bcf     RS_LQD          ;
        movlw   0xC0            ;
        movwf   LQD             ;
        bsf     E_LQD           ;
        nop                     ;
        nop                     ;
        bcf     E_LQD           ;
        call    CHK_BSY         ;
        nop                     ;
        nop                     ;
        movlw   0x59            ;
        movwf   LQDDATA         ;
        call    LQD_SND         ;
        movlw   0x20            ;
        movwf   LQDDATA         ;
        call    LQD_SND         ;
        movlw   0x3D            ;
        movwf   LQDDATA         ;
        call    LQD_SND         ;
        movlw   0x20            ;
        movwf   LQDDATA         ;
        call    LQD_SND         ;
        return                  ;
;-------------------------------------------------------------------------------
CHK_BSY bsf     RW_LQD          ; check for busy flag = 0
        bcf     RS_LQD          ;
        bsf     STATUS,RP0      ;       select bank 1
        bcf     STATUS,RP1      ;       select bank 1
        movlw   0xFF            ;
        movwf   TRISD           ;
        bcf     STATUS,RP0      ;       select bank 0
        bcf     STATUS,RP1      ;       select bank 0
KEEPCHK bcf     E_LQD           ;
        nop                     ;
        bsf     E_LQD           ;
        nop                     ;
        btfsc   BUSYFL          ;
        goto    KEEPCHK         ;
        bcf     E_LQD           ;
        bsf     STATUS,RP0      ;       select bank 1
        bcf     STATUS,RP1      ;       select bank 1
        movlw   0x00            ;
```

```
        movwf   TRISD           ;
        bcf     STATUS,RP0      ;       select bank 0
        bcf     STATUS,RP1      ;       select bank 0
        return                  ;
;-------------------------------------------------------------------------------
REV_ASC movf    STATUS,0x00     ;
        movwf   S_TEMP          ;
        bcf     STATUS,RP0      ;
        bcf     STATUS,RP1      ;
        movf    REVTEMP,0x00    ;
        movwf   REVTMP2         ;
        bcf     REVTMP2,0x00    ;
        bcf     REVTMP2,0x01    ;
        bcf     REVTMP2,0x02    ;
        bcf     REVTMP2,0x03    ;
        swapf   REVTMP2,0x01    ;
        movf    REVTMP2,0x00    ;
        sublw   0x09            ;
        btfsc   STATUS,0x00     ;
        goto    NMBR            ;
        goto    LTTR            ;
NMBR    movlw   0x30            ;
        addwf   REVTMP2,0x00    ;
        movwf   REVTMP3         ;
        goto    REVNEXT         ;
LTTR    movlw   0x37            ;
        addwf   REVTMP2,0x00    ;
        movwf   REVTMP3         ;
        goto    REVNEXT         ;
REVNEXT movf    REVTEMP,0x00    ;
        movwf   REVTMP2         ;
        bcf     REVTMP2,0x04    ;
        bcf     REVTMP2,0x05    ;
        bcf     REVTMP2,0x06    ;
        bcf     REVTMP2,0x07    ;
        movf    REVTMP2,0x00    ;
        sublw   0x09            ;
        btfsc   STATUS,0x00     ;
        goto    NMBR2           ;
        goto    LTTR2           ;
NMBR2   movlw   0x30            ;
        addwf   REVTMP2,0x00    ;
        movwf   REVTMP4         ;
        goto    REVDONE         ;
LTTR2   movlw   0x37            ;
        addwf   REVTMP2,0x00    ;
        movwf   REVTMP4         ;
        goto    REVDONE         ;
REVDONE movf    S_TEMP,0x00     ;
        movwf   STATUS          ;
        return                  ;
;-------------------------------------------------------------------------------
DELAY   movlw   0xFA            ;
        movwf   COUNT           ; 250 cycles
        decfsz  COUNT,0x01      ;
        goto    $ - 1           ;
        return                  ;
;-------------------------------------------------------------------------------
DELAY2  movf    SECONDS,0x00    ;
        movwf   NDELAY          ;
        bsf     MOTL_OE         ;
        bsf     MOTR_OE         ;
D2NL    decfsz  NDELAY,0x01     ; Delay causes a SECONDS second long delay
        goto    D2OL            ;
        goto    D2_EXIT         ;
D2OL    movlw   0xFF            ;
        movwf   ODELAY          ;
D2OL2   decfsz  ODELAY,0x01     ;
        goto    D2IL            ;
        goto    D2NL            ;
D2IL    movlw   0x9D            ;
```

```
        movwf   IDELAY          ;
D2IL2   decfsz  IDELAY,0x01     ;
        goto    D2NOP           ;
        goto    D2OL2           ;
D2NOP   goto    D2IL2           ;
D2_EXITreturn                   ;
;-------------------------------------------------------------------------------
; 90 degree turn routine (to the right)
;-------------------------------------------------------------------------------
NDT_R   movlw   0x92            ;
        movwf   RPULSES         ;
        bsf     MOTL_DR         ;
        bsf     MOTR_DR         ;
        bcf     MOTL_OE         ;
        bcf     MOTR_OE         ;
        bsf     NAV_OK,0x00     ;
        bsf     INTCON,GIE      ; enable unmasked interrupts
CK_P    movf    RPULSES,0x00    ;
        btfss   STATUS,0x02     ;
        goto    CK_P            ;
        goto    DWRT            ;
DWRT    bcf     INTCON,GIE      ; enable unmasked interrupts
        bsf     MOTL_OE         ;
        bsf     MOTR_OE         ;
        bcf     NAV_OK,0x00     ;
        return                  ;
;-------------------------------------------------------------------------------
; 45 degree turn routine (to the right)
;-------------------------------------------------------------------------------
FFDT_R  movlw   0x4A            ;
        movwf   RPULSES         ;
        bsf     MOTL_DR         ;
        bsf     MOTR_DR         ;
        bcf     MOTL_OE         ;
        bcf     MOTR_OE         ;
        bsf     NAV_OK,0x00     ;
        bsf     INTCON,GIE      ; enable unmasked interrupts
CK_PZ   movf    RPULSES,0x00    ;
        btfss   STATUS,0x02     ;
        goto    CK_PZ           ;
        goto    DWFFRT          ;
DWFFRT  bcf     INTCON,GIE      ; enable unmasked interrupts
        bsf     MOTL_OE         ;
        bsf     MOTR_OE         ;
        bcf     NAV_OK,0x00     ;
        return                  ;
;-------------------------------------------------------------------------------
; 90 degree turn routine (to the left)
;-------------------------------------------------------------------------------
NDT_L   movlw   0x92            ;
        movwf   RPULSES         ;
        bcf     MOTL_DR         ;
        bcf     MOTR_DR         ;
        bcf     MOTL_OE         ;
        bcf     MOTR_OE         ;
        bsf     NAV_OK,0x00     ;
        bsf     INTCON,GIE      ; enable unmasked interrupts
CK_P2   movf    RPULSES,0x00    ;
        btfss   STATUS,0x02     ;
        goto    CK_P2           ;
        goto    DWLT            ;
DWLT    bcf     INTCON,GIE      ; enable unmasked interrupts
        bsf     MOTL_OE         ;
        bsf     MOTR_OE         ;
        bcf     NAV_OK,0x00     ;
        return                  ;
;-------------------------------------------------------------------------------
; 45 degree turn routine (to the left)
;-------------------------------------------------------------------------------
FFDT_L  movlw   0x4A            ;
        movwf   RPULSES         ;
```

```
        bcf     MOTL_DR         ;
        bcf     MOTR_DR         ;
        bcf     MOTL_OE         ;
        bcf     MOTR_OE         ;
        bsf     NAV_OK,0x00     ;
        bsf     INTCON,GIE      ; enable unmasked interrupts
CK_P2Z  movf    RPULSES,0x00    ;
        btfss   STATUS,0x02     ;
        goto    CK_P2Z          ;
        goto    DWFFLT          ;
DWFFLT  bcf     INTCON,GIE      ; enable unmasked interrupts
        bsf     MOTL_OE         ;
        bsf     MOTR_OE         ;
        bcf     NAV_OK,0x00     ;
        return                  ;
;-------------------------------------------------------------------------------
; this rountine takes in the value of 1/10ths inches (actually .047 inches per pulse)
; to move in the register DIST
;-------------------------------------------------------------------------------
M_F     movf    DIST,0x00       ;
        movwf   RPULSES         ;
        bcf     MOTL_DR         ;
        bsf     MOTR_DR         ;
        bcf     MOTL_OE         ;
        bcf     MOTR_OE         ;
        bsf     NAV_OK,0x00     ;
        bsf     INTCON,GIE      ; enable unmasked interrupts
CK_P3   movf    RPULSES,0x00    ;
        btfss   STATUS,0x02     ;
        goto    CK_P3           ;
        goto    DM_F            ;
DM_F    bcf     INTCON,GIE      ; enable unmasked interrupts
        bsf     MOTL_OE         ;
        bsf     MOTR_OE         ;
        bcf     NAV_OK,0x00     ;
        return                  ;
;-------------------------------------------------------------------------------
; this rountine takes in the value of 1/10ths inches (actually .047 inches per pulse)
; to move in the register DIST
;-------------------------------------------------------------------------------
M_B     movf    DIST,0x00       ;
        movwf   RPULSES         ;
        bsf     MOTL_DR         ;
        bcf     MOTR_DR         ;
        bcf     MOTL_OE         ;
        bcf     MOTR_OE         ;
        bsf     NAV_OK,0x00     ;
        bsf     INTCON,GIE      ; enable unmasked interrupts
BACKW   movf    RPULSES,0x00    ;
        btfss   STATUS,0x02     ;
        goto    BACKW           ;
        goto    DM_F12          ;
DM_F12  bcf     INTCON,GIE      ; enable unmasked interrupts
        bsf     MOTL_OE         ;
        bsf     MOTR_OE         ;
        bcf     NAV_OK,0x00     ;
        return                  ;
;-------------------------------------------------------------------------------
;ACC rountine accelerates the robot to the desired wheel speed to avoid stalls
;-------------------------------------------------------------------------------
ACC     btfss   NAV_OK,0x01     ;
        goto    R_WH            ;
        goto    L_W             ;
R_WH    movf    RWT,0x01        ; check if desired time is zero
        btfss   STATUS,0x02     ;
        goto    RWNOT_Z         ;
        goto    SLOW_RW         ;
SLOW_RW clrf    RWT_TMP         ;
        return                  ;
RWNOT_Z decfsz  ACCRCNT,0x01    ;
        return                  ;
```

```
        movf    MAX_ACC,0x00    ;
        movwf   ACCRCNT         ;
        movf    RWT,0x00        ; get user desired wheel time
        subwf   RWT_TMP,0x00    ; compare current value with dersired value
        btfss   STATUS,0x00     ;
        goto    ACC_RP          ; negative acceleration needed
        goto    ACC_RN          ; positive acceleration needed
ACC_RN  movf    RWT,0x00        ; get user desired wheel time
        subwf   RWT_TMP,0x00    ; compare current value with dersired value
        btfss   STATUS,0x02     ;
        goto    KEEPRAN         ;
        goto    DONERAN         ;
KEEPRAN decf    RWT_TMP,0x01    ; decrement wheel time by one
DONERAN return                  ;
ACC_RP  movf    RWT,0x00        ; compare current value with dersired value
        subwf   RWT_TMP,0x00    ;
        btfss   STATUS,0x02     ;
        goto    KEEPRAP         ;
        goto    DONERAP         ;
KEEPRAP incf    RWT_TMP,0x01    ; increment wheel time by one
DONERAP return                  ;
L_W     movf    LWTH,0x01       ; check if desired time is zero
        btfss   STATUS,0x02     ;
        goto    LWNOT_Z         ;
        goto    SLOW_LW         ;
SLOW_LW clrf    LWTHTMP         ;
        return                  ;
LWNOT_Z decfsz  ACCLCNT,0x01    ;
        return                  ;
        movf    MAX_ACC,0x00    ;
        movwf   ACCLCNT         ;
        movf    LWTH,0x00       ; get user desired wheel time
        subwf   LWTHTMP,0x00    ; compare current value with dersired value
        btfss   STATUS,0x00     ;
        goto    ACC_LP          ; negative acceleration needed
        goto    ACC_LN          ; positive acceleration needed
ACC_LN  movf    LWTH,0x00       ; get user desired wheel time
        subwf   LWTHTMP,0x00    ; compare current value with dersired value
        btfss   STATUS,0x02     ;
        goto    KEEPLAN         ;
        goto    DONELAN         ;
KEEPLAN decf    LWTHTMP,0x01    ; decrement wheel time by one
DONELAN return                  ;
ACC_LP  movf    LWTH,0x00       ; compare current value with dersired value
        subwf   LWTHTMP,0x00    ;
        btfss   STATUS,0x02     ;
        goto    KEEPLAP         ;
        goto    DONELAP         ;
KEEPLAP incf    LWTHTMP,0x01    ; increment wheel time by one
DONELAP return                  ;
;-------------------------------------------------------------------------------
;SQUARE rountine will drive the robot in a square shape with dimensions DIST x DIST
;-------------------------------------------------------------------------------
SQUARE  movlw   0x64            ; 100 decimal
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        call    NDT_R           ;
        call    DELAY2          ;
        movlw   0x64            ; 100 decimal
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        call    NDT_R           ;
        call    DELAY2          ;
        movlw   0x64            ; 100 decimal
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        call    NDT_R           ;
        call    DELAY2          ;
```

```
        movlw   0x64            ; 100 decimal
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        call    NDT_R           ;
        call    DELAY2          ;
        return                  ;
;-------------------------------------------------------------------------------
;CIRCLE will drive robot in a circle
;-------------------------------------------------------------------------------
CIRCLE  movlw   0x80            ;
        movwf   LWTH            ;
        movlw   0x00            ;
        movwf   RWT             ;
        bcf     MOTL_DR         ; move forward
        bsf     MOTR_DR         ;
        bcf     MOTL_OE         ;
        bcf     MOTR_OE         ;
        call    DELAY2          ;
        return                  ;
;-------------------------------------------------------------------------------
LETTERG bcf     MARKER          ;
        call    DELAY2          ;
        movlw   0x7F            ;       * draw left portion of G
        movwf   DIST            ;       *
        call    M_F             ;       *
        call    DELAY2          ;       *
        bsf     MARKER          ; lift up pen
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_R           ; turn 90 degrees to the right
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        bcf     MARKER          ; put down pen
        call    DELAY2          ;
        movlw   0x40            ; * * * * Top part of G
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        bsf     MARKER          ; lift up pen
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_R           ; turn 90 degrees to the right
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        movlw   0x40            ;
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_R           ; turn 90 degrees to the right
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
```

```
        bcf     MARKER          ; put down pen
        call    DELAY2          ;
        movlw   0x20            ;
        movwf   DIST            ;
        call    M_F             ;       *** PART of G
        call    DELAY2          ;
        bsf     MARKER          ; pick up pen
        call    DELAY2          ;
        movlw   0x20            ;
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_L           ; turn 90 degrees to the left
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        bcf     MARKER          ; put down pen
        call    DELAY2          ;
        movlw   0x40            ;
        movwf   DIST            ;       *
        call    M_F             ;       * PART of G
        call    DELAY2          ;       *
        bsf     MARKER          ; pick up pen
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_R           ; turn 90 degrees to the right
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        bcf     MARKER          ; put down pen
        call    DELAY2          ;
        movlw   0x40            ;
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ; *******  PART of G
        bsf     MARKER          ; pick up pen
        call    DELAY2          ;
        movlw   0x40            ; get ready for next letter
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        movlw   0x20            ; space between letters
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_R           ; turn 90 degrees to the right
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        return                  ;
;-------------------------------------------------------------------------------
LETTERA bcf     MARKER          ;
        call    DELAY2          ;
        movlw   0x7F            ;       * draw left portion of A
```

```
movwf   DIST            ;       *
call    M_F             ;       *
call    DELAY2          ;       *
bsf     MARKER          ; lift up pen
call    DELAY2          ;
movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
movwf   DIST            ;
call    M_B             ;
call    DELAY2          ;
call    NDT_R           ; turn 90 degrees to the right
call    DELAY2          ;
movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
movwf   DIST            ;
call    M_F             ;
call    DELAY2          ;
bcf     MARKER          ; put down pen
call    DELAY2          ;
movlw   0x40            ; * * * * Top part of A
movwf   DIST            ;
call    M_F             ;
call    DELAY2          ;
bsf     MARKER          ; lift up pen
call    DELAY2          ;
movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
movwf   DIST            ;
call    M_B             ;
call    DELAY2          ;
call    NDT_R           ; turn 90 degrees to the right
call    DELAY2          ;
movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
movwf   DIST            ;
call    M_F             ;
call    DELAY2          ;
bcf     MARKER          ;
call    DELAY2          ;
movlw   0x7F            ;       * draw right portion of A
movwf   DIST            ;       *
call    M_F             ;       *
call    DELAY2          ;       *
bsf     MARKER          ; lift up pen
call    DELAY2          ;
movlw   0x40            ;
movwf   DIST            ;
call    M_B             ;
call    DELAY2          ;
movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
movwf   DIST            ;
call    M_B             ;
call    DELAY2          ;
call    NDT_R           ; turn 90 degrees to the right
call    DELAY2          ;
movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
movwf   DIST            ;
call    M_F             ;
call    DELAY2          ;
bcf     MARKER          ; put down pen
call    DELAY2          ;
movlw   0x40            ; Middle part of A ****
movwf   DIST            ;
call    M_F             ;
call    DELAY2          ;
bsf     MARKER          ; lift up pen
call    DELAY2          ;
movlw   0x40            ;
movwf   DIST            ;
call    M_B             ;
call    DELAY2          ;
movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
movwf   DIST            ;
call    M_B             ;
call    DELAY2          ;
```

```
        call    NDT_L           ; turn 90 degrees to the left
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        movlw   0x40            ;  get ready for next letter
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_L           ; turn 90 degrees to the left
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        movlw   0x20            ; space between letters
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_L           ; turn 90 degrees to the left
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        return                  ;
;-------------------------------------------------------------------------------
LETTERT bsf     MARKER          ;
        call    DELAY2          ;
        movlw   0x7F            ;
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_R           ; turn 90 degrees to the right
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        bcf     MARKER          ; put down pen
        call    DELAY2          ;
        movlw   0x40            ;      DRAW TOP of T *********
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        bsf     MARKER          ; pick up pen
        call    DELAY2          ;
        movlw   0x20            ;
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_R           ; turn 90 degrees to the right
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
```

```
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        bcf     MARKER          ; put down pen
        call    DELAY2          ;
        movlw   0x7F            ; DRAW VERTICAL PART of T     *
        movwf   DIST            ;                             *
        call    M_F             ;                             *
        call    DELAY2          ;                             *
        bsf     MARKER          ; pick up pen
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_L           ; turn 90 degrees to the left
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        movlw   0x40            ;
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        movlw   0x20            ;
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_L           ; turn 90 degrees to the left
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        return                  ;
;-------------------------------------------------------------------------------
LETTERObcf      MARKER          ;
        call    DELAY2          ;
        movlw   0x7F            ;       DRAW LEFT VERT OF O
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        bsf     MARKER          ; pick up pen
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_R           ; turn 90 degrees to the right
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        bcf     MARKER          ; put down pen
        call    DELAY2          ;
        movlw   0x40            ;       DRAW TOP OF O
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        bsf     MARKER          ; pick up pen
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
```

```
        call    NDT_R           ; turn 90 degrees to the right
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        bcf     MARKER          ; put down pen
        call    DELAY2          ;
        movlw   0x7F            ;       DRAW RIGHT VERT OF O
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        bsf     MARKER          ; pick up pen
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_R           ; turn 90 degrees to the right
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        bcf     MARKER          ; put down pen
        call    DELAY2          ;
        movlw   0x40            ;       DRAW BOTTOM OF O
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        bsf     MARKER          ; pick up pen
        call    DELAY2          ;
        movlw   0x40            ;
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        movlw   0x20            ; space between letters
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_R           ; turn 90 degrees to the right
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        return                  ;
;-------------------------------------------------------------------------------
LETTERR bcf     MARKER          ;
        call    DELAY2          ;
        movlw   0x7F            ;       DRAW LEFT VERT OF R
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        bsf     MARKER          ; pick up pen
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_R           ; turn 90 degrees to the right
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        bcf     MARKER          ; put down pen
```

```
call   DELAY2         ;
movlw  0x40           ;       DRAW TOP OF R
movwf  DIST           ;
call   M_F            ;
call   DELAY2         ;
bsf    MARKER         ; pick up pen
call   DELAY2         ;
movf   MALIGNB,0x00   ; Approx 2.75 inches to re-align the pen
movwf  DIST           ;
call   M_B            ;
call   DELAY2         ;
call   NDT_R          ; turn 90 degrees to the right
call   DELAY2         ;
movf   MALIGNF,0x00   ; Approx 2.75 inches to re-align the pen
movwf  DIST           ;
call   M_F            ;
call   DELAY2         ;
bcf    MARKER         ; put down pen
call   DELAY2         ;
movlw  0x40           ;       DRAW RIGHT VERT OF R
movwf  DIST           ;
call   M_F            ;
call   DELAY2         ;
bsf    MARKER         ; pick up pen
call   DELAY2         ;
movf   MALIGNB,0x00   ; Approx 2.75 inches to re-align the pen
movwf  DIST           ;
call   M_B            ;
call   DELAY2         ;
call   NDT_R          ; turn 90 degrees to the right
call   DELAY2         ;
movf   MALIGNF,0x00   ; Approx 2.75 inches to re-align the pen
movwf  DIST           ;
call   M_F            ;
call   DELAY2         ;
bcf    MARKER         ; put down pen
call   DELAY2         ;
movlw  0x40           ;       DRAW MIDDLE HORZ OF R
movwf  DIST           ;
call   M_F            ;
call   DELAY2         ;
bsf    MARKER         ; pick up pen
call   DELAY2         ;
movf   MALIGNB,0x00   ; Approx 2.75 inches to re-align the pen
movwf  DIST           ;
call   M_B            ;
call   DELAY2         ;
call   NDT_L          ; turn 90 degrees to the left
call   DELAY2         ;
movf   MALIGNF,0x00   ; Approx 2.75 inches to re-align the pen
movwf  DIST           ;
call   M_F            ;
call   DELAY2         ;
movf   MALIGNB,0x00   ; Approx 2.75 inches to re-align the pen
movwf  DIST           ;
call   M_B            ;
call   DELAY2         ;
call   FFDT_L         ; turn 45 degrees to the left
call   DELAY2         ;
movf   MALIGNF,0x00   ; Approx 2.75 inches to re-align the pen
movwf  DIST           ;
call   M_F            ;
call   DELAY2         ;
bcf    MARKER         ; put down pen
call   DELAY2         ;
movlw  0x5A           ;       DRAW DIAGONAL OF R
movwf  DIST           ;
call   M_F            ;
call   DELAY2         ;
bsf    MARKER         ; pick up pen
call   DELAY2         ;
```

```
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    FFDT_L          ; turn 45 degrees to the left
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        movlw   0x20            ; space between letters
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_L           ; turn 90 degrees to the left
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        return                  ;
;-------------------------------------------------------------------------------
LETTERS bsf     MARKER          ; pick up pen
        call    DELAY2          ;
        movlw   0x40            ;
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        bcf     MARKER          ; put down pen
        call    DELAY2          ;
        movlw   0x40            ; draw left vert of S
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        bsf     MARKER          ; pick up pen
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_R           ; turn 90 degrees to the right
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        bcf     MARKER          ; put down pen
        call    DELAY2          ;
        movlw   0x40            ; draw TOP of S
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        bsf     MARKER          ; pick up pen
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_R           ; turn 90 degrees to the right
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        movlw   0x40            ;
        movwf   DIST            ;
        call    M_F             ;
```

```
call   DELAY2         ;
movf   MALIGNB,0x00   ; Approx 2.75 inches to re-align the pen
movwf  DIST           ;
call   M_B            ;
call   DELAY2         ;
call   NDT_R          ; turn 90 degrees to the right
call   DELAY2         ;
movf   MALIGNF,0x00   ; Approx 2.75 inches to re-align the pen
movwf  DIST           ;
call   M_F            ;
call   DELAY2         ;
bcf    MARKER         ; put down pen
call   DELAY2         ;
movlw  0x40           ; draw MIDDLE of S
movwf  DIST           ;
call   M_F            ;
call   DELAY2         ;
bsf    MARKER         ; pick up pen
call   DELAY2         ;
movlw  0x40           ;
movwf  DIST           ;
call   M_B            ;
call   DELAY2         ;
movf   MALIGNB,0x00   ; Approx 2.75 inches to re-align the pen
movwf  DIST           ;
call   M_B            ;
call   DELAY2         ;
call   NDT_L          ; turn 90 degrees to the left
call   DELAY2         ;
movf   MALIGNF,0x00   ; Approx 2.75 inches to re-align the pen
movwf  DIST           ;
call   M_F            ;
call   DELAY2         ;
bcf    MARKER         ; put down pen
call   DELAY2         ;
movlw  0x40           ; draw RIGHT VERT of S
movwf  DIST           ;
call   M_F            ;
call   DELAY2         ;
bsf    MARKER         ; pick up pen
call   DELAY2         ;
movf   MALIGNB,0x00   ; Approx 2.75 inches to re-align the pen
movwf  DIST           ;
call   M_B            ;
call   DELAY2         ;
call   NDT_R          ; turn 90 degrees to the right
call   DELAY2         ;
movf   MALIGNF,0x00   ; Approx 2.75 inches to re-align the pen
movwf  DIST           ;
call   M_F            ;
call   DELAY2         ;
bcf    MARKER         ; put down pen
call   DELAY2         ;
movlw  0x40           ; draw MIDDLE of S
movwf  DIST           ;
call   M_F            ;
call   DELAY2         ;
bsf    MARKER         ; pick up pen
call   DELAY2         ;
movlw  0x40           ;
movwf  DIST           ;
call   M_B            ;
call   DELAY2         ;
movlw  0x20           ;
movwf  DIST           ;
call   M_B            ;
call   DELAY2         ;
movf   MALIGNB,0x00   ; Approx 2.75 inches to re-align the pen
movwf  DIST           ;
call   M_B            ;
call   DELAY2         ;
```

```
        call    NDT_R           ; turn 90 degrees to the right
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        return                  ;
;-------------------------------------------------------------------------------
BLANK_S bsf     MARKER          ;
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_R           ; turn 90 degrees to the right
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        movlw   0x40            ;
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        movf    MALIGNB,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_B             ;
        call    DELAY2          ;
        call    NDT_L           ; turn 90 degrees to the left
        call    DELAY2          ;
        movf    MALIGNF,0x00    ; Approx 2.75 inches to re-align the pen
        movwf   DIST            ;
        call    M_F             ;
        call    DELAY2          ;
        return                  ;
;-------------------------------------------------------------------------------
        end;
```

3. PS/2 Protocol

The PS/2 protocol allows synchronous, bidirectional bit-serial communication between the host and the pointing device. Either side may transmit a command or data byte at any time, although only one side can transmit at one time. During initialization, the host sends command bytes to the device. Some commands are followed by argument bytes. The device acknowledges each command and argument byte with an ACK ($FA) byte, possibly followed by one or more data bytes. If the host has enabled "Stream mode" transmission, then the device may send spontaneous data packets to the host describing finger motions and button state changes.

TouchPads integrated into notebook computers typically use the PS/2 protocol.

3.1. Electrical interface

The PS/2 protocol includes two signal wires as well as +5V power and ground. The signal wires, CLK and DATA, are bidirectional "open-collector" signals; they are normally held at a high (+5V) level by a 5–10K pull-up resistor on the host, but either the host or the TouchPad device can pull them low at any time. When the port is idle, both signal wires are floating high. The host can inhibit the device at any time by holding CLK low.

Note that neither side ever actively pulls CLK or DATA high; to output a logic 1, the wire is left undriven and allowed to float high. The CLK and DATA lines should have a total capacitance of no more than 500pF to ensure that the 5–10K pull-up resistor is able to drive them to a high voltage level in a reasonable time.

An external PS/2 mouse port uses a mini-DIN-6 connector with the following pinout (male connector view):

1	PS/2 DATA
2	N/C
3	Ground 0V
4	Power +5V
5	PS/2 CLK
6	N/C

Figure 3-1. PS/2 cable pinout

On the Synaptics Standard PS/2 TouchPad module TM41P*xx*134, the 8-pin FFC connector has the following pinout:

1	2	3	4	5	6	7	8
Power +5V	PS/2 DATA	PS/2 CLK	Right Switch	Left Switch	Ground 0V	N/C	N/C

Figure 3-2. PS/2 module connector pinout

Copyright © 1998 Synaptics, Inc.

3.2. Byte transmission

Each byte transmitted between the device and the host includes a start bit (a logic 0), eight data bits (LSB first), a parity bit (odd parity), and a stop bit (a logic 1). Odd parity means the eight data bits and the parity bit together contain an odd number of 1's. During transmission, the device pulses the CLK signal low for each of the 11 bits, while the transmitting party (either the host or the device) pulls the DATA wire low to signal a logic 0 or allows DATA to float high to signal a logic 1.

Between transmissions, the bus can be in one of three states:

- **Idle.** If CLK and DATA are both high, there is no activity on the bus.
- **Inhibit.** If the host is holding CLK low, the device is inhibited from transmitting data. However, internal TouchPad processing continues to occur.
- **Request to send.** If the host is holding DATA low and allowing CLK to float high, the host is ready to transmit a command or argument byte to the device.

3.2.1. Output to host

The device can transmit a byte to the host whenever the bus is idle. The device cannot transmit if the bus is inhibited or in the request-to-send state.

If the bus is inhibited, the device waits for the bus to leave the inhibit state before transmitting. The device is guaranteed to wait at least 50µs after the inhibition ends before pulling CLK low to begin the start bit. (The device *may* wait considerably longer before beginning its transmission; the host's raising of the CLK wire is not a command to the device to begin transmission, but rather a signal that the device is now allowed to transmit as soon as it is ready to do so.)

If the bus is in the host request-to-send state, the device discards its pending transmission and receives and processes the host command or argument byte. (The one exception is the Resend ($FE) command, which responds by resending the most recent transmission even if that transmission was interrupted by the Resend command itself.)

The device transmits a byte of data by pulsing CLK low and then high a total of 11 times, while transmitting the start bit, data bits, parity bit, and stop bit on the DATA wire. The host is expected to sample the DATA wire each time the CLK wire is low; the device changes the state of the DATA wire during the CLK high period.

If the host inhibits the bus by holding CLK low for at least 100µs during a device transmission, the device will recognize this and abort the transmission. The device recognizes an inhibit by noting that the CLK wire remains low during the high portion of the clock cycle. If the inhibit occurs before the rising edge of the tenth clock (the parity bit), the transmission of the byte is cancelled and the device will resend the interrupted byte as soon as the inhibit is released. (An ACK ($FA) reply to a command or argument byte is simply thrown away if cancelled, although the command being acknowledged is not cancelled, nor are the additional response bytes, if any, that follow the ACK.) If the inhibit begins after the tenth clock, the transmission is considered complete and the host must accept the transmitted byte.

The host may hold CLK low after the transmission, effectively extending clock 11, to inhibit the device from sending further data while the host processes the transmission. When the *Absolute* and *Rate* mode bits are both 1, the TouchPad reports 6 × 80 = 480 bytes per second, which allows for about 2 milliseconds per byte. Since the waveform shown in Figure 3-7 takes about one millisecond, the host should inhibit the bus for less than one millisecond per byte on average in order to achieve the maximum packet rate.

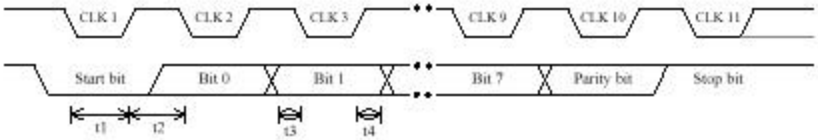


Figure 3-7. PS/2 output waveforms

In Figure 3-7, the CLK signal is low for 30–50µs (t1) and high for 30–50µs (t2) in each bit cell. DATA will be valid at least 5µs before the falling edge (t3) and at least 5µs after the rising edge (t4) of the clock. Device actions are shown in black; host actions are in gray.

3.2.2. Input from host

The host signals its intent to transmit a command or argument byte by holding CLK low for at least 100µs, then pulling DATA low and releasing CLK, thus putting the bus into the host request-to-send state. The device checks for this state at least every 10ms (t5). When the device detects a request-to-send, it pulses CLK low 11 times to receive a byte. The host is expected to change the DATA line while CLK is low; the device samples the DATA line while CLK is high. The host can abort the transmission midway through by holding CLK low for at least 100µs at any time before the eleventh CLK pulse.

After the tenth clock, the device checks for a valid stop bit (DATA line high), and responds by pulling DATA low and clocking one more time (the "line control bit"). The host can then hold CLK low within 50µs (t12) to inhibit the device until the host is ready to receive the reply. If the device finds DATA low during the stop bit, a framing error has occurred; the device continues to clock until DATA goes high, then sends a Resend to the host as described in the next section.

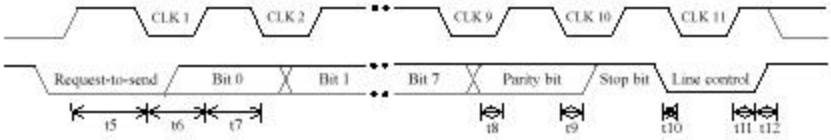


Figure 3-8. PS/2 input waveforms

In Figure 3-8, the CLK signal is low for 30–50µs (t6) and high for 30–50µs (t7) in each bit cell. DATA is sampled when CLK is high, and must be valid no later than 1µs after the rising edge of the clock (t8 ≥ –1µs, t9 ≥ 0µs). In the line control bit, DATA goes low at least 5µs before the falling edge (t10) and stays low at least 5µs after the rising edge (t11) of the clock. Device actions are shown in black; host actions are in gray.

3.2.3. Acknowledgement of commands

Each command or argument byte produces at least one response byte from the device. For every command or argument byte except the Resend ($FE) command, the response always begins with an "Acknowledge" or ACK ($FA) byte. Depending on the command, the ACK byte may be followed by additional data bytes to make up a complete response. For the Resend ($FE) command, the response sometimes does not begin with an ACK.

The device responds within 25ms, unless the host prevents it from doing so by inhibiting the bus. In multi-byte responses, the bytes of the response will be separated by no more than 20ms. The Reset ($FF) command is an exception, where the $FA and $AA bytes are separated by up to 500ms of calibration delay. The host must wait for the complete response to a command or argument before sending another byte. If the host *does* interrupt the response from a previous command with a new command, the TouchPad discards the unsent previous response as described in section 3.2.1.

If the device receives an erroneous input (an invalid command or argument byte, a byte with incorrect parity, or a framing error), the device sends a Resend ($FE) response to the host instead of an ACK. If the next input from the host is also invalid, the device sends an Error ($FC) response. When the host gets an $FE response, it should retry the offending command. If an argument byte elicits an $FE response, the host should retransmit the entire command, not just the argument byte.

On many PC's, the PS/2 port will also report a manufactured $FE response if the device does not send a response after a suitable timeout, or if the device does not respond to the request-to-send signal at all. Thus, an apparent $FE response from the TouchPad may also indicate that the TouchPad has been disconnected from the PS/2 port.

Historical notes:

Parity errors and framing errors are detected properly by current Synaptics TouchPads (version 4.*x* and later), but some earlier TouchPads ignored parity and framing errors. Likewise, earlier TouchPads did no range checking on Set Resolution and Set Sample Rate argument bytes; modern 4.*x* TouchPads will reject out-of-range Resolution arguments but still do no range checking on Sample Rate arguments.

3.3. Power-on reset

At power-on, the PS/2 device performs a self-test and calibration, then transmits the completion code $AA and ID code $00. If the device fails its self-test, it transmits error code $FC and ID code $00. This processing also occurs when a software Reset ($FF) command is received. The host should not attempt to send commands to the device until the calibration/self-test is complete.

Power-on self-test and calibration takes 300–1000ms. Self-test and calibration following a software Reset command takes 300–500ms. (In the standard Synaptics TouchPad device, the delays are nominally 750ms and 350ms, respectively.)

The Synaptics TouchPad never sends an $FC power-on/reset error code. Because the calibration algorithm is designed to adapt to environmental conditions rather than signal a hard failure, the power-on/reset response is always $AA, $00.

3.4. Command set

The Synaptics TouchPad accepts the full standard PS/2 "mouse" command set. This section describes the full set of standard mouse commands, along with any special properties of those commands as they are implemented on the Synaptics TouchPad.

If the device is in Stream mode (the default) and has been enabled with an Enable ($F4) command, then the host should disable the device with a Disable ($F5) command before sending any other command. However, if the host *does* send a command during enabled Stream mode, the device abandons any data packet or previous command response that was being transmitted at the time of the command; the device will not send any further data packets until the response to the new command is finished.

As elsewhere in this document, "$" signifies hexadecimal notation.

$FF **Reset.** Perform a software reset and recalibration as described in section 3.3 above. Response is ACK ($FA), followed by $AA, $00 after a calibration delay of 300–500ms.

$FE **Resend.** The host sends this command when it detects invalid output from the device. The device retransmits the last packet of data, for example, a three- or six-byte motion data packet, a one-byte response to the Read Device Type ($F2) command, or the two-byte completion-and-ID reset response ($AA, $00). The ACK ($FA) byte sent to acknowledge a command is not stored in any buffer or resent; however, if the last output from the device was an ACK with no additional data bytes, Resend responds with an ACK.

The device will send a Resend ($FE) to the host if it receives invalid input from the host; see section 3.2.3.

$F6 **Set Defaults.** Restore conditions to the initial power-up state. This resets the sample rate, resolution, scaling, and Stream mode to the same states as for the Reset ($FF) command, and disables the device. This command disables Absolute mode, but it leaves the rest of the TouchPad mode byte unaffected.

$F5 **Disable.** Disable Stream mode reporting of motion data packets. All other device operations continue as usual.

$F4 **Enable.** Begin sending motion data packets if in Stream mode. To avoid undesirable bus contention, driver software should send the Enable as the very last command in its PS/2 initialization sequence.

Note that a PS/2 device includes two distinct state bits: the enable/disable flag controlled by commands $F4 and $F5, and the Stream/Remote flag controlled by commands $EA and $F0. These two flags are independent, and both must be set properly (enabled, Stream mode) for the device to send motion packets. The intention is that disabled Stream mode means the host is not interested in motion packets, while Remote mode means the host

plans to poll explicitly for motion data. In practice, Remote mode and disabled Stream mode are identical in the Synaptics TouchPad.

$F3 **Set Sample Rate.** Followed by one argument byte, this command sets the PS/2 "sample rate" parameter to the specified value in samples per second. Legal values are 10, 20, 40, 60, 80, 100, and 200 (decimal) samples per second.

The Set Sample Rate command is a two-byte command. The command byte and argument byte each receive an ACK ($FA) from the device. Thus, a complete Set Sample Rate = 10 command consists of $F3 from the host, $FA from the device, $0A from the host, and $FA from the device.

The Synaptics TouchPad records the sample rate argument and will respond properly to a later Status Request ($E9) command, but this value does not actually affect TouchPad data reporting. Stream mode reporting occurs at either 40 or 80 samples per second, and is controlled by the *Rate* bit of the TouchPad mode byte; see section 2.5.

$F2 **Read Device Type.** The response is an ACK ($FA) followed by a $00 device ID byte.

$F0 **Set Remote Mode.** Switch to Remote mode, as distinct from the default Stream mode. In Remote mode, the device sends motion data packets only in response to a Read Data ($EB) command.

$EE **Set Wrap Mode.** Switch into special "echo" or "Wrap" mode. In this mode, all bytes sent to the device except Reset ($FF) and Reset Wrap Mode ($EC) are echoed back verbatim.

$EC **Reset Wrap Mode.** If the device is in Wrap mode, it returns to its previous mode of operation, except that Stream mode data reporting is disabled. If the device is not in Wrap mode, this command has no effect.

$EB **Read Data.** The device replies with an ACK ($FA) followed by a three- or six-byte motion data packet as described below in section 3.6. This command is meant to be used in Remote mode (see command $F0), though it also works in Stream mode. In Remote mode, this command is the only way to get a data packet. The packet is transmitted even if no motion or button events have occurred. The host can poll as often as PS/2 bus bandwidth allows, but since the underlying motion data are updated only 40 or 80 times per second (according to the *Rate* bit; section 2.5), there is little point in polling more often than that.

$EA **Set Stream Mode.** Switch to Stream mode, the default mode of operation. In this mode, motion data packets are sent to the host whenever finger motion or button events occur and data reporting has been enabled. Maximum packet rate is governed by the current TouchPad sample rate, described below.

Stream mode is the recommended way to use a Synaptics TouchPad; nearly all PC-compatible computers operate their pointing devices in Stream mode.

$E9 **Status Request.** Response is an ACK ($FA), followed by a 3-byte status packet consisting of the following data:

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	0	Remote	Enable	Scaling	0	Left	Middle	Right
Byte 2	0	0	0	0	0	0	Resolution	
Byte 3	Sample rate							

Figure 3-9. Standard status request response

Remote: 1 = Remote (polled) mode, 0 = Stream mode.

Enable: 1 = Data reporting enabled, 0 = disabled. This bit only has effect in Stream mode.

Scaling: 1 = Scaling is 2:1, 0 = scaling is 1:1. See commands $E6 and $E7 below.

Left: 1 = Left button is currently pressed, 0 = released.

Middle: 1 = Middle button is currently pressed, 0 = released.

Right: 1 = Right button is currently pressed, 0 = released.

Resolution: The current resolution setting, from 0 to 3 as described under Set Resolution ($E8) below.

Sample rate: The current sample rate setting, from 10 to 200 as described under Set Sample Rate ($F3) above.

For example, after Reset or Set Defaults, a Status Request command will return the bytes

$FA $00 $02 $64

indicating no buttons pressed, Stream mode, Disabled mode, Scaling 1:1, Resolution $02, and Sample rate $64 = 100 decimal.

The Status Request command returns different data in the context of a TouchPad special command sequence; see section 3.5 below.

$E8 **Set Resolution.** Followed by one argument byte, this command sets the PS/2 "resolution" parameter. Legal argument values are $00, $01, $02, and $03, corresponding to resolutions of 1, 2, 4, and 8 counts per mm, respectively.

The Synaptics TouchPad records the resolution argument and will respond properly to a later Status Request ($E9) command, but this value does not actually affect TouchPad data reporting. Sections 2.3.2, 2.4.2, and 3.6.1 describe the actual resolution reported by the TouchPad.

3.5.2. Mode setting sequence

If a Set Sample Rate 20 ($F3, $14) command is preceded by four Set Resolution commands encoding an 8-bit argument, the 8-bit argument is stored as the new value for the TouchPad mode byte as described in section 2.5 and Figure 2-14.

For example, to set the mode byte to $C1 (Absolute mode, high packet rate, Wmode enabled) one would use the sequence of commands,

$E8 $03 $E8 $00 $E8 $00 $E8 $01 $F3 $14

where the argument $C1 is encoded as follows:

($03 × 64) + ($00 × 16) + ($00 × 4) + $01 = $C1.

All ten command and argument bytes receive the usual ACK ($FA) acknowledgments. Note that, as described at the beginning of section 3.4, it is important to ensure that the device is disabled ($F5) before sending this command sequence; to receive Absolute mode packets, follow this sequence with an Enable ($F4) command.

Historical notes:

Older Synaptics TouchPads supported up to four mode bytes; the sequences to set those bytes ended with Set Sample Rate commands with arguments other than $14. On the present (4.*x*) TouchPad, sequences of four Set Resolution commands followed by a Set Sample Rate with any argument other than $14 have an undefined effect on the TouchPad and should not be used.

Some older Synaptics TouchPads also supported a second way to read or write the mode byte using PS/2 command code $E1. See section 7.1.2.

3.6. Data reporting

The Synaptics TouchPad supports two formats for motion data packets. The default Relative format is compatible with standard PS/2 mice. The Absolute format gives additional information that may be of use to TouchPad-cognizant applications.

Data packets are sent in response to Read Data ($EB) commands. If Stream mode is selected and data reporting is enabled, data packets are also sent unsolicited whenever finger motion and/or button state changes occur. Synaptics recommends using Stream mode instead of Read Data commands to obtain data packets.

During transmission of a motion packet, the individual bytes of the packet will be separated by no more than 20ms (assuming the host does not inhibit the bus). While PS/2 motion packets are lacking in explicit synchronization bits, if the host sees a delay of more than 20ms between bytes it can assume the delay comes at a packet boundary.

3.6.1. Default packet format

In the default Relative format, each motion packet consists of three bytes. The first byte encodes various status bits, and the other two bytes encode the amount of motion in X and Y that has occurred since the previous packet.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Y ovfl	X ovfl	Y sign	X sign	1	Middle	Right	Left
Byte 2	X delta							
Byte 3	Y delta							

Figure 3-17. PS/2 relative motion packet

Y ovfl: 1 = Y delta value exceeds the range –256…255, 0 = no overflow. When this bit is set, the reported Y delta will be either –256 or +255.

X ovfl: 1 = X delta value exceeds the range –256…255, 0 = no overflow. When this bit is set, the reported X delta will be either –256 or +255.

Y sign: 1 = Y delta value is negative, 0 = Y delta is zero or positive.

X sign: 1 = X delta value is negative, 0 = X delta is zero or positive.

Middle: 1 = Middle button is currently pressed, 0 = released.

Right: 1 = Right button is currently pressed, 0 = released.

Left: 1 = Left button is currently pressed (or gesture in progress), 0 = released.

X delta: This is the amount of motion ΔX that has occurred in the X (horizontal) direction since the last motion data report. This byte and the "X sign" bit of byte 1 combine to form a nine-bit signed, two's-complement integer. Rightward motion is positive, leftward is negative.

Y delta: This is the amount of motion ΔY that has occurred in the Y (vertical) direction. Upward motion is positive, downward is negative.

Note that the three button state bits reflect a combination of physical switch inputs and gestures. The "left button" bit is set if either the left physical switch is closed, or a tap or drag gesture is in progress. (If the *DisGest* mode bit is set, then the "left button" bit reports only the state of the physical left switch.) The "right button" bit is set only if the right physical switch is closed. Because standard Synaptics TouchPads only support two buttons, the "middle button" bit is always zero.

The X and Y deltas report an accumulation of all motion that has occurred since the last packet was sent, even if host inhibition has prevented packet transmission for some time. Also, any host command except Resend ($FE) clears the motion accumulators, discarding any motion that had occurred before the command but that had not yet been sent in a packet.

The X and Y deltas have a resolution of about 240 DPI on a standard Synaptics pad; see section 2.6.3 for further details.

3.6.2. Absolute packet format

When Absolute mode is enabled, each motion report consists of six bytes. These bytes encode the absolute X, Y location of the finger on the sensor pad, as well as the Z