

ELE538 Microprocessor Systems

Lab 7

The *eebot* Guider

Peter Hiscocks
Department of Electrical and Computer Engineering
Ryerson Polytechnic University
phiscock@ee.ryerson.ca
August 26, 2002

Contents

1	The <i>Guider</i> Concept	2
1.0.1	Guider Transient Response	3
1.0.2	Line Tracker	4
1.0.3	Pattern Detector	4
2	Setting Up the Guider	5
2.1	Checking the LED Pattern	5
2.2	The Background and Line	5
2.3	Checking Guider Operation	5
2.4	Guider Calibration: The Line Follower	6
3	Guider Software	8
3.1	Guider API	8
3.2	The A/D Converter Hardware	8
4	Assignment: Read Guider	10
5	The <i>read-guider</i> Code	12
6	Appendix: Signal Averaging	23

List of Figures

1	Guider Sensor Locations, Top View & Side View	2
2	Guider Sensors	3
3	Guider Sensor Template	7
4	The API for <i>Read Sensors</i>	8
5	General Purpose Register	9
6	<i>OPTION</i> Register	9
7	A/D Control Register	10

1 The *Guider* Concept

The *eebot* robot is intended to find and follow a line on the floor surface. The line is a 3/4 inch wide red electrical tape on a background of black floor tiles¹. For the purpose of following this line, *eebot* has a *guider* section mounted at the underside of the bow area.

The sensors are 6 CdS (Cadmium Sulphide) photoresistors that are high resistance in darkness and low resistance when illuminated. The sensors are labelled **A** through **F**, as shown in figure 1.

The small circles are high-intensity red LEDs, which illuminate the floor surface under each sensor, so there is one LED per sensor. The large circles are the sensors themselves. Lines show LED-sensor pairs.

The relationship between a typical sensor and LED is shown in the bottom half of the diagram. The LEDs are mounted on the top side of the guider circuit board and illuminate the floor surface via a hole in the guider circuit board. This arrangement allows the CdS cells to be very close to the floor surface, which improves the sensitivity and reduces the effect of ambient light. As well, the LEDs are protected from being knocked out of alignment.

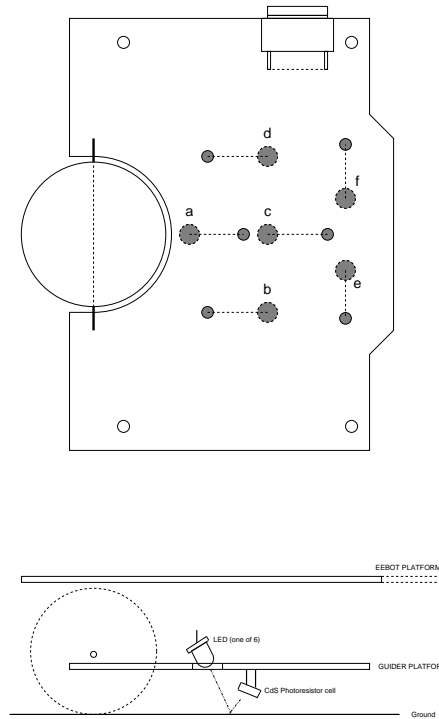


Figure 1: Guider Sensor Locations, Top View & Side View

Each CdS photoresistor is driven by a constant current source so that it generates a voltage proportional to its resistance. Since the resistance decreases with increasing light level, so does the sensor voltage.

¹Black tape on a light background works equally well

A simplified schematic of the Guider Sensors is shown in figure 2 on page 3.

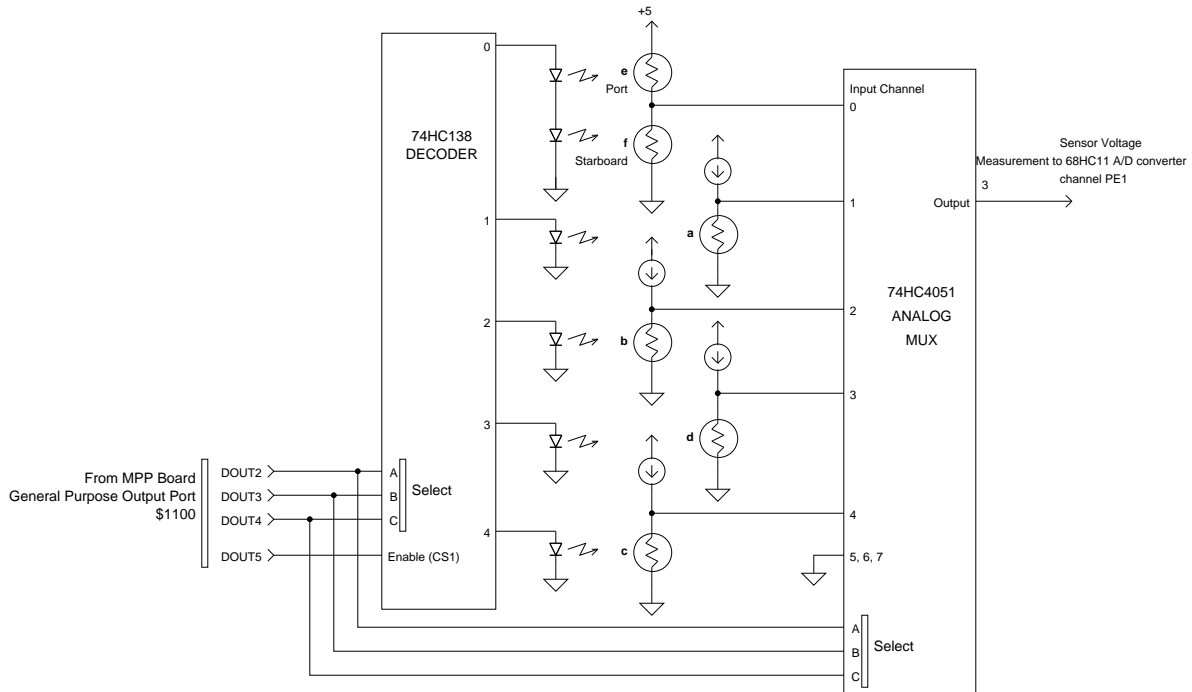


Figure 2: Guider Sensors

The LEDs and CdS cells are selected by the same signals, the three general purpose digital outputs D2, D3 and D4. The 74HC138 decoder that drives the LEDs is enabled and disabled by the general purpose digital output D5.

One of the voltages developed across the sensors is selected by a 74HC4051 analogue multiplexer and directed to the 68HC11 A/D input channel PE1. The same three general purpose digital outputs D2, D3 and D4 selecting the LED also select the corresponding photosensor. Notice that 3 of the 8 possible sensor inputs are unused and connected to ground, so they should read zero.

In general, the LEDs will be enabled so D5 should be set to a high (logic 1, +5V) level. The D2, D3 and D4 signals will have some value between 000 and 100 (msb-nsb-lsb) to illuminate one of the LEDs and simultaneously select one of the guider signals. The guider software could scan the 5 sensor signals or repeatedly read one sensor, whatever is required.

To detect the background illumination, the LEDs can be disabled by lowering the D5 signal. The sensor readings would then indicate read the lighting without any LED illumination. This could be used as a baseline or *noise* value and subtracted from the value obtained when the LED is activated.

(The original guider design illuminated all LEDs at once, but that caused problems in light leaking from one LED into several sensors. Furthermore, there was no way to determine background illumination level.)

1.0.1 Guider Transient Response

The CdS photoresistors respond quickly to an increase in illumination (decreasing resistance) but rather slowly to decreasing illumination (increasing resistance). Consequently, for accurate measurements the photoresistor should be allowed time to darken between each measurement. The darkening time constant is in the order of 50 milliseconds². Consequently, the most rapid allowable scan rate to scan one sensor is 20 times per second. If all five sensors are being scanned, the fastest allowable scan rate increases because a given sensor can be darkening while others are being read. Then the minimum scan time per sensor is 10 milliseconds, or about 100 times per second.

The differential line tracker pair of cells **ab** actually recovers faster than a single cell, and the effective time constant is about 25 milliseconds. (This is because one of the pair is brightening while the other darkens.) So the line tracker signal could be scanned as fast as 40 times per second, which may be important for rapid response in steering.

1.0.2 Line Tracker

Sensors **e** and **f** used for *line tracking*. They are spaced 0.75 inches apart, so that they are centered over the two edges of electrical tape line when the robot is centred over the line. The two sensors are a voltage divider, so that the voltage at the centre tap is determined by the ratio of the two sensor resistances. The guidance software of the robot should position the robot so that the voltage at the centre tap from voltage divider sensors **e** and **f** is exactly midway between 5 volts and zero, ie, 2.5 volts. Then the two sensor resistances are equal, the light level on the two cells is equal, and the two cells are equally over the black line and light background.

Notice that the absolute value of the resistance of sensors **e** and **f** is unimportant: it is their ratio that determines the output voltage. Consequently, they should largely ignore changes in ambient light level and this does indeed prove to be the case.

However, if both sensors are over a completely black background or a completely white background, they will also generate 2.5 volts. Consequently, some other method is required for ensuring that the robot is over a valid line, and that is the function of the pattern detector.

1.0.3 Pattern Detector

Sensors **a**, **b**, **c** and **d** form a *pattern detector* for various line configurations. For example, if sensors **a** and **c** detect a dark line while sensors **b** and **d** detect a light background, then the robot is probably over a valid line and the information from the line tracker can be used for guidance.

If the pattern detector sensors all detect a light background, then the robot is off line, and should begin to search for a line. The line tracker cannot be used.

If the pattern detector senses a black line at right angles to the current line (for example, sensor **a** reads light, sensor **b** reads dark, sensor **c** reads dark, sensor **d** reads light, so the line angles off to port), then the robot should advance until the drive motors are over the bend in the line and then pivot counter-clockwise until the spur line is under sensors **a** and **c**. Similar interpretation strategies should enable the pattern detector to sense a **T** junction or a line that simply comes to an end.

The sensors of the pattern detector each detect absolute light level. A constant current is passed through each sensor and the voltage generated across the sensor is then proportional to its resistance. The sensor currents are

²This was determined by aiming an LED directly at a photoresistor and pulsing it from a function generator while measuring the voltage across the photoresistor with an oscilloscope.

each adjusted so that the sensor generates about 1.8 volts over a light surface and 3 volts over a dark surface³. One of the voltages developed across the sensors is selected by an analogue multiplexer and directed to the 68HC11 A/D input channel PE1.

The selection of one of the 5 sensor voltages is determined from the three general purpose digital outputs D2, D3 and D4. Notice that 3 of the 8 possible analog inputs are unused and connected to ground so they should read zero.

2 Setting Up the Guider

In this section, we describe how to test the operation and calibration of the *eebot* line sensor system.

2.1 Checking the LED Pattern

A template for checking the guider LED illumination pattern is shown in figure 3 on page 7 .

1. Set up the *eebot* so that it is powered up and connected to a 68HC11 microprocessor.
2. Load the program `read-guider.s19` and run it with a `G 6070` command. (If this doesn't work, check the `read-guider.lst` file – the starting address may have changed.)
3. Make a full-sized copy of the template of figure 3. Place it under the *eebot* guider, lined up with the front sphere and rear driving tires as marked on the template.
4. Check that the 6 LED illumination circles match up with the large circles on the template.
5. If an LED is not aligned with its illumination circle, advise the lab supervisor and he/she will adjust the aiming of the LED.

2.2 The Background and Line

A number of line and background colours and textures were tested to determine the combination for line detection by the guider. The surface to be provided for the line following project consists of black floor tiles, each one foot square, each with a line of red electrician's tape, 3/4 inch wide. The tiles may be assembled in different arrangements to create different robot courses.

2.3 Checking Guider Operation

Here is a quick procedure for checking that the guider is working, at least to some extent.

1. Connect the *eebot* chassis to the MPP board 68HC11 development system and ensure that both are powered up correctly. Turn on the LOGIC switch on the robot chassis.

³As part of the operating procedure, the operator should check the A/D reading from each of the pattern detector sensors over light and dark surfaces, and calibrate the software thresholds accordingly.

2. Load the program `read-guider.s19` and run it with a `G 6070` command. (If address 6070 doesn't work, check the `read-guider.lst` file – the starting address may have changed.) You should see a display of the five sensor readings A, B, C, D and E-F on the microcomputer LCD.

The guider sensor readings are shown in hexadecimal format, so we will use hexadecimal numbers (unless otherwise noted) throughout this section on calibration.

3. Apply the *finger test* to the guider. Make sure that the robot guider is over a light coloured surface. With an extended finger, carefully cover up each of the four different absolute sensors A, B, C and D in turn. As you do so, the reading for that sensor should increase noticeably, at least 50 counts (one volt). A variation of one or two counts is not significant – it may be due to noise in the A/D signals. Be careful not to disturb the aiming of the sensors or LEDs.
4. For the differential steering sensor pair E-F, covering up sensor E should cause the fifth reading to decrease. Covering up sensor F should cause the fifth reading to increase. Be careful not to dislodge the shades on sensors E and F: the differential sensor will not work correctly without them.

If this test fails on any or all of the sensors, get help: the guider is not functioning for some reason.

2.4 Guider Calibration: The Line Follower

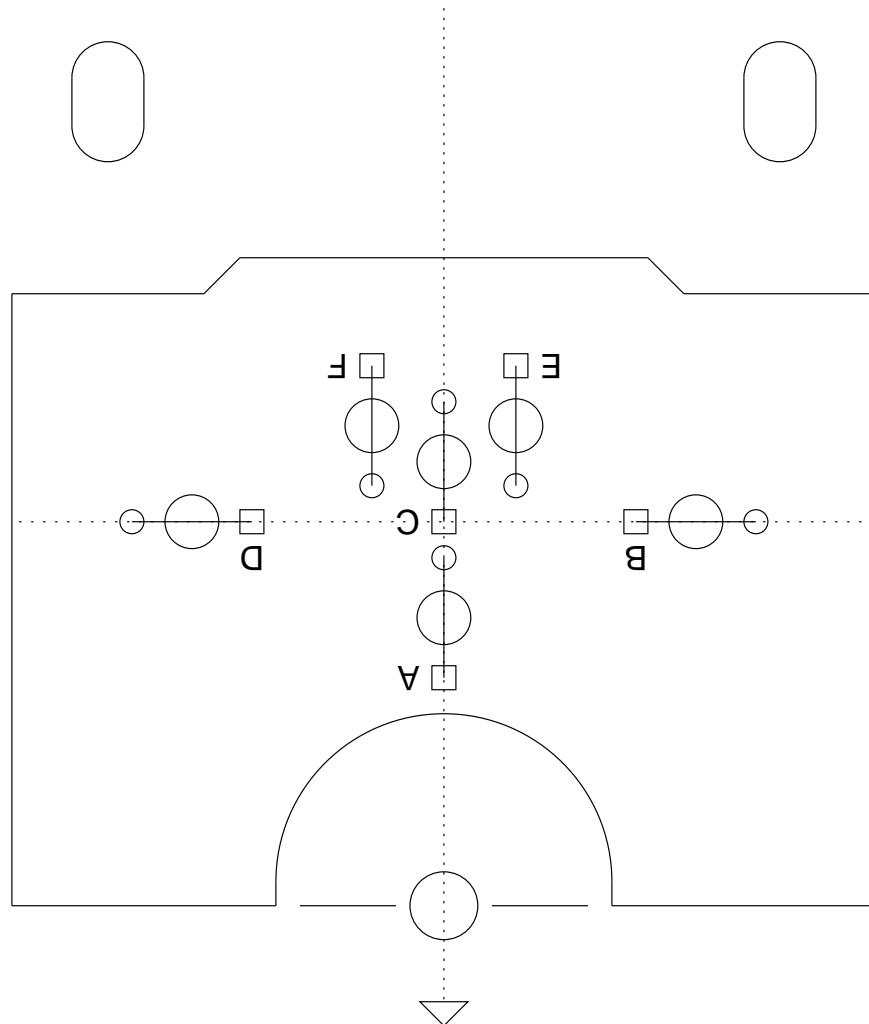
Once the robot is over the line and following it approximately (which can be determined from the absolute sensors A and C), the line-follower sensor E-F can be used to generate a correction signal to steer the robot.

As the robot strays from the line one of the line-follower E-F sensors will get darker and the other one lighter, changing the E-F sensor reading.

Suppose the E-F sensor reading varies between 28 with the robot pointing toward the PORT side to a value of A8 with the robot pointing toward the STARBOARD side.

Consequently, the robot should try to steer so that the line sensor has a value halfway between the extremes of 28 and A8, ie, about 68. Values above this should cause the robot to steer in one direction, and below this it should steer in the other direction. If you use this feature to steer the robot, you would build this value into your computer program.

If the line sensor is to be used as part of a negative feedback system, it will be necessary to know the *sensor gain*. This is *the amount by which the sensor reading changes divided by the distance over which this change occurs*. For example, if the sensor reading changes 28 and A8 when the line sensor moves sideways by 0.5cm, then the sensor gain (in decimal units) is 256 counts per centimetre.



eebot Guider Alignment Template

- LEDs ○
- Photoresistors □
- Target areas ○

Position template under guider area.
Adjust LED's to illuminate target areas.

Figure 3: Guider Sensor Template

3 Guider Software

In this section, we describe the `read-guider` routine. You will need to use this routine (possibly with modifications) in your robot guidance project.

3.1 Guider API

At its most abstract level, we can picture that there is some routine that reads the guider sensors and puts them in a specific location in memory. The way in which this routine is accessed by other software routines is known as the *Application Program Interface* (API) for that routine. (The name derives from the common situation where a particular routine is part of the operating system of the computer. When an application must access some feature of the hardware, then it does so through the API of that particular routine.)

To improve documentation, we can relabel the names to something more informative than **A**, **B** and so as shown in the following table:

Designation	Name	Function
E-F	SENSOR_LINE	Line sensor
A	SENSOR_BOW	Front sensor
B	SENSOR_PORT	Port sensor
C	SENSOR_MID	Middle sensor
D	SENSOR_STAR	Starboard sensor

Then the header for the routine, which defines the interface for this routine (the API), might look like the code of figure 4.

```
*-----  
*                               Read Sensors  
*  
* This routine reads the eebot guider sensors and puts the results in RAM  
* registers.  
* Passed:      None  
* Returns:     Sensor readings in:  
*              SENSOR_LINE  
*              SENSOR_BOW  
*              SENSOR_PORT  
*              SENSOR_MID  
*              SENSOR_STAR
```

Figure 4: The API for *Read Sensors*

The control software main loop would periodically call `READ_SENSORS` to update the sensor readings. Routines that use the sensor signals can do so simply by reading these RAM registers.

3.2 The A/D Converter Hardware

The 68HC11 A/D converter was described and applied previously in Lab 3. In this application, we will use the A/D in a slightly different mode.

Referring to figure 2 on page 3, all the guider signals are brought into Channel PE1 of the A/D converter (AN1) via a multiplexer on the guider board. Consequently, the `READ_SENSORS` routine should do three things:

- enable the guider LEDs by setting GPOUT D5 to 1
- select the appropriate sensor (by setting the appropriate bit pattern in GPOUT, which selects the appropriate multiplexer channel on the robot guider) and
- then read Channel PE1 of the A/D converter.

The bit assignments in the GPOUT register are shown in figure 5. Bits 2,3 and 4 control which of the sensor voltage is routed to channel 1 of the 68HC11 A/D converter. When changing these bits, it is important not to affect the motor direction bits, also in GPOUT.

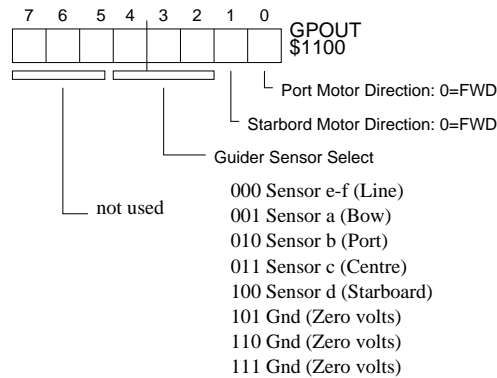


Figure 5: General Purpose Register

Now we need to consider how to set up the A/D converter. The two registers controlling the A/D converter are OPTION and ADCTL.

Only two bits in the OPTION register are relevant to the A/D converter, as shown in figure 6. These two bits are set up in the Buffalo monitor to power up the A/D converter and select the processor E clock. If the program is later made stand-alone (the program runs from a processor reset, without the monitor present), then there must be instructions to initialize the option register. The safest policy is to set those bits in this program, even though the monitor program has already set them.

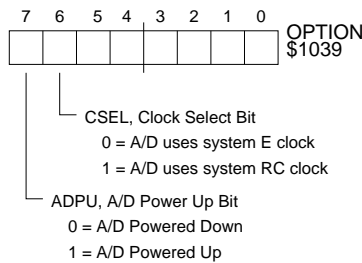


Figure 6: OPTION Register

The OPTION register need only be loaded once. The ADCTL register must be loaded very time an A/D conversion is to take place⁴. The bit assignments of the ADCTL register are shown in figure 7 on page 10.

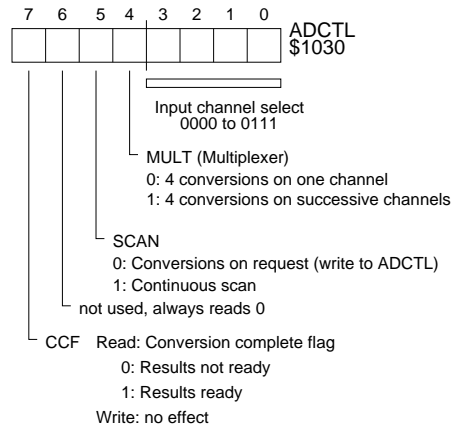


Figure 7: A/D Control Register

The appropriate value to store in ADCTL, working from MSBit to LSBit, can be determined as follows:

- Bit 7 cannot be changed by writing to it, so what we write to that position is unimportant: let's choose a 0.
- Bit 6 has no function, choose 0.
- Bit 5: We want conversions to occur *on request* by writing to ADCTL and then waiting for bit 7 to go high, so bit 5 should be 0.
- Bit 4: We want conversions on one channel, so bit 4 is 0.
- Bits 3 through 0: These bits should select channel AD1, so they contain 0001.

Consequently, the conversation with the A/D hardware will consists of writing 00000001 to ADCTL and then looping while waiting for bit 7 to go high. In *single conversion* mode, the A/D takes 32 cycles per conversion and performs 4 conversions on the selected channel, so a complete conversion will take $4 \times 32 = 128$ machine cycles to complete.

Now we can write pseudocode for READ_SENSORS, as shown in figure 8.

The complete listing for read-guider is shown in section 5.

4 Assignment: Read Guider

Demonstrate operation of the provided program read-guider-2.asm. Be prepared to answer technical questions on the hardware and programming of the guider.

⁴Assuming that we are in *single conversion* scan mode, which is the most appropriate mode for this application.

```
Loop      Initialize a pointer PTR into the RAM at the start of the Sensor Array storage
          Store %00000001 to the ADCTL
          Repeat
            Read ADCTL
          Until Bit 7 of ADCTL == 1
          Store the contents of ADRL at the pointer PTR
          If the pointer is at the last entry in Sensor Array, then exit
          Else increment the pointer and loop again.
```

Figure 8: *Read Sensors* Pseudocode

5 The read-guider Code

```
*-----
*           'Read Guider' Demo Routine
*
* Reads the eebot guider sensors and displays the values
* on the Liquid Crystal Display.
*
* Peter Hiscocks
* 14 October, 2001
*
* Version 2:
* 15 November 2001
*
* Modified from version 1 to support selection of the individual LED
* associated with a sensor, to reduce crosstalk from unselected sensor
* LEDs.
* The guider hardware was modified with the addition of a 74HC138 decoder that
* drives the individual LEDs, so that only the LED associated with a given
* sensor is ON when that sensor is being read.
* This requires that the software be modified to enable the decoder with bit 5
* in GPOUT ($1100).
* The CdS cells are very slow in responding to changes in light, so a 20
* millisecond delay is inserted between selecting a particular sensor and
* reading its value.
* Substantial improvement:
*     Draws less battery current for longer life
*     Creates less heat in the 5V logic regulator
*     Much greater contrast between dark and light readings
*
* Assembling on other machines
* -----
* The assembler as11 varies in behaviour between different machines. On the
* Solaris machines at Ryerson, for example, the assembler generates errors
* for variables longer than 13 characters. If you assemble this program to
* generate an .s19 file, check the .lst file carefully for error messages
* and, if necessary, shorten the variable names that are problematic.
*
* Overview:
* -----
* This program is intended as a test routine for the guider sensors of the
* eebot robot and contains routines that will be useful in the robot
* guidance project.
* The guider consists of four absolute brightness sensors and one
* differential brightness pair of sensors. They are arranged at the nose of
* the robot in the following pattern (viewed from above):
*
*           A
*          B C D
*         E-F
*
* The sensors are cadmium sulphide (CdS) photoresistive cells, for which the
* resistance increases with decreasing light level. The absolute cells
* A,B,C and D are driven from a constant current source, and the voltage
* across the cell measured via the 68HC11 A/D converter channel PE1. Thus
* the sensor reading increases as the sensor becomes darker (over a black
* line, for example).
*
* The differential sensor E-F is a voltage divider with the two CdS cells E
* and F separated 0.75 inches, which is the width of electrical tape. It is
* intended to be used to track the edges of the electrical tape 'line' once
* the absolute cells have 'found' a black line. Cell E is at the top of the
* divider, so as the reading from this sensor increases, cell E is becoming
* lighter, ie, cell E is straying onto the white background.
* Simultaneously, cell F is becoming darker as it moves over the black
* tape, and its resistance is increasing, aiding the same effect. The
* differential action should ignore ambient light.
```



```

FCB 0 ;

    ORG $6070 ; Start of program text
*-----
*
*           Display Sensors

MAIN JSR G_LEDS_ON ; Enable the guider LEDs
JSR READ_SENSORS ; Read the 5 guider sensors
JSR G_LEDS_OFF ; Disable the guider LEDs
JSR DISPLAY_SENSORS ; and write them to the LCD
JSR LCD_DELAY ; Delay to avoid display artifacts
JMP MAIN ; Loop forever

*-----
*
*           Read Sensors
*
* This routine reads the eebot guider sensors and puts the results in RAM
* registers.
*
* Note: Do not confuse the analog multiplexer on the Guider board with the
* multiplexer in the 68HC11. The guider board mux must be set to the
* appropriate channel using the SELECT_SENSOR routine. The 68HC11 always
* reads the selected sensor on the 68HC11 A/D channel PE1.
*
* The A/D conversion mode used in this routine is to read the first four A/D
* channels PE0,1,2,3 into HC11 registers ADR1,2,3,4. The only result used
* in this routine is the value from PE1, read from ADR2. However, other
* routines may wish to use the results in ADR1, 3 and 4.
* Consequently, Scan=0, Mult=1 and Channel=0000 for the A/D control word.
*
* Passed:      None
* Returns:     Sensor readings in:
*             SENSOR_LINE (0) (Sensor e/f)
*             SENSOR_BOW (1) (Sensor a)
*             SENSOR_PORT (2) (Sensor b)
*             SENSOR_MID (3) (Sensor c)
*             SENSOR_STBD (4) (Sensor d)
* Note:
* The sensor number is shown in brackets
*
* Algorithm:
* Initialize the sensor number to 0
* Initialize a pointer into the RAM at the start of the Sensor Array storage
* Loop Store %00010000 to the ADCTL (to select PE1 and start a conversion)
* Repeat
*     Read ADCTL
*     Until Bit 7 of ADCTL == 1 (at which time the conversion is complete)
*     Store the contents of ADR1 at the pointer
*     If the pointer is at the last entry in Sensor Array, then
* Exit
*     Else
* Increment the sensor number
* Increment the pointer
* Loop again.

READ_SENSORS CLR SENSOR_NUM ; Select sensor number 0
LDX #SENSOR_LINE ; Point at the start of the sensor array

RS_MAIN_LOOP LDAA SENSOR_NUM ; Select the correct sensor input
JSR SELECT_SENSOR ; on the hardware
JSR DELAY_20_MS ; Delay to allow the sensor to stabilize

LDAA #%00010000 ; Start A/D conversion on PE0 through PE3
STAA ADCTL
RS_AD_LOOP LDAA ADCTL ; Repeat until A/D signals done
BPL RS_AD_LOOP

LDAA ADR2 ; A/D conversion is complete in ADR2

```

```

STAA 0,X ; so copy it to the sensor register
CPX #SENSOR_STBD ; If this is the last reading
BEQ RS_EXIT ; Then exit

INC SENSOR_NUM ; Else, increment the sensor number
INX ; and the pointer into the sensor array
BRA RS_MAIN_LOOP ; and do it again

RS_EXIT RTS

*-----
*                               Select Sensor
*
* This routine selects the sensor number passed in ACCA. The motor direction
* bits 0, 1 and the unused bits 5,6,7 in the same machine register GPOUT are
* not affected.
* Bits 2,3,4 are connected to a 74HC4051 analog mux on the guider board,
* which selects the guider sensor to be connected to PE1.
*
* Passed: Sensor Number in ACCA
* Returns: Nothing
* Side Effects: ACCA is changed
*
* Algorithm:
* First, clear the sensor bits 2,3,4 in the shadow register to zeros
*   by ANDing it with the mask 11100011. The zeros in the mask clear
*   the corresponding bits in the shadow register. The 1's have no effect.
* Next, move the sensor selection number left two positions to align it
*   with the correct bit positions for sensor selection.
* Clear all the bits around the (shifted) sensor number by ANDing it with
*   the mask 00011100. The zeros in the mask clear everything except
*   the sensor number.
* Now we can combine the sensor number with the GPOUT shadow register using
*   logical OR. The effect is that only bits 2,3,4 are changed in the shadow
*   register, and these bits now correspond to the sensor number.
* Finally, save the shadow register to the hardware.

SELECT_SENSOR PSHA ; Save the sensor number for the moment

LDAA GPOUT_SHADOW ; Clear the sensor selection bits to zeros
ANDA #%11100011 ;
STAA GPOUT_SHADOW ; and save it back into the shadow reg

PULA ; Get the sensor number
ASLA ; Shift the selection number left, twice
ASLA ;
ANDA #%00011100 ; Clear irrelevant bit positions

ORAA GPOUT_SHADOW ; OR it into the sensor bit positions
STAA GPOUT_SHADOW ; Keep an up-to-date copy in the shadow reg
STAA GPOUT ; Update the hardware
RTS
*
*-----
*                               Display Sensor Readings
*
* Passed: Sensor values in RAM locations SENSOR_LINE through SENSOR_STBD.
* Returns: Nothing
* Side: Everything
*
* This routine writes the sensor values to the LCD. It uses the 'shadow
* buffer' approach. The display buffer is built by the display controller
* routine and then copied in its entirety to the actual LCD display.
* Although simpler approaches will work in this application, we take that
* approach to make the code more re-useable.
* It's important that the display controller not write over other
* information on the display, so writing the display has to be centralized

```

```

* with a controller routine like this one.
* In a more complex program with additional things to display on the LCD,
* this routine would be extended to read other variables and place
* them on the display. It might even read some 'display select' variable to
* determine what should be on the display.

* For the purposes of this routine, we'll put the sensor values on the LCD
* in such a way that they (sort of) mimic the position of the sensors, so
* the display looks like this:
* 01234567890123456789
*  FF_____
*  PP_MM_SS_LL_____

* Where FF is the front sensor, PP is port, MM is mid, SS is starboard and
* LL is the line sensor.

* The corresponding addresses in the LCD buffer are defined in the following
* equates (In all cases, the display position is the MSDigit).

DP_FRONT_SENSOR EQU TOP_LINE+3
DP_PORT_SENSOR EQU BOT_LINE+0
DP_MID_SENSOR EQU BOT_LINE+3
DP_STBD_SENSOR EQU BOT_LINE+6
DP_LINE_SENSOR EQU BOT_LINE+9

DISPLAY_SENSORS JSR CLR_LCD_BUFF ; Write 'space' characters to the buffer

LDAA SENSOR_BOW ; Get the FRONT sensor value
JSR BIN2ASC ; Convert to ascii string in D
LDX #DP_FRONT_SENSOR ; Point to the LCD buffer position
STD 0,X ; and write the 2 ascii digits there

LDAA SENSOR_PORT ; Repeat for the PORT value
JSR BIN2ASC
LDX #DP_PORT_SENSOR
STD 0,X

LDAA SENSOR_MID ; Repeat for the MID value
JSR BIN2ASC
LDX #DP_MID_SENSOR
STD 0,X

LDAA SENSOR_STBD ; Repeat for the STARBOARD value
JSR BIN2ASC
LDX #DP_STBD_SENSOR
STD 0,X

LDAA SENSOR_LINE ; Repeat for the LINE value
JSR BIN2ASC
LDX #DP_LINE_SENSOR
STD 0,X

JSR LCD_CLR_HOME ; Clear the display, home the cursor

LDX #TOP_LINE ; Now copy the buffer top line to the LCD
JSR LCD_WRT_STR

LDAA #LCD_SEC_LINE ; Position the LCD cursor on the second line
JSR LCD_POS_CRSR

LDX #BOT_LINE ; Copy the buffer bottom line to the LCD
JSR LCD_WRT_STR

RTS

*-----
* Clear Display Buffer

```

* This routine writes 'space' characters (ascii 20) into the LCD display
 * buffer in order to prepare it for the building of a new display buffer.
 * This need only be done once at the start of the program. Thereafter the
 * display routine should maintain the buffer properly.

```
CLR_LCD_BUFF    LDX #CLEAR_LINE
LDY #TOP_LINE
JSR STRCPY
```

```
CLB_SECOND     LDX #CLEAR_LINE
               LDY #BOT_LINE
JSR STRCPY
```

```
CLB_EXIT RTS
```

```
*-----
*                               String Copy
```

```
* Copies a null-terminated string (including the null) from one location to  

* another
```

```
* Passed: X contains starting address of null-terminated string  

*        Y contains first address of destination
```

```
STRCPY PSHX ; Protect the registers used
PSHY
PSHA
STRCPY_LOOP LDAA 0,X ; Get a source character
STAA 0,Y ; Copy it to the destination
BEQ STRCPY_EXIT ; If it was the null, then exit
INX ; Else increment the pointers
INY
BRA STRCPY_LOOP ; and do it again
STRCPY_EXIT PULA ; Restore the registers
PULY
PULX
RTS
```

```
*-----
*                               Binary to ASCII
```

```
* Converts an 8 bit binary value in ACCA to the equivalent ASCII character 2  

* character string in accumulator D  

* Uses a table-driven method rather than various tricks.
```

```
* Passed: Binary value in ACCA  

* Returns: ASCII Character string in D  

* Side Fx: ACCB is destroyed
```

```
HEX_TABLE FCC '0123456789ABCDEF'; Table for converting values
```

```
BIN2ASC PSHA ; Save a copy of the input number on the stack
TAB ; and copy it into ACCB
ANDB #%00001111 ; Strip off the upper nibble of ACCB
CLRA ; D now contains 000n where n is the LS nibble
ADD #HEX_TABLE ; Set up for indexed load
XGDX
LDAA 0,X ; Get the LS nibble character

PULB ; Retrieve the input number into ACCB
PSHA ; and push the LS nibble character in its place
RORB ; Move the upper nibble of the input number
RORB ; into the lower nibble position.
RORB
RORB
ANDB #%00001111 ; Strip off the upper nibble
CLRA ; D now contains 000n where n is the MS nibble
ADD #HEX_TABLE ; Set up for indexed load
```

```

                XGDX
                LDAA 0,X                ; Get the MSnibble character into ACCA
PULB ; Retrieve the LSnibble character into ACCB

```

```
RTS
```

```

*-----
*                               Routines to control the Liquid Crystal Display

```

```

*-----
*                               Initialize the LCD

```

```

* This routine should be executed once on reset. The sequence of events was
* determined from the Hitachi manual for the LCD controller.

```

```

LCD_INIT LDAA #INTERFACE ; Write a control byte (see equates)
STAA LCD_CONTROL

```

```
JSR LCD_DELAY ; Delay for at least 4.1 milliseconds
```

```

LDAA #INTERFACE ; Do it again (that's what the manual says!)
STAA LCD_CONTROL

```

```
JSR LCD_DELAY ; Delay at least 100 microseconds
```

```

LDAA #INTERFACE ; Do it again (third time lucky!)
STAA LCD_CONTROL

```

```

LDAA #CURSOR_OFF ; Write a control byte (see equates)
JSR LCD_WRT_CTRL

```

```

LDAA #SHIFT_OFF ; Write a control byte (see equates)
JSR LCD_WRT_CTRL

```

```

LDAA #CLEAR_HOME ; Clear the display, home the cursor
JSR LCD_WRT_CTRL

```

```
RTS ; The display now works correctly.
```

```

*-----
*                               Write a Control Byte to the LCD

```

```

* This routine waits until the LCD is ready to receive information and then
* writes the passed byte to the control register.

```

```

* A note on checking the LCD flag bit:
* When the LCD is busy, the MSBit of the control register reads as a 1, so
* the byte is considered negative. (Remember, for 2's complement numbers,
* the MSBit is a 1 when the number is negative.) In the routine, the BMI
* instruction causes the machine to re-read the control register until the
* MSBit is 0, ie, the byte reads as a positive number. This only works
* because the flag is in the MSBit position. If it were in some other
* position, it would be necessary to mask out the bit and then check the
* result with BNE or BEQ.

```

```

* Passed: Control byte in ACCA
* Returns: Nothing
* Side Effects: None

```

```

LCD_WRT_CTRL PSHB ; Save the B register
LCD_WCON_LOOP LDAB LCD_CONTROL ; Wait until the LCD is ready
BMI LCD_WCON_LOOP ; ie, the flag is cleared
STAA LCD_CONTROL ; Write the control byte
PULB ; Restore the B register
RTS

```

```

*-----
*                               Write a Data Byte to the LCD

```

```

* This routine waits until the LCD is ready to receive information and then
* writes the passed byte to the data register (ie, to show the character on
* the display). The character is printed at the current position of the
* display cursor, and the cursor incremented. (That mode was set up in the
* initialization.)

```

```

* Passed: Data byte in ACCA
* Returns: Nothing
* Side Effects: None

```

```

LCD_WRT_CHAR PSHB ; Save the B register
LCD_WCHAR_LOOP LDAB LCD_CONTROL ; Wait until the LCD is ready
BMI LCD_WCHAR_LOOP ; ie, the flag is cleared
STAA LCD_DATA ; Write the data byte
PULB ; Restore the B register
RTS

```

```

*-----
*                               Write a String to the LCD

```

```

* This routine writes a null-terminated string to the LCD, starting at the
* current position of the cursor.
* Control characters such as 'carriage return' and 'line feed' are written
* as raw characters to the display. There is no attempt to process them
* properly.

```

```

* Passed: Pointer to first character in string, in X index register
* Returns: Nothing
* Side Effects: The X index register is changed.

```

```

LCD_WRT_STR PSHA ; Protect ACCA
WLS_LOOP LDAA 0,X ; Get the character at the string
CMPA #NULL ; If it's the null terminator,
BEQ WLS_EXIT ; then exit
JSR LCD_WRT_CHAR ; Else write it to the display
INX ; Increment the string pointer
BRA WLS_LOOP ; and do it again
WLS_EXIT PULA ; Restore ACCA
RTS

```

```

*-----
*                               Position the Cursor

```

```

* This routine positions the display cursor in preparation for the writing
* of a character or string.
* For a 20x2 display:
* The first line of the display runs from 0 .. 19.
* The second line runs from 64 .. 83.

```

```

* The control instruction to position the cursor has the format
*
*      laaaaaaa
* where aaaaaaa is a 7 bit address.

```

```

* Passed: 7 bit cursor Address in ACCA
* Returns: Nothing
* Side Effects: None

```

```

LCD_POS_CRSR ORAA #%10000000 ; Set the high bit of the control word
JSR LCD_WRT_CTRL ; and set the cursor address
RTS

```

```

*-----
*                               Home Cursor, Clear Display

```

```

* Clears the display and homes the cursor.

```

```

* Passed: Nothing
* Returns: Nothing
* SideFX: Nothing (except CCR, of course)

```

```

LCD_CLR_HOME PSHA
LDAA #CLEAR_HOME ; Clear the display, home the cursor
JSR LCD_WRT_CTRL
PULA
RTS

*-----
*           LCD Delay Routine

* A very simple delay routine that delays for 2^16 counts. At a 1
* microsecond clock, this is 3 microseconds for the DEX and 3 microseconds for
* the BNE instruction. So the total delay is about 1/3 second per call to
* this routine. It's only executed twice at startup, so the delay is not
* critical as long as it's large enough.

* Passed: Nothing
* Returns: Nothing
* Side effects: None

LCD_DELAY PSHX ; Protect the X register
LDX #$FFFF ; Initialize the loop counter
LCD_DLY_LOOP DEX ; If the counter is not zero
BNE LCD_DLY_LOOP ; then loop again
PULX ; Restore the X register
RTS ; Else return

*-----
*           20 Millisecond Delay

* This routine delays for approximately 20 milliseconds to allow a CdS
* sensor to stabilize after the corresponding LED has been turned on.
* The processor clock is assumed to be 1 microsecond.

* Passed: Nothing
* Returns: Nothing
* Sides: Nothing

DELAY_20_MS PSHX ; Protect the X register
LDX #$1046 ; Initialize the loop counter
DLY_20_LOOP DEX ; If the counter is not zero
BNE DLY_20_LOOP ; then loop again
PULX ; Restore the X register
RTS ; Else return

*-----
*           Guider LEDs ON

* This routine enables the guider LEDs so that readings of the sensor
* correspond to the 'illuminated' situation.

* Passed: Nothing
* Returns: Nothing
* Side: GPOUT bit 5 is changed

G_LEDS_ON PSHA ; Protect A
LDAA GPOUT_SHADOW
ORAA #%00100000 ; Set bit 5
STAA GPOUT_SHADOW
STAA GPOUT
PULA ; Restore A
RTS

*-----
*           Guider LEDs OFF

* This routine disables the guider LEDs. Readings of the sensor
* correspond to the 'ambient lighting' situation.

```

```
* Passed:      Nothing
* Returns:     Nothing
* Side:        GPOUT bit 5 is changed

G_LEDS_OFF     PSHA      ; Protect A
                LDAA     GPOUT_SHADOW
AND# #11011111 ; Clear bit 5
                STAA     GPOUT_SHADOW
                STAA     GPOUT
PULA ; Restore A
                RTS
```

6 Appendix: Signal Averaging

If a particular signal is corrupted by random noise, the variance of the noise may be reduced by collecting a series of samples and then averaging them together. Each time the number of samples is doubled, the random component is reduced by 3db (a factor of 0.707). Consequently, 4 samples decrease the noise by a factor of 6db, or 0.5. This strategy may be applied to the values read by an A/D channel in order to reduce the noise on the reading.

For this to apply, the samples must be *uncorrelated*, that is, random with respect to each other. To see why this is necessary, consider that a (desired) DC signal is applied to an A/D input. To this is added a very slow-changing (undesired) noise signal. The A/D clock is 1μ second and according to the data book each conversion takes 32 cycles so each conversion takes 32μ seconds. If the noise signal does not change significantly in this 32μ second interval between samples it will not average to zero.

As well, a periodic signal which is synchronous with the samples will also not average to zero. (Think of an AC noise signal where the peak of each cycle coincides with the sampling of the waveform: this will always return the same value, so the average will not go to zero.) In essence, this is a problem in digital signal processing, in which the sampling interval affects the frequencies filtered. Without knowing the properties of a noise signal, we can't predict how effective it will be to average 4 sensor readings. However, it's easy to do in software, it will not cannot worsen the noise level and might improve it.

When the 68HC11 A/D converter is set up to read one channel, it does four successive reads and conversions and puts the results in the four HC11 registers AN0 through AN3. It is then a simple matter of summing the four readings and dividing the result by 4 to obtain an average. (You may wish to add this feature to the READ_SENSORS routine.)

Before using software averaging to reduce noise, electronic methods should be used to ensure that the signal into the A/D is as noise-free as possible. For example, it may be necessary to shield a signal from electromagnetic coupling, move the analog ground connection, or use electronic filtering on the signal to remove noise. Software averaging is not a good substitute for these measures.