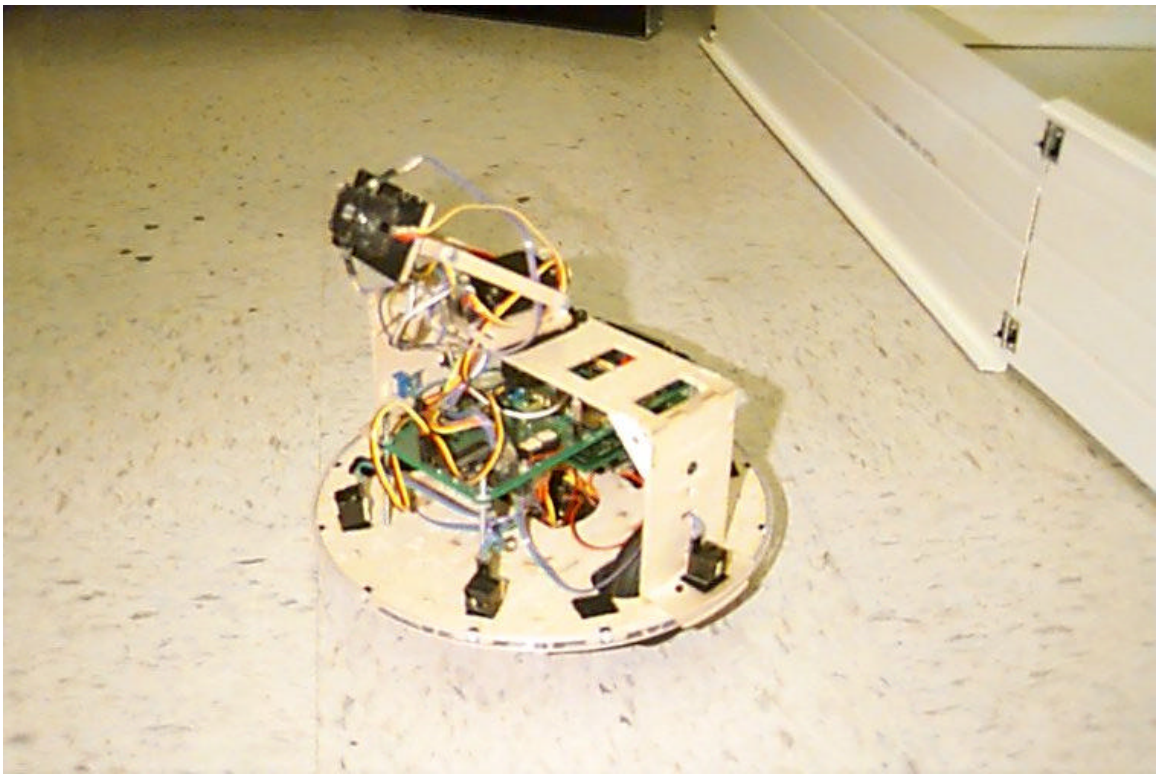
University of Florida
Department of Electrical and Computer Engineering
Intelligent Machine Design Laboratory
EEL 5666

Final Report:
HERMES



Student Name: Brent Leary
Date: 4/21/99
T.A.s: Scott Jantz
Aamir Qaiyumi

Table of Contents

ABSTRACT
Hermes is designed to track and follow a moving light source while avoiding obstacles. In order to accomplish this, a special sensor equipped with five CDS cells is mounted on top of a talrik platform. The light-sensor is actuated by two servos that enable it to move horizontally and vertically in space. Its goal is to keep itself pointing at a single source of light (i.e. a lamp). The Talrik's body moves in the direction the light-sensor is pointing while trying to avoid any objects in its path.

EXECUTIVE SUMMARY

Hermes was developed to determine whether real time light tracking in three dimensions with any degree of precision was possible. He is controlled by a HC6811 microprocessor with an ME11 daughter card. The platform consists of two main parts, the Talrik body and the pan-tilt head that holds the light-sensor. The pan-tilt head is mounted on top of the Talrik body. It is actuated by two servos and can rotate horizontally and vertically 180 degrees. This allows the light-sensor to face any point in a hemisphere above it. The body of the Talrik is equipped with IR emitters, IR detectors, and bump switches for collision avoidance and detection. Hermes behaviors include collision detection, collision avoidance, light sensing, light following and light finding.

INTRODUCTION

Hermes is an autonomous mobile agent designed for light navigation. He consists of a Talrik platform and is controlled with and HC6811 microprocessor mounted in an EVBU board with and attached ME11 daughter card. The objective of his design is to be able to pinpoint the location of an ambient light source. This ability allows a wide range of behaviors to be implemented. My original intent was mapping, but since the Vector 2X digital compass and the motors both use the same pins for control in PORTD, I chose light following based behaviors. I will include the digital compass as one of my sensors even though it was not used in the final demo because I had it fully working on my platform when I discovered the resource conflict.

INTEGRATED SYSTEMS

Hermes has a total of 20 sensors.

- 10 bump switches
- 4 IR detectors
- 5 CDS cells
- 1 digital compass

Their readings are fed into the HC6811 microprocessor through the header on the EVBU board. All of them except the digital compass use the HC's analog port. Hermes has an output latch at 0x6000 and an analog multiplexor chip in order to expand the number of analog inputs to 14. The output latch is used to control the analog multiplexor and the digital compass. All ten bump switches use only a single analog port. This is done by wiring them into a voltage divider

different voltage when bumped. The four IR detectors are hacked to provide an analog output and are attached to the HC's analog port. The IR detectors are used in obstacle avoidance and are used to detect the 40kHz light given off by the 12 IR emitters evenly spaced around the diameter of Hermes. The IR emitters are attached to the ME11 and are controlled with an output latch at 0x7000. The 5 CDS cells are wired into a voltage divider circuit with the other resistance being 5k. The result of the voltage divider circuit is fed into an analog port through the analog multiplexor chip. Two servos using OC4 and OC5 pins PA3 and PA4 respectively control the pan-tilt head which houses the 5 CDS cells in the light-sensor. The motors are controlled by OC2 and OC3 using pins PD1, PD2, PA2, and PA3. The digital compass communicates the heading to the HC6811 through the SPI system. The current heading can be sampled at about 10hz. Hermes uses his CDS cells in the light-sensor to keep the pan-tilt head pointing at a moving light source. By knowing the current position of the servos used in the pan-tilt head, the direction of the light relative to Hermes current orientation is known and therefore corrections to the motors can be made so that he turn in the direction of the light.

MOBILE PLATFORM

Hermes uses a Talrik platform. The pan-tilt array is mounted on top of a bridge that straddles the platform's body. This gets it above the wiring and circuitry so it can rotate freely. The platform is a pre-established design that best suited my intention of testing out a pan-tilt head.

ACUTATION

into the servo. The servos are hacked into motors by taking out the physical stops and the potentiometer. A full hack is needed because the heat generated by running the motors for long periods of time will fry the potentiometer's circuitry.

SENSORS

IR Sensor Suite

Hermes has four IR detectors, two facing forward and two facing backward on the Talrik platform. Three are connected to lines five, six, and seven of my analog multiplexor and the other is connected to analog port PE1. The detectors have been hacked to allow analog output roughly within the range of 80 – 120 on an analog port.

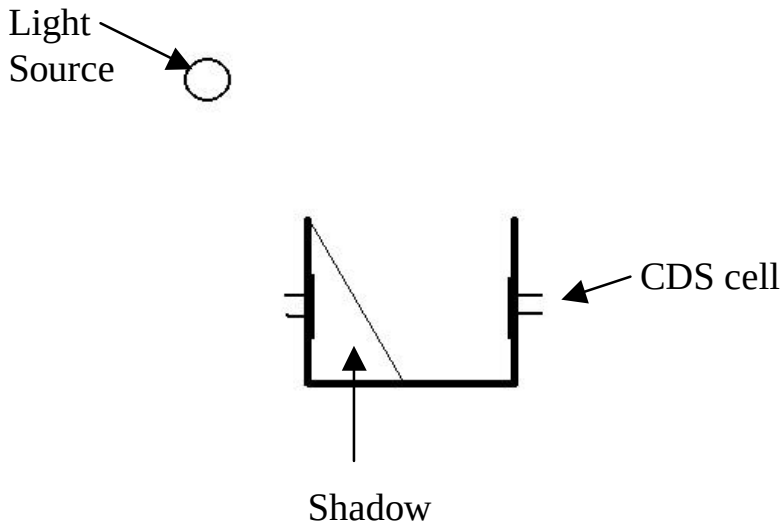
Twelve evenly spaced IR emitters encircle the platform. They are broken into four groups of three – forward left, forward right, backward left, and backward right. A group is powered by a 40kHz signal generated by dividing the 68HC11's E-clock with a 74HC390 decade counter on the ME11 daughter board.

Bump Sensor Suite

Ten evenly spaced bump switches surround Hermes' perimeter, and are surrounded by a rigid bumper. Each bump switch is wired as shown in Appendix C. When a value other than 0 is read on PE2, Hermes has bumped something, and that value carries information about which bump switch was hit.

CDS Sensor Suite

The CDS sensors are mounted in a light-sensing device best described as wooden box with one side missing. Circular holes are cut into the sides of the box and the CDS cells are inserted facing the box's interior. The box is then mounted on the pan-tilt head that is able to rotate horizontally and vertically. The sensor's goal is to accurately keep itself pointing at a light source as Hermes moves about his environment. In order to accomplish this task, the sensor must be able to tell which direction the light moving across its dish and be able to make adjustment that will keep the light directly in the center of the sensor. The following diagram illustrates the light-sensors use:



The direction of the light source is binary, left or right. By having Hermes try to keep the CDS cell values the same with servo commands, the sensor will follow the light source.

Digital Compass Sensor Suite

companies web page (Appendix A). The compass can be used to keep Hermes from going in circles without having to calibrate his motors, but its primary use will be to give the heading from a point of origin.

BEHAVIORS

Hermes reactions to different combinations of sensor stimulus are called behaviors. The following is a list of behaviors and conditions that evoke them.

- A bump switch is pressed - back up about a foot and rotates 90 degrees
- A single IR senses an object – turn away from the object
- Both IR sense an object – stop and rotate until both IR do not sense an object
- No light present – positions both servos for the pan-tilt head at 90 degrees and stop motors
- Light present – actuate servos to follow light, actuate motors to turn towards the direction the horizontal servo is pointing
- Light directly overhead – stop motors

Precedence for the behaviors is created using if-then-else statements in icc with the lowest precedence behavior appearing last. Each behavior, when active, has control of the motors. To control the pan-tilt head's servos, the difference between the left and right CDS cells is used to determine the horizontal servo's direction and the difference between the top and bottom CDS cells is used to control the vertical servo's direction. The sign of difference (positive or negative) determines in which direction the servo is incremented.

EXPERIMENTAL LAYOUT AND RESULTS

IR

To determine the range of a hacked IR detector, I used IC to read an analog port with an attached IR detector. One reading was taken with no IR emitters turned on and the other with an IR emitter directly facing it. A range of 80 – 120 was found.

To determine if my IR emitters were indeed emitting IR light, I held them up to a video camera available in the IMDL lab. The monitor attached to the video camera will display 40kHz IR light in the visible spectrum making it easy to test if the emitters were operational.

Bump

Making use of IC and the analog(#) function, I depressed each individual bump switch while taking readings off analog port PE2. Each switch returned a distinct value when pressed.

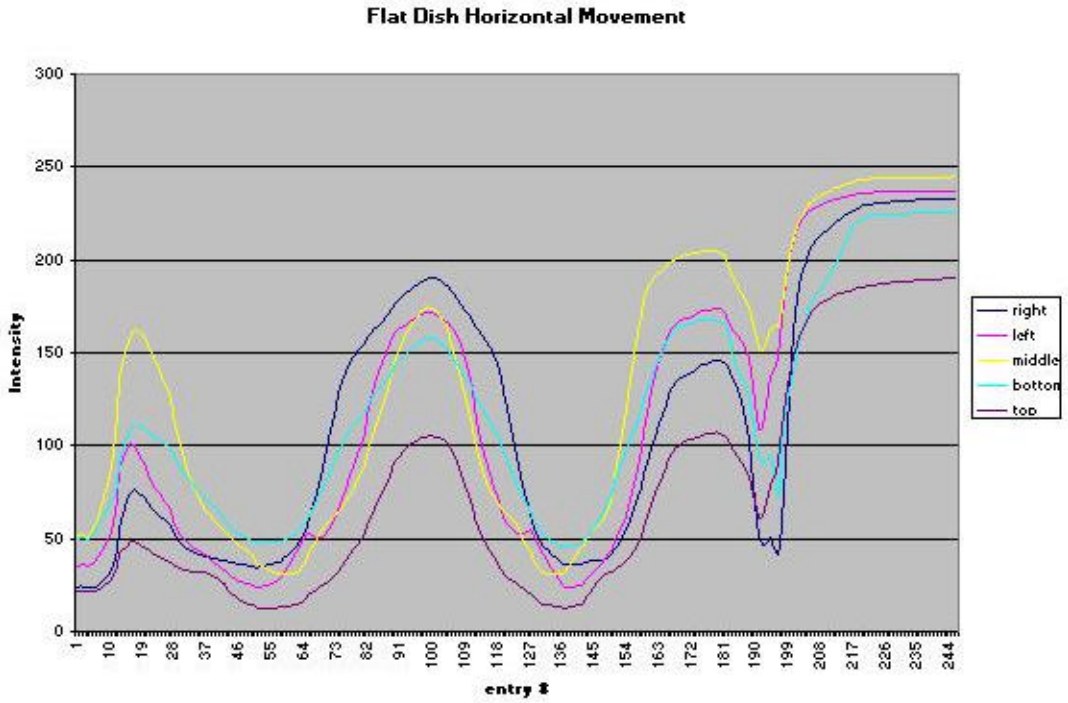
Light-Sensor

Hermes' light-sensor has undergone three revisions with the final one being used:

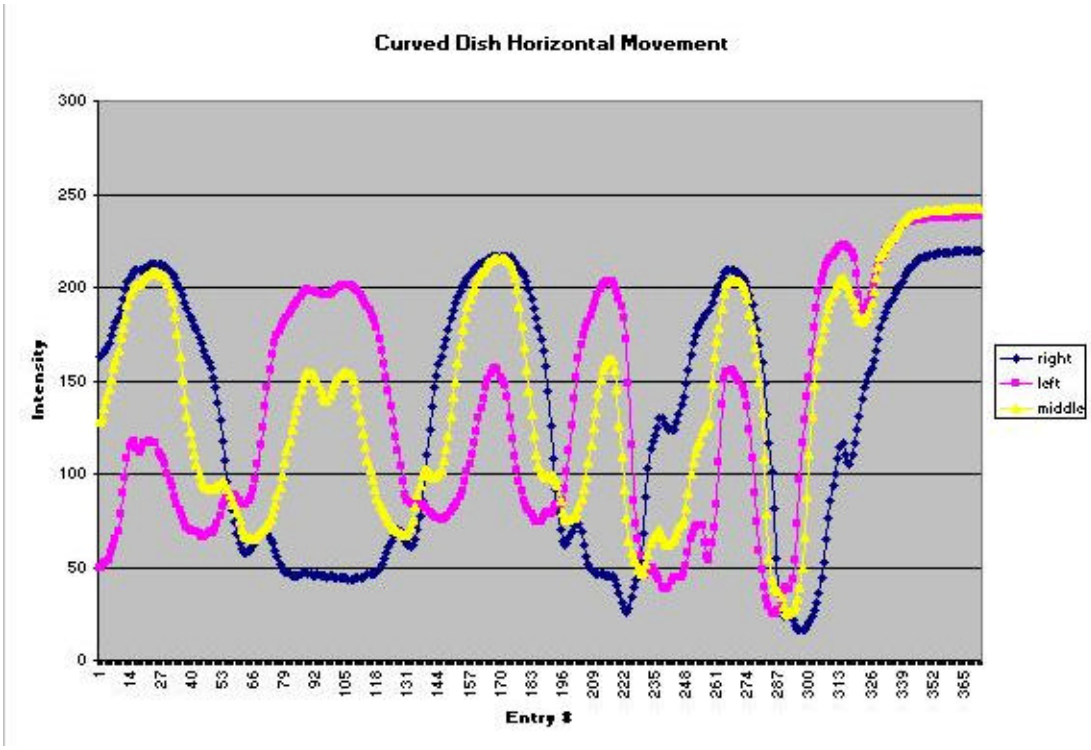
- A flat dish with collimated CDS cells arranged in an X pattern
- A dish with slightly raised sides using collimated CDS cells arranged in an X pattern
- A box painted black using CDS cells with no collimation

The first two underwent testing for my sensor report, and as a result of those tests, I felt a re-design was needed, so the third and final design was used on Hermes.

To test the first two sensors, I propped them up and approximately five feet away moved a light perpendicularly across their face. Using half inch collimated CDS cells inserted into a flat dish,



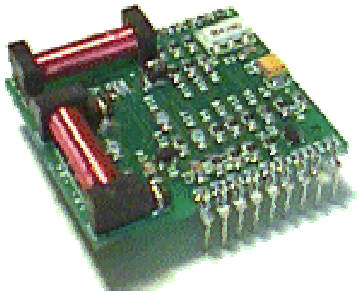
Notice the direction of the light's movement across the sensor is impossible to tell because all of the CDS cell's intensities vary together. However, a dish with slightly raised sides results in the following data:



Notice that the right sensor's intensity reading is low, while the middle and left sensor's readings are high. From this difference in intensity the direction of the lights movement can be obtained. The focus of this version with only slightly inset sides was too broad. I noticed that the only visible difference between the CDS cells values resulted from the shadow being cast by the inset sides, but in order for the CDS to be in shadow the light source needed to be at an extreme angle to the sensor. Raising the inset to a 90 angle on the final version gave the sensor a very narrow focus because, unless the light is directly above the sensor, one of the CDS cells will be in direct light and the other in shadow.

Digital Compass

Vector 2x



The Vector 2x digital compass interfaces with the HC6811 through its SPI system. I configured the Vector 2x as the master and the 68HC11 as the slave. I chose this configuration because it was the simplest and I have no other devices that need to use the SPI system. The wiring schematic is provided in appendix C. The compass is set to give continuous reading through the SPI system, so when the HC6811 wants to take a reading, two bytes are received through the HC6811's SPDR register. The Vector 2x's manual states that data is ready on a rising clock

rising edge, and the CPHA bit set to one, so the first edge of the SCLK began a transfer not the SS line, in the SPCR register. The compass works with both bits zero for binary mode, and CPOL set to 1 for BCD mode. I believe the CPOL settings are backward. The compass is accurate when Hermes is rotating, staying in the same position on the floor. However, in the imdl lab when running him across the floor while reading the compass heading output to hyperterminal, the heading can vary by 10 degrees due to the metal in the floor. The Vector 2x has plus or minus one degree of error, but the error does not build because it can be polled for a new heading every .2 seconds. A weak magnetic field will slow down this sampling rate. The compass can compensate for static magnetic field distortions such as those created by hard iron, but cannot compensate for the non-static distortions of an A/C motor or soft iron (ferrous metal). The digital compass on the SPI system uses the same port as the motors when using the ME11 daughter card. They both use PORTD pins 1 and 2.

CONCLUSION

Hermes is a robot capable of following a moving light while avoiding any obstacles in its path. He integrates the behaviors of collision avoidance, obstacle avoidance and light tracking. The pan-tilt head with mounted light sensor operated as planned, tracking a single ambient light source as it moved about in space. With this ability, an autonomous mobile agent is free to wander about while having the constant knowledge of the direction and distance of a light source. Future improvements to the light-sensor could include replacing the resistors in the CDS cells voltage divider circuit with digital potentiometers. This would give the light-sensor the

REFERENCES

Hermes would not have been realized without the help from the following sources:

- The IMDL T.A.’s Scott Jantz and Aamir Qaiyumi
- My classmates, especially Jamie, who provided their input and assistance
- My friend Peter Collins

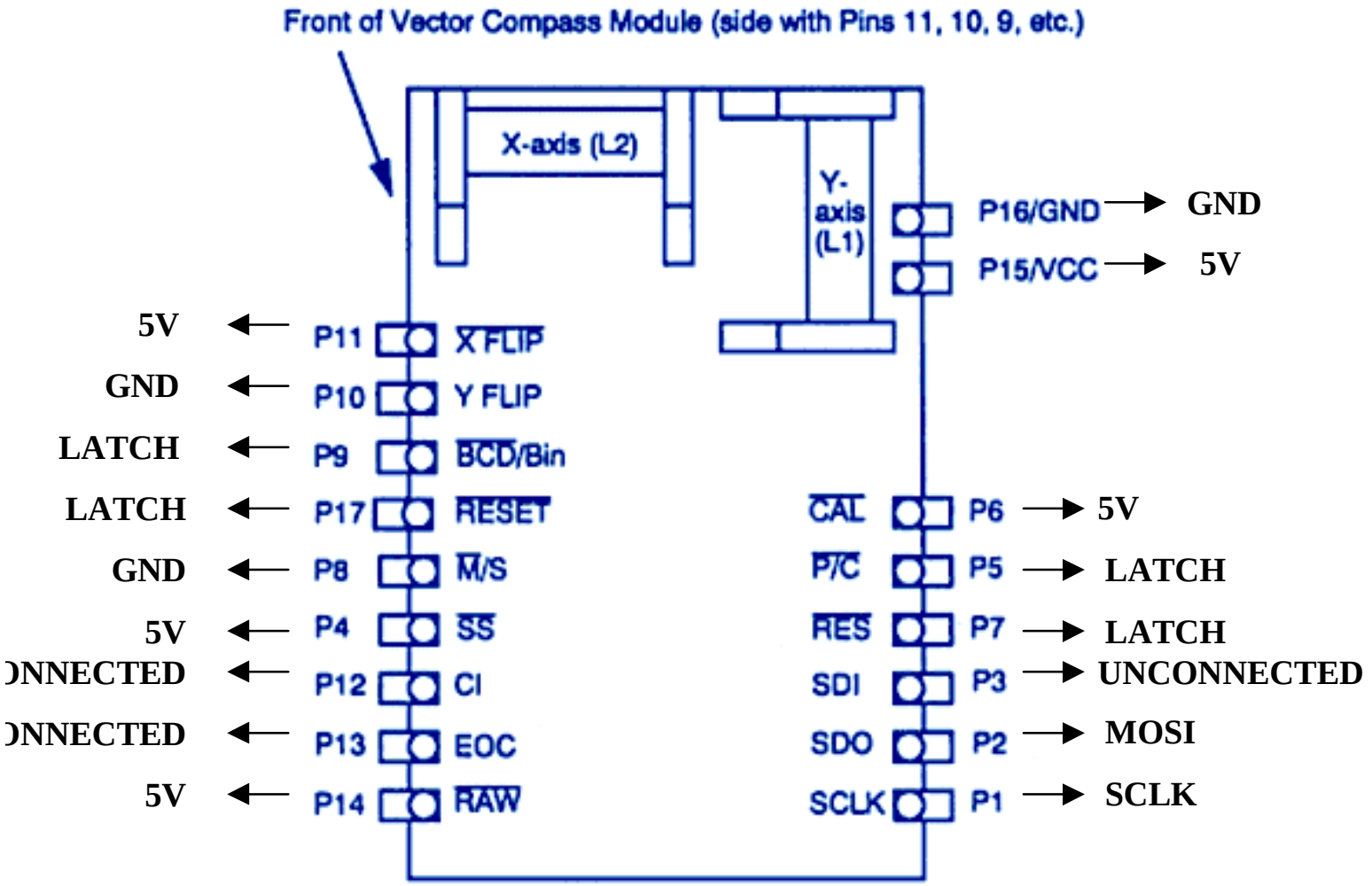
Appendix A: VENDOR INFORMATION

Vendor Information	Parts Purchased	Price
Mekatronics 316 NW 17th St., Suite A Gainesville, FL 32603 (407)672-6780 http://www.mekatronics.com	• LED emitter • Detector: Sharp GPIU58Y • Bump Switches • CDS cells • Diamond Servo • Dubro model aircraft wheel	$0.75 $3.00 $0.75 Free $11.50 $3.00
Precision Navagation, Inc. 555 Skylane Blvd., Suite E Santa Rosa, CA 95403 (707) 566-2261 http://www.precisionnavigation.com	• VECTOR 2X™	$50.00

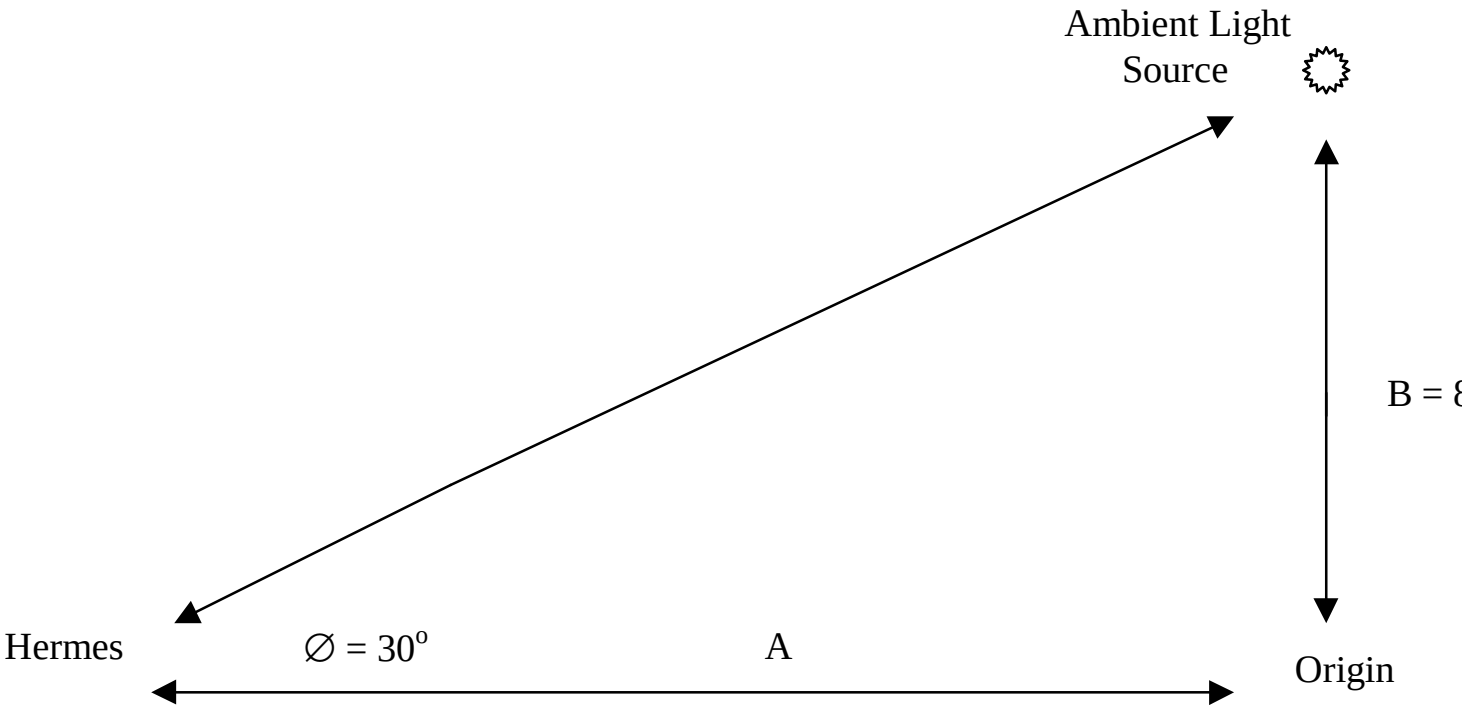
Appendix B: DIAGRAMS

Diagrams

Vector 2x Schematic



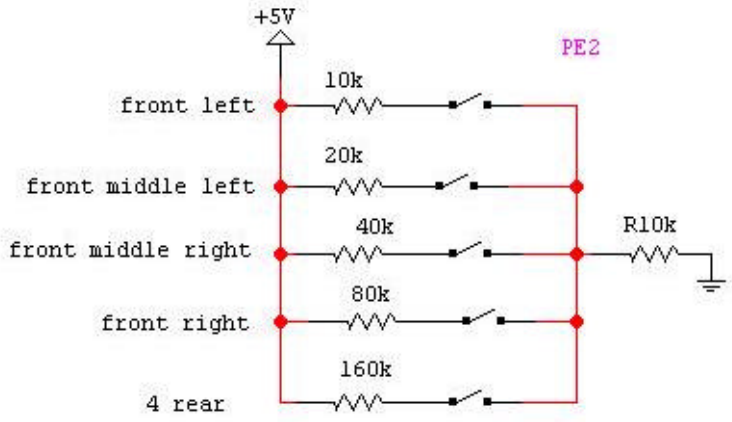
Derivation of Distance A



From the angle ∅ derived from the sensor array and the known height of a light source, the distance A o a point of origin can be calculated as follows:

$$\tan 30^{o} = 8' / A$$
$$A = 8' / \tan 30^{o}$$
$$A = 13.85'$$

Bump Switch Schematic



x C: CODE

```
C
t Leary
/99
es' Brain
es.c


**************Includes*****************/

e "me11oc45.h"

*************Constants******************/

 IRLATCH    *(unsigned char *)0x7000

 OUTLATCH   *(unsigned char *)0x6000

 LATCHSTATE *(unsigned char *)0x010F

 LEFT  1

 RIGHT 0

 HORZ  1

 VERT  0

 RIGHTBIAS  0   /* CDS cell biases */

 LEFTBIAS   0

 TOPBIAS    0

 BOTTOMBIAS 10

 VERTTHRESH 20

 HORZTHRESH 25

 HINC       3

 VINC       1

 LIGHTHEIGHT 6  /* feet */

 MOTORINC    1

 IRDIFF      7



*************Variables*****************/

IRhigh;  /* higher IR threshold */
```

```
sensor data */
ct cds

TCH  0x00   0x01     0x02   0x03    0x04 */
nt   left,  right,  middle,  top,  bottom; /* Current */
nt  dleft_right, dtop_bottom;              /* difference */
_data;

os current positions */
ct servo

nsigned int horz;
nsigned int vert;
vo_data;

rs PW */
ct motor

nt lpw;
nt rpw;
or_data;

*****UPDATE SENSORS************/
 IRbump_sense()

Rbump_data.bump = analog(2);

RLATCH = 0x04;        turn on fron left IR  */
Rbump_data.fl_eye = analog(1);  /*( - IRlow)/scale; */


RLATCH = 0x08; */
ET_BIT(LATCHSTATE, 0x07);   /* 7 */
UTLATCH = LATCHSTATE;
Rbump_data.fr_eye = analog(0);

RLATCH = 0x02; */
LEAR_BIT(LATCHSTATE, 0x01); /* 6 */
UTLATCH = LATCHSTATE;
Rbump_data.l_eye = analog(0);

RLATCH = 0x01; */
LEAR_BIT(LATCHSTATE, 0x02); /* 5 */
ET_BIT(LATCHSTATE, 0x01);
UTLATCH = LATCHSTATE;
Rbump_data.r_eye = analog(0);

RLATCH = 0x00;  turn off IR */


*****UPDATE SENSORS************/
 cds_sense()

ET_BIT(LATCHSTATE, 0x04);
LEAR_BIT(LATCHSTATE, 0x03); /* 4 */
```

```
UTLATCH = LATCHSTATE;
ds_data.middle = analog(0);

LEAR_BIT(LATCHSTATE, 0x02); /* 1 */
ET_BIT(LATCHSTATE, 0x01);
UTLATCH = LATCHSTATE;
ds_data.top = analog(0) + TOPBIAS;

LEAR_BIT(LATCHSTATE, 0x01); /* 0 */
UTLATCH = LATCHSTATE;
ds_data.bottom = analog(0) + BOTTOMBIAS;

* CALCULATE DIFFERENCES */
ds_data.dleft_right = cds_data.left - cds_data.right;
ds_data.dtop_bottom = cds_data.top  - cds_data.bottom;


********ACTUATION***************/
 motor_actuate()

otor(LEFT, motor_data.lpw + 2);
otor(RIGHT, motor_data.rpw);


********ACTUATION***************/
 servo_actuate()

ervo(HORZ, servo_data.horz);
ervo(VERT, servo_data.vert);



*****TERMINAL OUTPUT**********/
 cds_output()

ds_sense();
umout(cds_data.left);
rite(" ");
umout(cds_data.middle);
rite(" ");
umout(cds_data.right);
rite("    ");
umout(cds_data.top);
rite(" ");
umout(cds_data.middle);
rite(" ");
umout(cds_data.bottom);
rite("    ");
umout(cds_data.dleft_right);
rite(" ");
umout(cds_data.dtop_bottom);
rite("\r");


*****TERMINAL OUTPUT**********/
 IRbump_output()
```

```
rite(" ");
umout(IRbump_data.bump);
rite(" ");
umout(IRlow);
rite("\r");


********BEHAVIORS***************/
 sensor_light_follow()
 updates CDS cell info and
 actuates horizontal and vertical servos
 depending on the CDS cell information
 255 is absolute dark
  0  is absolute light */

ds_sense();  /* update cds cell's info */

* Calc Horz Movement */
f(servo_data.vert < 2750)

  if (cds_data.dleft_right > HORZTHRESH && servo_data.horz <= 4300)
  {/* right cds darker */
     servo_data.horz = servo_data.horz + HINC;
  }


  else if (cds_data.dleft_right < -HORZTHRESH && servo_data.horz >= 1000)
  {/* left cds darker */
     servo_data.horz = servo_data.horz - HINC;
  }

 /* Reverse for opposite vert position */
lse

  if (cds_data.dleft_right > HORZTHRESH && servo_data.horz >= 1000)
  {
     servo_data.horz = servo_data.horz - HINC;
  }

  else if (cds_data.dleft_right < -HORZTHRESH && servo_data.horz <= 4300)
  {
     servo_data.horz = servo_data.horz + HINC;
  }


* Calc Vert Movement */

f (cds_data.dtop_bottom > VERTTHRESH && servo_data.vert <= 4500)
/* bottom cds darker */
  servo_data.vert = servo_data.vert + VINC;



lse if (cds_data.dtop_bottom < -VERTTHRESH && servo_data.vert >= 1000)
/* top cds darker */
  servo_data.vert = servo_data.vert - VINC;
```

```
  servo_data.vert = 2750;
  servo_data.horz = 2650;

ervo_actuate();


********BEHAVIORS***************/
 light_loop(int loops, int length)

nt i, a;
or (i = 0; i < loops; i++)

  for (a = 0; a < length; a++)
  {
     sensor_light_follow();
  }



********BEHAVIORS***************/
 front_collision()

otor_data.lpw = -30;
otor_data.rpw = -30;
otor_actuate();
ight_loop(4,500);
f (servo_data.horz < 2750)

  motor_data.lpw = 30;
  motor_data.rpw = -30;

lse

  motor_data.lpw = -30;
  motor_data.rpw = 30;

otor_actuate();
ight_loop(3,250); /* 180 == light_loop(3,500); */
otor_data.lpw = 0;
otor_data.rpw = 0;
otor_actuate();


********BEHAVIORS***************/
 hermes_light_follow()


* KEEP LIGHT IN FRONT */
f (servo_data.vert > 2900)

  if (servo_data.horz < 2750)
  {
     motor_data.lpw = -20;
     motor_data.rpw = 20;
  }
  else
```

```
  tight_loop(3,175);
  motor_data.lpw = 0;
  motor_data.rpw = 0;
  motor_actuate();


Rbump_sense();

* BUMP AVOIDANCE */
f (IRbump_data.bump < 9)
/* No bump */

lse if (IRbump_data.bump < 14)
/* middle right */
  front_collision();

lse if (IRbump_data.bump < 18)
/* Front middle */
  front_collision();

lse if (IRbump_data.bump < 31)
  /* Front middle right */
  front_collision();

lse if (IRbump_data.bump < 47)
 /* Front middle left */
  front_collision();

lse if (IRbump_data.bump < 81)
 /* Front left */
  front_collision();

lse if (IRbump_data.bump < 135)
 /* Back */
/*  motor_data.lpw = 0;
    motor_data.rpw = 0;
    motor_actuate();
    delay(500,5000); */


 /* IR AVOIDANCE */
f (IRbump_data.fr_eye >= 100 &&
  IRbump_data.fl_eye >= 100)
 /* Both detect object */
  if (IRbump_data.l_eye > IRbump_data.r_eye + 5)
  {/* right side clearer turn right*/
     motor_data.lpw = 20;
     motor_data.rpw = -20;
  }
  else
  {/* left side clearer turn left */
     motor_data.lpw = -20;
     motor_data.rpw = 20;
  }
  motor_actuate();
  while(IRbump_data.fl_eye > 90 ||
```

```
lse if (IRbump_data.fr_eye > IRbump_data.fl_eye + IRDIFF)
/* front right eye detected object */
  if (motor_data.lpw > 0)
  { motor_data.lpw = motor_data.lpw - 5; }
  if (motor_data.rpw < 70)
  { motor_data.rpw = motor_data.rpw + 1; }

lse if (IRbump_data.fr_eye + IRDIFF < IRbump_data.fl_eye)
 /* front left eye detected object */
  if (motor_data.lpw < 70)
  { motor_data.lpw = motor_data.lpw + 1; }
  if (motor_data.rpw > 0)
  { motor_data.rpw = motor_data.rpw - 5; }


* STOPPING CONDITION */
lse if (servo_data.vert > 2700 && servo_data.vert < 2800)
 /* stop */
  if (motor_data.lpw > 0)
  { motor_data.lpw = motor_data.lpw - 1; }
  if (motor_data.rpw > 0)
  { motor_data.rpw = motor_data.rpw - 1; }


* CHASE LIGHT */
lse if (servo_data.horz > 2700 && IRbump_data.l_eye < 90)
  /* turn left */
  if (servo_data.horz >= 4000)
  {
     motor_data.lpw = 0;
     motor_data.rpw = 60;
  }
  else if (servo_data.horz > 3600)
  {
     motor_data.lpw = 5;
     motor_data.rpw = 60;
  }
  else if (servo_data.horz > 3300)
  {
     motor_data.lpw = 10;
     motor_data.rpw = 60;
  }
  else if (servo_data.horz > 3000)
  {
     motor_data.lpw = 20;
     motor_data.rpw = 60;
  }
  else
  {
     motor_data.lpw = 30;
     motor_data.rpw = 60;
  }

lse if (servo_data.horz < 2600 && IRbump_data.r_eye < 90)
 /* turn right */
  if (servo_data.horz < 1400 )
```

```
  else if (servo_data.horz < 2000)
  {
     motor_data.lpw = 60;
     motor_data.rpw = 10;
  }
  else if (servo_data.horz < 2300)
  {
     motor_data.lpw = 60;
     motor_data.rpw = 20;
  }
  else
  {
     motor_data.lpw = 60;
     motor_data.rpw = 30;
  }

/* STOPING CONDITION */
lse

  motor_data.lpw = 98;
  motor_data.rpw = 98;


otor_actuate();


********BEHAVIORS***************/
distancefromlight()

ouble angle = (servo_data.vert - 1000) / 19.44444;
f (angle == 90)

  return 0; }
lse

  return LIGHTHEIGHT/tan(angle); }


***MAIN*************/
 main()

 HARDWARE INITIALIZATIONS */
nit_serial();
rite("Initialized Serial\n\r");

nit_analog();
rite("Initialized Analog\n\r");

nit_motors();
rite("Initialized Motors\n\r");

nit_servos();
rite("Initialized Servos\n\r");

 VARIABLE INITIALIZATIONS */
Rbump_data.fl_eye = IRbump_data.fr_eye = IRbump_data.l_eye = 0;
```

```
otor_data.rpw = 0;
otor_data.lpw = 0;

Rhigh = 115;
Rlow = analog(1);
cale = IRhigh - IRlow;
RLATCH = 0xFF;  /* turn on IR */

hile (1)

  sensor_light_follow();
  hermes_light_follow();
```

```
.C
t Leary
/99
ass Code

********BEHAVIORS***************/
 turn(int degrees, int direction)

nt templpw = motor_data.lpw;
nt temprpw = motor_data.rpw;
nt desiredheading;
nt curheading = compass_data.heading;
umout(curheading);
rite(" Current Heading\r\n");
f (direction == LEFT)

  if ((curheading - degrees) < 0)
  {desiredheading = 360 - degrees;}
  else
  {desiredheading = (curheading - degrees);}

  motor_data.lpw = -7;
  motor_data.rpw = 7;
  motor_actuate();
  while (compass_data.heading != desiredheading - 1 &&
        compass_data.heading != desiredheading     &&
        compass_data.heading != desiredheading + 1)
  {
     compass_sense();
     write(" LEFT desired heading "); numout(desiredheading);
     write(" Current Heading "); numout(compass_data.heading);
     write(" LPW "); numout(motor_data.lpw);
     write(" RPW "); numout(motor_data.rpw);
     write("\r");
  }

lse

  desiredheading = (curheading + degrees) % 360;
  motor_data.lpw = 7;
  motor_data.rpw = -7;
  motor_actuate();
  while (compass_data.heading != desiredheading - 1 &&
        compass_data.heading != desiredheading &&
        compass_data.heading != desiredheading + 1)
  {
     compass_sense();
     write(" RIGHT desired heading "); numout(desiredheading);
     write(" Current Heading "); numout(compass_data.heading);
     write(" LPW "); numout(motor_data.lpw);
     write(" RPW "); numout(motor_data.rpw);
     write("\r");
  }
```

```
itializes SPI system as 68HC11 as SLAVE
 Bits 7     6     5      4     3     2
CR = int   sys   wired  M/S   CPOL  CHPA
     on    on     or    1 0   0 Rs   1
      0     1     0      0     0     1   0x44
                               1     0   0x48
                               0     0   0x40
RD Setting automatic when hc11 is a slave


PCR = 0x48;  */  /* works with BCD */
PCR = 0x40;      /* works with Binary */


******INIT COMPASS***************/
 init_compass()

*            8      7       6     5      4  3  2  1
  OUTLATCH *RES *BCD/BIN  *P/C  *RESET      C  B  A */

* Reset compass 5V = *SS-tied high, *CAL-tied high, *RESET, P/C */
LEAR_BIT(LATCHSTATE,0xF0);
ET_BIT(LATCHSTATE,0x70);
UTLATCH = LATCHSTATE;

*  10 msec  */
elay(5,5000);

* 0Vs *reset */
LEAR_BIT(LATCHSTATE,0x10);
UTLATCH = LATCHSTATE;

* 10 msec */
elay(5,5000);

* 5v *reset */
ET_BIT(LATCHSTATE,0x10);
UTLATCH = LATCHSTATE;

* 500 msec */
elay(10,5000);

* P/C 0Vs contious output */
LEAR_BIT(LATCHSTATE,0x20)
  OUTLATCH = LATCHSTATE;


*****UPDATE SENSORS************/
 compass_sense()

* GET READING FROM COMPASS */
nt test1 = 0;
nt test2 = 0;
ompass_data.headingpp = compass_data.headingp;
ompass_data.headingp = compass_data.heading;
```

```
/* calculate heading */
f (compass_data.reading2 == 1 && compass_data.reading1 < 104)

  compass_data.heading = 256 + compass_data.reading1;
  compass_data.error_count = 0;

lse if (compass_data.reading2 == 0)

  compass_data.heading = compass_data.reading1;
  compass_data.error_count = 0;

lse

  compass_data.error_count += 1;


* Reset compass if to many errors */
f (compass_data.error_count > 100)

  compass_data.error_count = 0;
  init_compass();



*****TERMINAL OUTPUT**********/
 compass_output()

ompass_sense();
umout(compass_data.heading);
rite("  ");
umout(compass_data.headingp);
rite("  ");
umout(compass_data.headingpp);
rite("  ");
umout(compass_data.error_count);
rite("  ");



ass Sensor Data */
ct compass

nt reading1;
nt reading2;
nt heading;
nt headingp;
nt headingpp;
nt error_count;
pass_data;
```

LUDE FILE

C45.H

```
rupt Control */
 INTR_ON() asm(" cli")
 INTR_OFF()      asm(" sei")
 bit(x)    (1 << (x))

bits */
 RDRF      bit(5)
 TDRE      bit(7)
 T8  bit(6)
 R8  bit(7)

bits */
 MSTR      bit(4)
 SPE bit(6)
 SPIF      bit(7)

OM */
 EEPGM     bit(0)
 EELAT     bit(1)

control */
 CLEAR_BIT(x,y) x &= ~y;
 SET_BIT(x,y) x |= y;
 CLEAR_FLAG(x,y) x &= y;

sters */
 _IO_BASE  0x1000
 PORTA     *(unsigned char volatile *)(_IO_BASE + 0x00)
 PIOC      *(unsigned char volatile *)(_IO_BASE + 0x02)
 PORTC     *(unsigned char volatile *)(_IO_BASE + 0x03)
 PORTB     *(unsigned char volatile *)(_IO_BASE + 0x04)
 PORTCL    *(unsigned char volatile *)(_IO_BASE + 0x05)
 DDRC      *(unsigned char volatile *)(_IO_BASE + 0x07)
 PORTD     *(unsigned char volatile *)(_IO_BASE + 0x08)
 DDRD      *(unsigned char volatile *)(_IO_BASE + 0x09)
 PORTE     *(unsigned char volatile *)(_IO_BASE + 0x0A)
 CFORC     *(unsigned char volatile *)(_IO_BASE + 0x0B)
 OC1M      *(unsigned char volatile *)(_IO_BASE + 0x0C)
 OC1D      *(unsigned char volatile *)(_IO_BASE + 0x0D)
 TCNT      *(unsigned short volatile *)(_IO_BASE + 0x0E)
 TIC1      *(unsigned short volatile *)(_IO_BASE + 0x10)
 TIC2      *(unsigned short volatile *)(_IO_BASE + 0x12)
 TIC3      *(unsigned short volatile *)(_IO_BASE + 0x14)
 TOC1      *(unsigned short volatile *)(_IO_BASE + 0x16)
 TOC2      *(unsigned short volatile *)(_IO_BASE + 0x18)
 TOC3      *(unsigned short volatile *)(_IO_BASE + 0x1A)
 TOC4      *(unsigned short volatile *)(_IO_BASE + 0x1C)
 TOC5      *(unsigned short volatile *)(_IO_BASE + 0x1E)
 TCTL1     *(unsigned char volatile *)(_IO_BASE + 0x20)
 TCTL2     *(unsigned char volatile *)(_IO_BASE + 0x21)
 TMSK1     *(unsigned char volatile *)(_IO_BASE + 0x22)
 TFLG1     *(unsigned char volatile *)(_IO_BASE + 0x23)
```

```
 SCCR2     *(unsigned char volatile *)(_IO_BASE + 0x2D)
 SCSR      *(unsigned char volatile *)(_IO_BASE + 0x2E)
 SCDR      *(unsigned char volatile *)(_IO_BASE + 0x2F)
 ADCTL     *(unsigned char volatile *)(_IO_BASE + 0x30)
 ADR1      *(unsigned char volatile *)(_IO_BASE + 0x31)
 ADR2      *(unsigned char volatile *)(_IO_BASE + 0x32)
 ADR3      *(unsigned char volatile *)(_IO_BASE + 0x33)
 ADR4      *(unsigned char volatile *)(_IO_BASE + 0x34)
 OPTION    *(unsigned char volatile *)(_IO_BASE + 0x39)
 COPRST    *(unsigned char volatile *)(_IO_BASE + 0x3A)
 PPROG     *(unsigned char volatile *)(_IO_BASE + 0x3B)
 HPRIO     *(unsigned char volatile *)(_IO_BASE + 0x3C)
 INIT      *(unsigned char volatile *)(_IO_BASE + 0x3D)
 TEST1     *(unsigned char volatile *)(_IO_BASE + 0x3E)
 CONFIG    *(unsigned char volatile *)(_IO_BASE + 0x3F)

 PERIODM 65,500          /* Pulse period for motors */
 PERIOD 40000            /* Pulse period for servos */
 PERIOD_1PC 655

gned int read_sci();
gned char read_spi();
 write_eeprom(unsigned char *addr, unsigned char c);
 write_sci(unsigned int);
 write_spi(unsigned char);

def enum {
9600 = 0x30, BAUD4800 = 0x31, BAUD2400 = 0x32,
1200 = 0x33, BAUD600 = 0x34, BAUD300 = 0x35
udRate;
 setbaud(BaudRate);


rn void motor0(), motor1(), servo_OC4(), servo_OC5();
rn void _start(); /* entry point in crt11.s */

 abs_address:0xffd6
 (*interrupt_vectors[])() =

/* SCI */
/* SPI */
/* PAIE */
/* PAO */
/* TOF */
o_OC5, /* TOC5 */
o_OC4, /* TOC4 */
r1,    /* TOC3 */
r0,    /* TOC2 */
/* TOC1 */
/* TIC3 */
/* TIC2 */
/* TIC1 */
/* RTI */
/* IRQ */
/* XIRQ */
/* SWI */
```

```
duty_cycle[2];  /* Specifies the PWM duty cycle for two motors */
gned width[2];  /* Holds PWM value for the two servos */


*******MOTORS**********************/
 init_motors(void)
unction: This routine initializes the motors
puts:   None
tputs:  None
tes:    This routine MUST be called to enable motor operation!



NTR_OFF();

ake PORTD pins 4 and 5 output pins */
ET_BIT(DDRD,0x30);

et all OCx pins to output low */
ET_BIT(TCTL1, 0xAA);
LEAR_BIT(TCTL1, 0x55);

et PWM duty cycle to 0 first */
uty_cycle[0] = duty_cycle[1] = 0;

nable motor interrupts on OC2 and OC3 */
ET_BIT(TMSK1, 0x60);

NTR_ON();


*******MOTORS**********************/
 motor(int index, int per_cent_duty_cycle)
unction: Sets duty cycle and direction of motor specified by index
puts:   index in [0,1]
        -100% <= per_cent_duty_cycle <= 100%
        A negative % reverses the motor direction
tputs:  duty_cycle[index]
        0 <= duty_cycle[index]<= PERIOD (Typically, PERIOD = 65,500)
tes:    Checks for proper input bounds


f (per_cent_duty_cycle < 0)

  per_cent_duty_cycle = -per_cent_duty_cycle;
* Set negative direction of motors */
  if (index == 0) CLEAR_BIT(PORTD, 0x10);
  if (index == 1) SET_BIT(PORTD, 0x20);

lse

* Set positive direction of motors */
  if (index == 0) SET_BIT(PORTD, 0x10);
  if (index == 1) CLEAR_BIT(PORTD, 0x20);
```

```
*******MOTORS**********************/
 motor0()
unction: This interrupt routine controls
          the PWM to motor0 using OC2
nputs:   duty_cycle[0] (global)
utputs:  Side effects on TCTL1, TOC2, TFLG1.
otes:    init_motors() assumed to have executed



eep the motor off if no duty cycle specified.*/

f(duty_cycle[0] == 0)

  CLEAR_BIT(TCTL1, 0x40);

lse
  if(TCTL1 & 0x40)
  {
     TOC2 += duty_cycle[0];           /* Keep up for width */
     CLEAR_BIT(TCTL1, 0x40);          /*  Set to turn off */
  }
  else
  {
     TOC2 += (PERIODM - duty_cycle[0]);
     SET_BIT(TCTL1, 0x40);            /* Set to raise signal */
  }
LEAR_FLAG(TFLG1, 0x40);               /* Clear OC2F interrupt Flag */



*******MOTORS**********************/
 motor1()
unction: This interrupt routine controls the PWM to motor1 using OC3
puts:   duty_cycle[1] (global)
tputs:  Side effects on TCTL1, TOC2, TFLG1.
tes:    init_motors() assumed to have executed



eep the motor off if no duty cycle specified.*/

f(duty_cycle[1] == 0)

  CLEAR_BIT(TCTL1, 0x10);

lse
  if(TCTL1 & 0x10)
  {
     TOC3 += duty_cycle[1];         /* Keep up for width */
     CLEAR_BIT(TCTL1, 0x10);        /*  Set to turn off */
  }
  else
  {
     TOC3 += (PERIODM - duty_cycle[1]);
```

```
NTR_OFF();
LEAR_BIT(PACTL,0x04);       /* Set IC4/OC5 to OC5 */
ET_BIT(TCTL1,0x0A);         /* Set OC4 & OC5 to 0v */
LEAR_BIT(TCTL1,0x05);

idth[0] = width[1] =0;      /* Set PWM's to 0 first */
ET_BIT(TMSK1,0x18);         /* Enable interupts */
NTR_ON();


*********SERVOS*******************/
 servo(int index, unsigned newwidth)
ets a servo to a certain pulse width */

idth[index] = newwidth;


*********SERVOS*******************/
 servo_OC4()

f(width[0] == 0)

  CLEAR_BIT(TCTL1, 0x04);

lse if(TCTL1 & 0x04)

  TOC4 += width[0];                /* Keep up for width */
  CLEAR_BIT(TCTL1,0x04);           /*  Set to turn off */

lse

  TOC4 += (PERIOD - width[0]);
  SET_BIT(TCTL1,0x04);             /* Set to raise signal */

LEAR_FLAG(TFLG1,0x10);             /* Turn off OC4 interrupt */



*********SERVOS*******************/
 servo_OC5()

f(width[1] == 0)

  CLEAR_BIT(TCTL1, 0x01);

lse if(TCTL1 & 0x01)

  TOC5 += width[1];              /* Keep up for width */
  CLEAR_BIT(TCTL1,0x01);         /*  Set to turn off */

lse

  TOC5 += (PERIOD - width[1]);
  SET_BIT(TCTL1,0x01);           /*   Set to raise signal */
```

```
*********ANALOG******************/
s analog port */
analog(int port)
akes one reading from the analog port
nd returns the value read
eturn Value : Value read from A/D port


DCTL=port;
hile((ADCTL & 0x80) != 0x80);
eturn(ADR1);


*********SERIAL******************/
ializes SCI port 9600 baud */
 init_serial()

LEAR_BIT(SPCR,0x20);
AUD = 0xb0;  /* 0xbO is 9600 0x35 is 300 baud */
CCR2 = 0x0C;


*********SERIAL******************/
lays character */
 put_char(int outchar)

nt test = 0;

hile (test == 0)

  test = SCSR & 0x80;


CDR = outchar;


*********SERIAL******************/
ts(unsigned char *s)

hile (*s)

  put_char(*s);
  s++;



*********SERIAL******************/
lays number */
mout(unsigned int n)

nt   i;
nsigned int k;
nsigned char digits[7];
```

```
*********SERIAL*****************/
lays character string */
 write(char strng[80])

nt index = 0;
nt a=0;

hile(strng[index] != 0) {
  if ((strng[index] == 0x5c) && (strng[index+1] == 0x6e))
  {
     put_char(10);
     put_char(13);

     index += 2;
  }
  else {
     put_char(strng[index]);
     index++;
  }



*********WASTE TIME**************/
id delay (int loops, int length)

nt i, a;
or (i = 0; i < loops; i++)

  for (a = 0; a < length; a++);
```