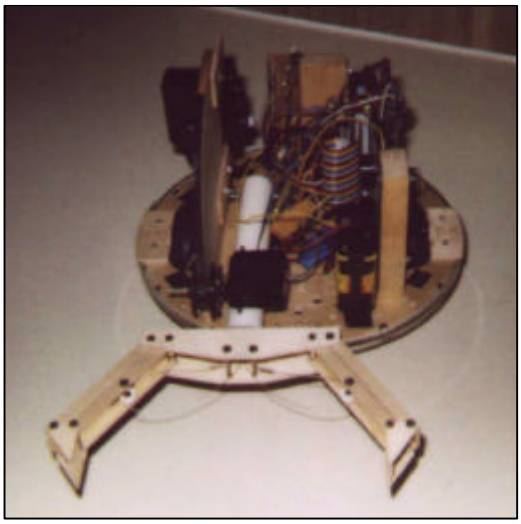
SISYPHUS



Kenneth Wiant
4EG – CEE
EEL 5666 spring 1998
Professor: Dr. Arroyo

Acknowledgements

I would like to thank Chris Beattie who talked me into taking this course and for helping me design the claw. I would also like to thank Taco Bell for donating parts for the arm design. I would like to thank Energizer for donating 10 Nickel Metal Hydride batteries to the cause. I want to thank The Independent Florida Alligator for sponsoring my project and for making Sisyphus the official robot of The Independent Florida Alligator. Lastly I want to thank all of my friends and support and for not abandoning while I've labored these past few months.

Table of Contents

Abstract

Sisyphus is an autonomous mobile robot design to compete in the IMDL contest for Spring 1998. It is capable of moving around an arena and locating objects, grabbing and lifting objects and delivering them into bins. The robot consists of a Motorola 68HC11, an ME11 and MTSX01 controller boards, two wheels to move around and an arm and a claw to manipulate objects. The sensor suite provides data to the robot by means of infra red sensors and limit switches.

Introduction

This robot was intended to compete in the IMDL contest for the Spring of 1998. It was supposed to be able to find objects, and place them into bins according to what the object was. I was able to achieve most of this, but it proved to be a long and difficult process of trial and error. My main problems were to find the object, determine what it is, find the bins, and determine which bin I was at and which I needed to be at and then deliver the object. I spent most of my time trying to find ways of grasping and lifting the robot. As a result my actuators were well developed, but my sensors could have used some more attention.

This paper will describe not only the final robot, but also a lot of the steps I took to reach this point. Notably, I will discuss the operation of the robot, what controls it, how it interacts with the arena and objects and how it senses its environment.

Integrated System

Sisyphus is an autonomous mobile robot that uses an arm to grab and lift objects and move to an appropriate bin and drop the object in.

To control the robot, I used a Motorola 68HC11 EVBU (the same one I used in EEL 4744 – Microprocessor Applications), the ME11 memory expansion board, and the MTSX01 sensor board.

The ME11 expansion board provides the microcontroller with 32 kilobytes of RAM as well as two motor drivers, a voltage regulator, and an oscillating output port for infrared LED's.

The MTSX01 was made to work with the 6811 EVBU and the ME11. This sensor board is made to allow up to 13 IR sensors, six cadmium sulfide photocell sensors, two shaft encoders, a battery charge sensor, and two bumper ports. There are also ports providing a connection to the data bus and memory mapped select lines for port expansion. The main problem with the MTSX01 is that there are no more schematics available. I was able to locate assembly instructions, but nothing more. Whatever specifications I was able to discover on my own, I have included in the appendix.

I used Interactive C to write the code for Sisyphus. I have an extensive C++ background so I should probably have written the code in ICC11, but I was able to get IC right away and start writing code very early.

I used IR and limit switches for sensors during the contest. I also built a joystick controller and a battery gauge for testing (and for letting other people play with it).

The 68HC11 gathered data from the sensors and responding by moving around the arena (using two wheels driven by the ME11 motor ports). If it encounters an object it uses a servo to close the claw on the object and then drives another motor (with it's own driver) to lift the object.

In addition to the sensors, controller, and actuators, I use two 8-packs of AA rechargeable batteries. The set used to power the processor and wheel motors were Nickel Metal Hydrides donated generously by Energizer, the other battery pack was Nickel Cadmium's, bought from Novasoft.

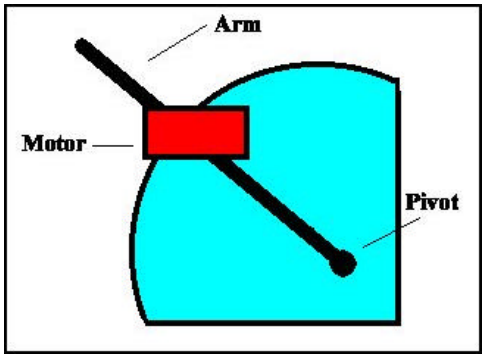
Mobile Platform

For the body of the robot, I used parts from the Talrik wood kit. All of the wood is five-ply model airplane wood. I used wood glue, super glue, and hot glue to assemble the body.

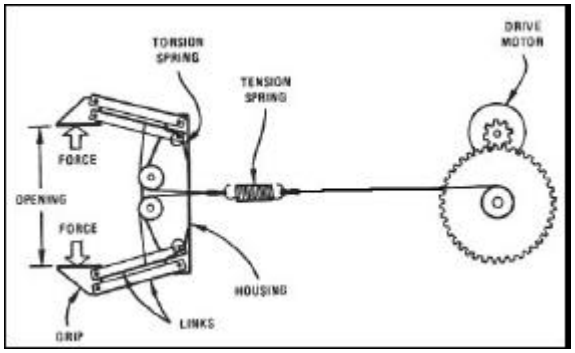
To move the robot around, I used two 2 servo hacked into DC motors. The ME11 expansion board's two motor ports drive these two motors.

The contest rules state that the entire robot must fit with an 18-inch cube and weigh no more than 20 pounds. The Talrik body is 10 inches in diameter, which give me room to add the claw and arm. I liked the round body, because it meant I could easily navigate the arena. Sisyphus could spin in place without fear of collision (as long as the arm was in the up position).

Actuation



The arm is controlled by one motor, which is attached to a toothed arc by means of a matching gear. The reason for this is due to early tests with motors, which failed. I tried to raise and lower the arm at the pivot, but the force the motor had to overcome was the weight of the object being lifted multiplied by the distance from the motor to the object. By moving the motor closer to the object, I was able to eliminate a large amount of the moment arm and get more yield from the motor. In the end, I was able to lift all of the objects with little difficulty using an old hacked motor bought used from Novasoft. In fact I ran the final contest using the same old motor. I used two limit switches to tell the CPU when the arm was in the up or down position.



The claw was based on a design by a previous student in IMDL, Chris Beattie, who based his idea on the MiniMover 5. In order for the design to work, the claw had to be much wider. I wanted the claw to be as wide as the robot itself so that I would be guaranteed of locating any object in its path. With Beattie's permission and help we redesigned the claw to be able to open 7.5 inches from inside fingertip to fingertip.

The drive motor pulls the cable (in this case, nylon coated steel fishing cable), when the claw can close no more, the drive motor stretches the tension spring, therefore I would be able to tell the motor to pull a certain amount, regardless of the size of the object. When the cable was released, the torsion springs forces the fingers open.

The claw was made from the same plywood as the rest of the body, as was the arc. The arm was one-inch thick PVC pipe. I used the pipe so that I could run the cable and spring inside the pipe (cleaner looking, and no chance of accidentally getting caught in the mess of wires and causing all sorts of damage).

To operate the motor, I purchased a special servo motor, rated at 333-ounce inches. The servo was twice as big as the other motors. Attached to the motor was a small arm that pulled the cable from the claw. I stripped some of the gears of this motor and had to purchase a new one, but this time I hacked the servo to operate just like a servo, except that the potentiometer was glued in place, so that the servo always thought it was in the center position (the TJ hack). Attached to this motor was a small spool (once again, smaller moment means less force to overcome to lift the same weight.) This created a new problem. The servo was so strong, that even when power was cut from the motor, it had enough strength to hold the cable in its position. I needed to add a limit switch to let the CPU know when the claw was fully open. To close the claw, I just timed how long it took to close the claw all the way when the batteries were fully charged. And then tried it again when the batteries were low. There was little difference so I just told the claw to close a certain amount and the tension spring took up the slack.

The inside of the claw was lined with foam padding used underneath carpets. This material was suggested to me by other IMDLers and worked wonderfully. It was soft enough to give when squeezed and had enough friction to hold up all the objects, plus I was able to cut it easily to fit the claw.

Sensors

The sensor suite of Sisyphus needed to be able to follow a wall until it encountered an object, determine what the object was, and find the bin it belongs in. I was able to use IR and bump switches to accomplish most of this. I tried to find a force transducer to sort the objects by weight, but these were prohibitively expensive, but would have worked perfectly.

The limit switches were plugged into the CDS ports on the MTSX01. These provide a power and a signal pin. When the switch was closed, the port read 255 otherwise it read 0.

There were 2 IR sensors mounted on the right side of the robot on the base. These were used to follow the wall. There was another on the rear of the robot to detect collision. The Claw had two IR sensors, one to detect straight ahead, and one break beam sensor to detect when an object was in the claw. There was also one more IR on the right side, but higher up than the other two.

There were two bumpers (front and rear) that ran around the entire robot.

The two right IR sensors tried to keep the robot a certain distance away from the wall while the front and rear IR sensors would tell the robot to turn if it got too close to a wall. The third IR on the right side was to tell which bin the robot was at. If the top sensor was getting a reading and the two bottom ones were too, then the robot was in the arena away

from the bins. If the top sensor was not getting a reading, and the bottom two were, then the robot was at one of the lower bins. If the top sensor was getting a reading and the bottom two were not, then the robot was at the middle bin.

Extra Sensors

In addition to sensors that were used in the contest, I built sensors for use in testing. These were a visible battery gauge and a joystick port.

The battery gauge was mostly already done for me. The MTSX01 had a voltage divider in place through the battery current. This signal was multiplexed into the analog ports on the 68HC11. I merely took this data and sent it to another output port that I had placed an 8-bit latch on and connected the output to a bank of 8 LED's. The data was first converted from an 8-bit number to a scale from one to eight lights. The more voltage the more lights. While the robot was running the level of the battery was visible by how high the bank of LED's was lit. I tested the system until I found the voltage the microprocessor would reset at and used this as the zero.

The joystick port was designed for use with an old ATARI joystick. The joystick has four directions (up, down, left, right) as well as a button for a total of 5 inputs. Originally I used them as five digital inputs (in the same way as the limit switches). But I did not have enough room to use all five as well as the three limit switches (there are only 6 CDS ports). I decided to build a set of voltage dividers and wire them together so that the highest voltage would be small enough to work with the analog ports of the 68HC11. My first design worked, not perfectly, but well enough. I plugged this input into one IR port instead of 5 CDS ports and was able to determine the direction (or button press) by means of software decoding.

Behaviors

The behaviors of Sisyphus could be sorted into to main categories, finding an object and depositing an object.

To find an object, the robot follows a wall on its right side until something breaks the beam in its claw. Then, it stops, closes the claw and raises the arm. As soon as this happens, the robot plays a short tune on the piezo speaker to the let me know it is in deposit mode.

In deposit mode, it follows the wall on the same side as before, but this time it travels backwards. Once it detects it is at the correct bin, it backs up at an angle and drives forward until it bumps the bin. The bumper is able to tell where the bump came from, so the robot will back up and turn and bump again, until it hits head on. Once it does so, it opens the claw, then backs up. Once it has backed up a set distance, it lowers its arm turns away from the bins and starts to find an object again.

The only problems I had were with sensor calibration. With another week of testing I could have had the robot working perfectly. Sometimes it will follow a little too close to the wall, sometimes a little too far away, or sometimes it will overshoot the bin. This problem is mostly due to a problem with lining up with the bin.

When the robot looks for the bins, it can spot them right away, often too early. When it would do this it would find the bin right at its corner. When the robot tried to bump the

bin, it would hit the corner and it would not spread the bump enough to trigger a switch, or it would simply miss since it was so close to the edge. To fix this I added a delay in the code that would make the robot travel a set distance until it would turn. Sometimes it would take too long.

I could have fixed most of these problems with more time to calibrate the sensors, and also if I had found a way to integrate the battery gauge with the sensor offsets.

Results

I took first place in the IMDL contest. I won with 13 points in the first round. My behaviors worked well, as well as my actuation. My main problems were with the sensors malfunctioning or just not set to the right thresholds.

Conclusion

I accomplished nearly all of my goals. If I had come up with the idea for the arm sooner as well as tried harder to get the force transducer, and calibrate my sensors better, I would have succeeded.

I did not feel that Sisyphus was intelligent enough for the contest. It was obvious that the sensors were not responding the way I would have liked. I built a battery gauge but failed to use it in my sensor offsets. The robot also had no way of telling which object was which.

However, I was able to grab lift and dump all of the objects in bins. All of my actuation worked. My behaviors worked well too. The robot performed like it knew what it was doing, but that it did not see everything it needed to.

I was told very early that using a claw and arm to compete in the IMDL competition would be too difficult, but I was able to do it reliably. I was able to find a better way to build a stronger arm and claw. I feel that although Sisyphus did not perform perfectly, it did perform intelligently and independently.

If I could do this project over, I would use the claw and arm design I ended up with. I spent a lot of time trying new claw and arm techniques and testing motors, and making static's calculations. I also would have tried harder to get a force transducer to weigh the

objects while they were being held up. I would also spend more time tweaking sensor values to get the robot to respond correctly to the environment.

Overall I am pleased with Sisyphus's performance at the IMDL contest. It felt as though he was doing all the work and I was just the coach in the corner of the ring saying "remember what I taught you, don't let me down."

MTSX01 memory mapping

IRDT ports 2 – 7 correspond to analog(2) through analog(7) respectively.
The rest of the sensors are multiplexed. First you have to poke an address to enable the correct signal.

Poke(0x4000, 0x00)	-	IRDT8
Poke(0x4000, 0x01)	-	IRDT9
Poke(0x4000, 0x02)	-	IRDT10
Poke(0x4000, 0x03)	-	IRDT11
Poke(0x4000, 0x04)	-	IRDT12
Poke(0x4000, 0x05)	-	IRDT13
Poke(0x4000, 0x06)	-	IRDT14
Poke(0x4000, 0x07)	-	CDS1

Then you have to check analog(0) to get the value.

Poke(0x4000, 0x00)	-	CDS2
Poke(0x4000, 0x08)	-	CDS3
Poke(0x4000, 0x10)	-	CDS4
Poke(0x4000, 0x18)	-	CDS5
Poke(0x4000, 0x20)	-	CDS6
Poke(0x4000, 0x28)	-	rear_bumper
Poke(0x4000, 0x30)	-	front_bumper
Poke(0x4000, 0x38)	-	battery_voltage

Then you have to check analog(1) to get the value.

The IOPORT port has 2 digital output pins , which correspond to

Poke(0x4000, 0x40)	-	pin 1
Poke(0x4000, 0x80)	-	pin 2

You can also think of this as 2 mux's that are enabled by adress 4000.
The data bus works as follows:
Bits 6-7 go to IOPORT digital output pins
Bits 3-5 go to select lines for chip U3
Bits 0-2 go to select lines for chip U2

IMDL Contest Rules - Spring 1998

Not reprinted here, go to www.mil.ufl.edu for more detials.

```
/* Sisyphus.c by Kenneth Wiant */
/* Spring 1998 – IMDL */
/* requires servo.icb, servo.c, serial.c, music.c, lib_rw10.c */
/* don't use lib_rw11.c */

/* GLOBALS */
/*Variables*/
int DIR = 0;            /* Arm position */
int IR2, IR3;           /* wall following */
int IR4, IR5;           /* Front / Break beam */
int IR7;                /* rear */
int IR8, IR9, IR10, IR13, IR14;
int IR11, IR12;         /* top , right claw */
int front, rear;        /* bumper values */
int BAT;                /* Battery Level */
int offset = 100;
int offset2 = 120;       /* IR offsets */
int offset3 = 120;
int offset4 = 100;
int offset5 = 120;
int offset13 = 0;
int ML = 0;             /* initial motor values */
int MR = 0;
int IML = 0;            /* initial input motor values */
int IMR = 0;
int OML = 0;
int OMR = 0;
int PID,PID1,PID2,PID3;           /* for multitasking */
int K = 6;              /* motor adjustment level */
int LS1,LS2,LS3;        /* limit switches */
int joy=0;
int DELAY = 0;          /* for centering at bin */
int x,y;                /* for debugging */

int irdt(int detector) {
if ((detector >= 2) && (detector <= 7)) return(analog(detector));
else if (detector == 8) { poke(0x4000,0x00); return(analog(0)); }
else if (detector == 9) { poke(0x4000,0x01); return(analog(0)); }
else if (detector ==10) { poke(0x4000,0x02); return(analog(0)); }
else if (detector ==11) { poke(0x4000,0x03); return(analog(0)); }
else if (detector ==12) { poke(0x4000,0x04); return(analog(0)); }
else if (detector ==13) { poke(0x4000,0x05); return(analog(0)); }
else if (detector ==14) { poke(0x4000,0x06); return(analog(0)); }
}

int cds(int detector) {
  if (detector == 1) { poke(0x4000,0x07); return(analog(0)); }
  else if (detector == 2) { poke(0x4000,0x00); return(analog(1)); }
  else if (detector == 3) { poke(0x4000,0x08); return(analog(1)); }
  else if (detector == 4) { poke(0x4000,0x10); return(analog(1)); }
  else if (detector == 5) { poke(0x4000,0x18); return(analog(1)); }
  else if (detector == 6) { poke(0x4000,0x20); return(analog(1)); }
}

int abs (int value)
{
```

```
        if (value < 0) return(-value);
        else return(value);
}

void read_sensors()
{
      while (1)
      {
      hog_processor();
      LS1 = cds(3);
      LS2 = cds(6);
      LS3 = cds(2);
      poke(0x7000,0xff);              /* turn on IR emmitters */
      wait(30);                       /* wait a bit */
      IR7 = irdt(7) - offset;
      IR2 = irdt(2) - offset2;
      IR3 = irdt(3) - offset3;
        IR4 = irdt(4) - offset4;
      IR5 = irdt(5) - offset5;
      IR8 = irdt(8);
      IR9 = irdt(9);
      IR10 - irdt(10);
      IR11 = irdt(11) - offset;
        IR12 = irdt(12);
        IR13 = irdt(13) - offset13;
      IR14 = irdt(14);
      poke(0x7000,0x00);              /* turn off IR emmitters */
      poke(0x4000,0x38);
        BAT = analog(1);
        battery();
      joy = irdt(6);                  /* Joystick position */
      poke(0x4000,0x30);
      front = analog(1);
      poke(0x4000,0x28);
      rear = analog(1);
      defer();
      }
}

void follow()
{
      while (1)
      {
        if (DIR != -1)   DELAY = 0;
      else
        {
        if (IR11 < 0 && IR3 > -2 && IR2 > -2 && DELAY > 4)
                {

                ML = 0; MR = -40; wait(1100); MR = 0;
                while (front < 50 || front > 100)
                        {
                        if (front > 1 && front < 50)
                                {ML = -45; MR = -20; wait(700);}
                        else if (front > 100)
                                {ML = -20; MR = -50; wait(700);}
                        else { ML = 35; MR = 35; }
```

```
                        }
                DIR = 0; drop_arm();
            }
        if (IR11 < 0 && IR3 > 0 && IR2 > 0 && DELAY <= 4) DELAY++;
        if (rear >5) { MR = 35; ML = 35; wait(700); MR = 0; wait(500);}
        else if (IR7 > 5) { MR = -35; ML = 30; }
        else if (IR7 > 0) { MR = -35; ML = 0;}
        else if (IR2 >= 0 && IR3 >= 0)
                {
                if (IR3 >= IR2) { ML = -35; MR = -45;}
                else { ML = -38+(IR2-IR3)*2; MR = -55;}
                }
        else if (IR3 >= 0 && IR2 < 0)
                { ML = -33; MR = 0;}
        else if (IR3 < 0 && IR2 < 0) { ML = -40; MR = -25;}
        else { ML=-50; MR=-40; }
      defer();
        }
      }
}

void seek()
{
        wait(100);
        IR5 = irdt(5);
      while (1)
      {
      if (DIR != 1) defer();
        else
        {
        if (IR5 < 0) {  ML = -25; MR =-25; DIR = 0; lift_arm();}
        else if (IR4 > 4) { MR = 40; ML = -40;}
        else if (IR4 > -2) { MR = 40; ML = 0;}
        else if (IR2 > 1 && IR3 > 1) { MR = 40+(IR3-IR2)*1; ML = 40;}
        else if (IR2 < 1 && IR3 > 1) { ML = 40+(IR2-IR3)*1; MR = 40;}
        else if (IR2 < 0 && IR3 < 0) { ML = 40; MR = 30;}
      else { ML=50; MR=50; }
      defer();
        }
      }
}


void motor_update()
{
      while(1)
      {
            OML = (ML + (K * OML)) / (K+1);
            OMR = (MR + (K * OMR)) / (K+1);
            motor(0,OML);
            motor(1,OMR);
            wait(10);
      }
}

void alert_tune()   /* Tune that plays at startup */
{
```

```
  poke(0x5000,0x10);
  tone(1046.5, 0.3);
  poke(0x5000,0x80);
  tone(1396.9, 0.3);
  poke(0x5000,0x10);
  tone(1046.5, 0.3);
  poke(0x5000,0x01);
  tone(526.9, 0.3);
  poke(0x5000,0x08);
  tone(746.5, 0.3);
}

void wait(int m_second)
      {
      long stop_time;
      stop_time = mseconds() + (long)m_second;
      while(stop_time > mseconds())
            defer();
      }

void close()
{
        wait (100); beep(); wait(100); beep(); wait(100);
        servo_deg(80.0);
      servo_on();
        wait(2500);
      servo_off();
        charge();
}
void open()
{
        wait (100); beep(); wait(100); beep(); wait(100);
      servo_deg(95.0);
      servo_on();
        poke(0x4000,0x00);
        LS3 = analog(1);
        while (LS3 < 100)
                {
                poke(0x4000,0x00);
                LS3 = analog(1);
                wait(5);
                }
        servo_off();
      wait(10);
}

void drop_arm()
{
      kill_process(PID);      /* kill motor_update */
        kill_process(PID1);     /* kill read_sensor */
        motor(0,0); motor(1,0);
        charge(); DIR=0;
      if (LS2 <= 200)
      {
      open();
      PID = start_process(motor_update());
        ML = -50; MR = -50; wait(1500);
```

```
        MR = 50; wait (500);
        kill_process(PID);
        motor(0,0); motor(1,0);
        poke(0x4000,0x20);
        LS2 = analog(1);
      while (LS2 <= 200)
            {
            poke(0x4000,0xa0);
            LS2 = analog(1);
            }
        PID1 = start_process(read_sensors());
      PID = start_process(motor_update());
        DIR = 1;
        }
}

void lift_arm()
{
      kill_process(PID);      /* kill motor_update */
        motor(0,0); motor(1,0);
      if (LS1 <= 200)
      {
        kill_process(PID1);
      close();
            while (LS1<=200)
            {
            poke(0x4000,0xc8);
            LS1 = analog(1);
            }
      wait(10);
      PID = start_process(motor_update());
        PID1 = start_process(read_sensors());
        DIR = -1;               /* start follow */
      }
}

void battery()
{
     if (BAT > 115)      poke(0x5000,0xff);
     else if (BAT > 110) poke(0x5000,0x7f);
     else if (BAT > 105) poke(0x5000,0x3f);
     else if (BAT > 100) poke(0x5000,0x1f);
     else if (BAT > 95)  poke(0x5000,0x0f);
     else if (BAT > 90)  poke(0x5000,0x07);
     else if (BAT > 85)  poke(0x5000,0x03);
     else if (BAT > 80)   poke(0x5000,0x01);
     else poke(0x5000,0x00);
}
void joystick()
{
      while (1)
      {
      if (joy > 30)
                {
                beep();
                if (LS1 > 100) drop_arm();
                else lift_arm();
```

```
                }
        else if (joy >= 19) {ML = 0; MR = 100;}
        else if (joy >= 17) {ML = 100; MR = 0;}
        else if (joy >= 13) {ML = 100; MR = 100;}
        else if (joy >= 7)  {ML = -100; MR = 100;}
        else if (joy >= 3)  {ML = 100; MR = -100;}
        else if (joy >= 1)  {ML = -100; MR = -100;}
        else {ML = 0; MR = 0;}
        defer();
        }
}

void output()
{
        init_serial();
        put_char(27); put_char(91); put_char(50); put_char(74);
        put_char(27); put_char(91); put_char(59); put_char(72);
        while(1)
        {
        put_char(27); put_char(91); put_char(59); put_char(72);
        print("IR2 = %d.    ", IR2); put_char(10); put_char(1 3);
        print("IR3 = %d.    ", IR3); put_char(10); put_char(13);
        print("IR4 = %d.    ", IR4); put_char(10); put_char(13);
        print("IR5 = %d.    ", IR5); put_char(10); put_char(13);
        print("IR7 = %d.    ", IR7); put_char(10); put_char(13);
        print("IR8 = %d.    ", IR8); put_char(10); put_char(13);
        print("IR9 = %d.    ", IR9); put_char(10); put_char(13);
        print("IR10 = %d.    ", IR10); put_char(10); put_char(13);
        print("IR11= %d.    ", IR11); put_char(10); put_char(13);
        print("IR12 = %d.    ", IR12); put_char( 10); put_char(13);
        print("IR13 = %d.    ", IR13); put_char(10); put_char(13);
        print("IR14= %d.    ", IR14); put_char(10); put_char(13);
        print("JOY = %d.    ", joy); put_char(10); put_char(13);
        print("LS1 = %d.    ", LS1); put_char(10); put_char(13);
        print("LS2 = %d.    ", LS2); put_char(10); put_char(13);
        print("LS3 = %d.    ", LS3); put_char(10); put_char(13);
        print("FRONT = %d.    ", front); put_char(10); put_char(13);
        print("REAR  = %d.    ", rear); put_char(10); put_char(13);
          print("BATT  = %d.    ", BAT); put_char(10); put_char(13);
        defer();
        }
}

void main()
{
        poke(0x4000,0x00);
          wait(2000);
        init_motors();
          poke(0x4000,0x30);
        front = analog(1);
        poke(0x4000,0x28);
        rear = analog(1);
        if (rear > 10)
              {
                  start_process(output());
              start_process(read_sensors());
              }
```

```
        else if (front > 10)
            {
                poke(0x4000,0x20);
            LS2 = analog(1);
            while (LS2 <= 200)
                  {
                  poke(0x4000,0xa0);
                  LS2 = analog(1);
                  }
            poke(0x4000,0x00);
            LS3 = cds(2);
            open();

                PID1 = start_process(read_sensors());
                PID  = start_process(motor_update());
            start_process(joystick());
            }
        else
                {
                alert_tune();
            poke(0x4000,0x20);                  /* set arm and claw to
default position */
                LS2 = analog(1);
                while (LS2 <= 200)
                  {
                  poke(0x4000,0xa0);
                  LS2 = analog(1);
                  }
            poke(0x4000,0x00);
            LS3 = cds(2);
            open();
                PID1 = start_process(read_sensors());
            PID2 = start_process(seek());
            PID3 = start_process(follow());
                DIR = 1;
                PID = start_process(motor_update());
                }
}
```