Bull Pen: A Predator - Based Mobile Robot

Final Report

Designer: Darren S. Walton

University of Florida

EEL 5666 Intelligent Machine Design Laboratory

Fall 1997

Instructors:	Keith L. Doty
	Antonio Arroyo
TA's:	Scott Jantz
	Kevin Harrelson
	Aamir Qaiyumi

December 12, 1997

Table of Contents

Abstract 3

Executive Summary 4

Introduction 6

Integrated System 6

Mobile Platform 7
- Microprocessor 7
- Chassis 7
- Motor Actuation 7

Sensors 8
- Bump Sensor 8
- Sharp IR Detectors 9
- PIR Motion Detectors 9
- CDS Light Detection Cells 10

Behaviors 11
- Bump Detection 11
- Object Avoidance 12
- Motion Following 12
- Light Following 12
- Wandering 13

Conclusion 13

Appendix A: Source Code 14

Abstract

This report is based upon the construction of a predator based mobile robot for Intelligent Machine Design Laboratory, EEL-5666. The robot is designed to perform the predator function in a predator/prey combination. The prey device is a modified “bump-and-go” child’s toy with 32 kHz IR emitters, a capture shut-off device, and a heat source. The predator is supposed to detect the motion of the prey and the 32 kHz IR emitted from the prey to locate the direction of the prey and attack it. The robot construction took place in 3 phases: Platform construction, sensor development, and behavior programming. When completed the robot should be able to detect the motion of the prey device, move towards the prey, and disable the prey.

Executive Summary

The robot designed in this project is the predator element of a predator/prey combination. The main purpose of the robot is to track-down and disable the prey device through the use of passive infrared motion sensors and 32 kHz infrared detection.

The robot was built on a standard Talrik platform for ease of platform design. Modifications of the platform were necessary to mount the motion sensors and the capture arm.

The sensor suite consists of 4 Sharp 44 kHz infrared detectors, 1 Sharp 32 kHz infrared detector, 3 Leviton passive infrared occupancy sensors, 10 microswitches, and 5 cadmium sulfide light sensitive resistors. The 44 kHz IR detectors are used for object avoidance. The 32 kHz detector is input for locating the prey device when directly in front of the predator. The occupancy sensors detect prey movement. The microswitches are used for detection of bumps by the robot into objects which do not reflect sufficient infrared to be detected by the 44 kHz detectors. And the Cadmium Sulfide cells are used to locate the direction of highest light intensity.

The source code for the behaviors was written using the interactive -C environment and the C programming language. The current behaviors integrated are bump, object avoidance, and motion following (chasing).

Introduction

The basis for my project was a predator robot capable of detecting the motion of its prey, chasing the prey, and capturing the prey by disabling it. In order to accomplish this the robot must have some method of detecting the motion of objects within its surroundings. The predator must also be able to distinguish the prey from other non-prey moving objects. Using infrared motion sensors and equipping the prey with a heat device allows the predator to distinguish the motion of the predator device, and by causing the predator to pause when checking the motion detectors will eliminate the false motion detections due to motion of the robot.

Integrated System

The robot is assembled on the basic Talrik platform with the addition of 3 passive infrared motions sensors. Standard Talrik hacked servo-motors are used for the drive wheels, and the behavior of the robot is controlled primarily by an arbitration function which determines which of the behaviors will control these motors. Sensor input is gathered and assigned to global variables which in turn are used by behavior algorithms to generate signals for the left and right wheel motors.

Upon reset the robot enters the searching mode to detect movement. It performs 8 checks in different directions (pivoting CCW), and if no motion is detected it begins wandering and avoiding objects for 10 seconds to attempt to clear obstacles between itself and possible prey. If a detection occurs the robot heads in the direction of the motion, avoiding obstacles, and attempts to intercept the prey. When the prey is in front of the robot and within 8 - 10 inches the 32 kHz IR signal indicates that the object is the prey and the robot lurches forward in an attempt to capture its prey.

Mobile Platform

Microprocessor

The MC68HC11E9 was chosen as the brain of the computer because of the availability of the EVBU board, as well as due to the abundance of technical support offered by the course assistants. Expanding the EVBU with the ME11 expansion board added the necessary additional memory and memory mapped I/O features for the project.

Chassis

The body of the robot is a slightly modified Talrik design, with substituted solid rubber wheels for the standard air filled tires. The MC68HC11 EVBU and ME11 expansion board are mounted in the center of the body, and the motion sensors are mounted along the bridge.

Motor actuation

The motion of the robot is realized by two hacked servo-motors which were provided with the Talrik partial kit. They are driven by the motor driver included on the ME11 expansion board using pulse width modulated signals produced from the MC68HC11 processor on output port A.

Sensors

Bump Sensor

The bump sensor is composed of an acrylic bumper which entirely encircles the main frame of the Talrik body. This bumper actuates 10 evenly spaced momentary microswitches which are connected to binary input at address 0x4000. The switches are wired such that the lower nibble of the address is the front 5 switches and the higher nibble is the rear 5 switches and generates the following decimal values as shown in the table below.

Switch Position	**Decimal value at 0x4000**
Far Left Front	1
Far Left Front and Front Left	3
Front Left	2
Front Center	6
Front Right	4
Front Right and Far Right Front	12
Far Right Front	8
Far Left Back	16
Far Left Back and Back Left	48
Back Left	32
Back Center	96
Back Right	64
Back Right and Far Right Back	192
Far Right Back	128

Sharp IR Detectors

There are 4 44 kHz sensitive Sharp Infrared light detectors mounted along the front edge of the robot body. These sensors have been hacked to provide an analog output signal which corresponds to the level of 44 kHz flicker IR light that is detected. The 4 detectors are arranged evenly spaced each “looking” out in a direction perpendicular with the edge of the robot. These detectors are used exclusively for object avoidance when the robot is searching or wandering.

Additionally there is a similarly hacked 32 kHz Infrared light detector mounted below the center motion sensor. This detector senses the presence of the prey vehicle which is equipped with 32 kHz IR emitting IR LEDs. This device is used to indicate the near proximity of the prey device.

<u>Leviton Passive Infrared Motion sensors</u>

Mounted along the bridge are 3 hacked motion sensors. These sensors detect changes in the heat zones in a conical region directly projected from the front of each sensor. The sensors deliver a 4 millisecond active low signal for each zone change detected. They have been spaced and mounted to provide maximum area coverage to the forward direction of the robot, without having significant overlap in the searched areas. These detectors provide the input to allow the robot to search for and detect the moving prey object.

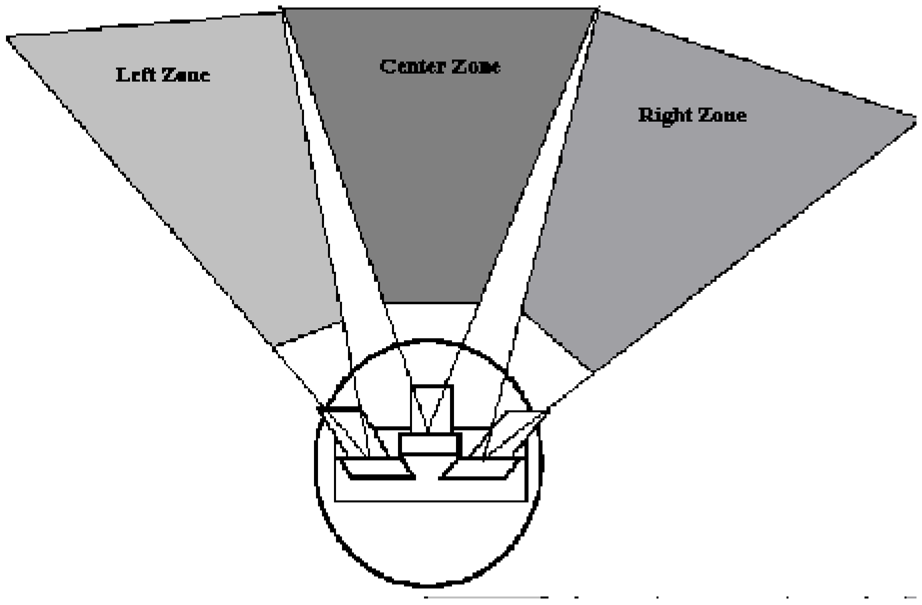

Figure 1: Motion Detector placement on robot body

CDS Light Detectors

Five Cadmium Sulfide photo resistors are mounted along the top of the motion sensors. These CDS resistors are arranged in a voltage divider circuit, each with a 100 K-Ohm resistor as shown in figure 2. The resistors decrease their resistance as light input increases. They are used to detect the flashing lamps on the prey vehicle. As light increases the resistance decreases across the CDS cell and thus the voltage at point A in figure 2 decreases.

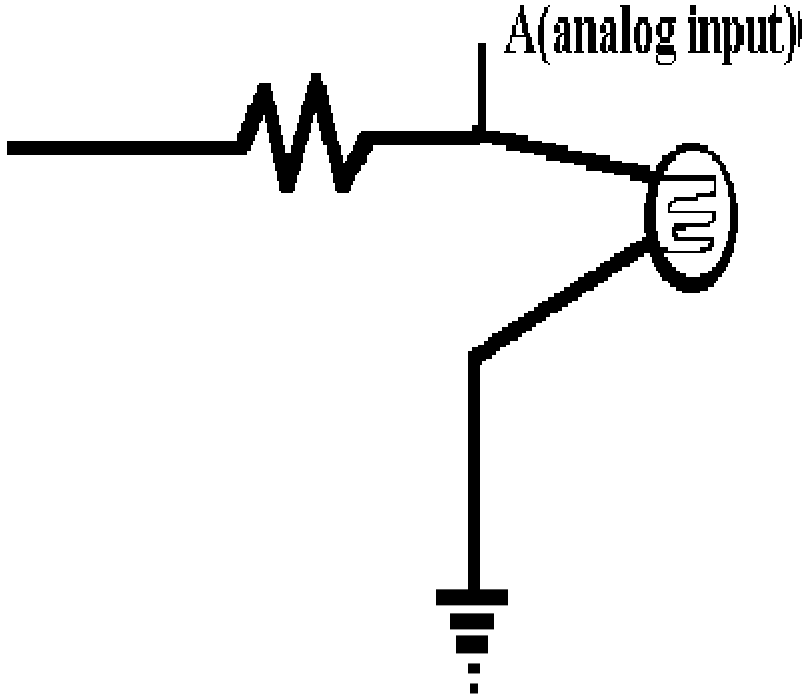

Figure 2: CDS cell voltage divider for light detection

Behaviors

The robot exhibits the following behaviors using the indicated sensors as inputs.

Bump

The robot backs away from objects which it has bumped into. The priority of this behavior is determined by the value of the binary inputs of the bump switches. This behavior overrides all behaviors except the attack behavior, in which case the robot must bump its forward section into the prey to cause the prey to be disabled.

Object Avoidance

Using the 44 kHz Sharp sensor output values as inputs, this behavior allows the robot to maneuver about in its environment as it wanders, searches for the prey, or attacks the prey without running into obstacles. Since the prey vehicle is small and virtually non-detectable by reflected IR (due to its dark color) the robot will not attempt to avoid the prey as an obstacle. Even if it did, the object avoidance maximum priority is less than the attack behavior's priority.

Motion Chasing

By using the motion detectors the robot attempts to locate the relative direction of the prey vehicle and then moves in that direction, hoping to get a positive detection by the 32 kHz IR sensor. When a motion is detected by the center sensor the robot moves forward for a fixed time then stops and waits for another motion detection. A detection by either of the left or right sensors causes the robot to pivot in that direction and then move forward as in the case of the center sensor. If no detection is registered before the time-out then the robot will pivot counter clockwise and start the detection timing cycle again.

Light Chasing

By using the 5 CDS cells located at on the top of the robot the robot can detect the light of the lamps on the prey object as being brighter than the surrounding ambient light. This provides an additional input as to the direction of the prey in relation to the predator.

Wandering

When the robot completes a full circle of motion detection pauses it will become bored with detection and begin wandering about in its environment by pivoting in a pseudo random direction and moving forward for a finite period of time. At the end of the wandering phase the robot reverts to motion chasing.

Attacking

When the robot has detected an increase in the 32 kHz sensor reading it interprets this input as confirmation that the prey vehicle is “near” and “dead-ahead”, in which case the robot goes into a full forward lunge to attempt to capture its prey. In this mode all object avoidance and bump detection is overridden.

Conclusion

Although test runs of the behaviors separately were generally successful, the overall integrated behavior did not function properly at the time of the writing of this report. Several hours of testing and debugging remain before the project will function as anticipated. The processor seems to have difficulty detecting the motion sensor signals, and the servo motors are in such bad shape that the one running in reverse has a speed of about 50 % that of the forward servo. This results in insufficient speed to catch up with the prey.

Appendix A: Source Code

```
/* This is the global variables and constants, as well as the main
program
for the control of Darren Walton’s predator robot.     */

        /* global defined values */

int on = 1;
int off = 0;

int min_IR = 90;
int max_IR = 125;

int zone = 6;
int priori_T = 25;

int pivot_time = 1500;         /* the time of each pivot in mseconds */
int pivot_dir = 0;              /* Direction of the pivot */

float fmax_motor_port = 100.0;   /* The max port and stbd motor commands
*/
float fmax_motor_stbd = 70.0;   /* to achieve "straight" movement */
float rmax_motor_port = -80.0;
float rmax_motor_stbd = -100.0;

        /* The behaviour priorities */

int avoid_pri = 1;
int bump_pri = 2;
int search_pri = 3;
int wander_pri = 4;
int attack_pri = 5;
int pivot_pri = 6;
int end_pri = 7;

int priorities[10]={0,0,0,0,0,0,0,0,0,0};              /* The priority
Array   */

int max_pri = 1;
int old_max_pri = 1;

/* The variables for the motor_control function.  These are global so
that
the motion detection behaviour can tell when the robot is really
stopped. */

float slew = 10.0;
float port = 0.0;
float stbd = 0.0;

        /* The global variables */

int detect_p;           /* The detection boolean values */
int detect_c;
int detect_s;
int detect = 0;

float detect_time;      /* The times for detection  */

int IR_out = 0x7000;                  /* The IR output latch address
*/
int binary_out = 0x4000;              /* The binary out port address
*/
int binary_in = 0x4000;               /* The binary in port address
*/
int bin_out = 0;
int bin_in = 0;                         /* The binary i/o values
*/
int bump_bar = 0;                       /* The bumper/ also bin_in
*/
int pivots = 0;
int searches = 0;


float motor_port = 0.0;
float motor_stbd = 0.0;                 /* The motor driver commands
*/

                                        /* the motor value arrays */
float port_motor[10] = {0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0.,0.0};
float stbd_motor[10] = {0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0.,0.0};

int port_move = 200;
int center_move = 200;
int stbd_move = 200;                    /* The motion detect values
*/

int nose = 0;                           /* The 32 kHz IR detector
*/

int port_ir = 100;                      /* The 44 kHz IR detectors
*/
int port_cntr_ir = 100;
int stbd_cntr_ir = 100;
int stbd_ir = 100;

/* The 44 kHz IR input array index and value array      */
int biggest = 0;
int ir[4];

/* CDS cells
int cds_hard_port = 0;
int cds_mid_port = 0;
int cds_center = 0;
int cds_mid_stbd = 0;
int cds_hard_stbd = 0;

*/
/* This file is the function library for Darren Walton's predator robot
*/
/*===================================================*/
/* Included functions and calling mechanisms

void wait(int milli_seconds)    This function causes the processor to
                        wait for milli_seconds milliseconds, but allows
                        the processor to continue on with other
processes

int analog7(int x)      This function sets the analog mux input to input
                        number x (0 - 7) and returns the value that is
                        detected there.  The delay is for the A/D port
                        to setup.

int binout(int bit, int val)
                        This function sets the binary output 0-3 to on
                        or off: 0 or 1.  no return value, however the
                        global variable bin_out is modified.
*/

/*===================================================*/
```

```
/* The waiting function, better than sleep because it allows other
processes to proceed immediatly */
void wait(int milli_seconds)
{
long timer_a;

timer_a = mseconds() + (long) milli_seconds;
while(timer_a > mseconds()) {
        defer();
        }
}

/*=================================================*/

/* analog7(int x)       input: x is an integer in the range 0-7
                        output: return value is the value at the
multiplexed
                                analog input pin x, OR 255 if error.
*/
int analog7(int x)
{
/* Check that the input value is in range 0 - 7
*/
if ((x >= 0) && (x <= 7))
{       /* First clear the lower 3 bits in the global variable */
        bin_out = bin_out & 0xf4;

        /* Now set the bits according to x */
        bin_out = bin_out | x;

        /* Write to the output to set the select lines */
        poke(binary_out, bin_out);

        /* Read and return the value */
        return analog(7);
}
/* If input out of range return an impossible A/D value of 255
*/
else return 255;

}

/*================================================*/

/* binout()     Writes to the 0x4000 output port for the unused binary
                        output values...bits 5, 6, and 7.
                input:  bit, the bit number to write out 0-2 (bit 5-7)
                        val, the value of the bit to output 1 or 0
                output: returns 1 if successful, 0 if unsuccessful
*/

int binout(int bit, int val)
{
        int mask;

/* No sense screwing around with improper values */
        if ((val <= 1)&&(val >= 0)&&(bit >= 0)&&(bit <= 2))
        {       /* Set the mask value for that bit */
                mask = (1 << (5+bit) );

                /* determine if clearing or setting */
                if (val == 0)
                {  /* complement the mask and clear bit */
                        mask = ~mask;
                        bin_out = bin_out & mask;
                }
                else    /* set the bit */
                {       bin_out = bin_out | mask;
                }

                /* Now write to the memory location */
                poke(binary_out, bin_out);

                /* Now the successful return value */
                return 1;
        }
        /* bad input values begets a "0" return value */
        else return 0;
}
/*==================================================*/
/* Control the arms of the robot for extension and retraction.  The
        arm motor is controlled by bits 3 and 4 of output port 0x4000.
        By setting bit 3 high the motor turns on
        By setting bit 3 low the motor turns off
        By setting bit 4 high the arms extend
        By setting bit 4 low the arms retract   */


void arms_out(void)
{
        int mask = 0x18;
/* Change the binary output value */
        bin_out = bin_out | mask;
/* Now write to the output port         */
        poke(binary_out, bin_out);
}

void arms_in(void)
{
        int mask = 0x08;
/* Change the binary output value */
        bin_out = bin_out & 0xE7;
        bin_out = bin_out | mask;
/* Now write to the output port */
        poke(binary_out, bin_out);
}

void arms_stop(void)
{       int mask = 0xE7;
/* Change the binary output value */
        bin_out = bin_out & mask;
/* Now write to the output port */
        poke(binary_out, bin_out);
}

/*===================================================*/
/* This file is the file containing the behaviours and modules which
need
to be called using "start_process()" function.
The "behaviours" and other called processes which are each called as a
process in the main program of the Predator robot

void motor_control(void)        This is the main motor controlling
function that
                        causes the motor speed and direction to ramp to
                        the new value and not cause instantaneous
changes
                        to motor speed and direction.

void sensor()           This function gathers the incoming data from all
```

```
                        the physical sensors and stores them in global
                        variables.

void arbitrator()       The arbitrator function to arbitrate between the
                        active behaviours.

void pivot()            Pivots the robot aprox 45 degrees in the
direction
                        set by the global variable: pivot_dir   0 = none
                                                               -1 = port
                                                                1 = stbd

void object_avoid()     Avoids objects using the IR detectors values
from
                        the sensor() function.

void search()           The motion detection following behaviour which
moves
                        the robot in the direction of detected motion.
This
                        also affects the pivot behaviour to enable
pointing
                        the robot towards the moving objects direction.

void bump()             Causes the robot to move away from an object
that
                        it has bumped into.

void wander()           The robot wanders aimlessly if there is no input
                        from the search() behaviour after waiting for
the
                        motion detection in eight directions.
*/

/*================================================*/
/* The following functions are in this file and have been tested:

motor_control()
arbitrator()
sensor()
object_avoid()
bump()
search()
pivot()
wander()

These are being tested:
attack()

*/
/*=====================================================*/
/* Motor Control function.  Directly controls the motor commands based
upon the global variables set by the arbitrator function
*/
/*========================================================*/
void motor_control()
{
while(1)
{
        port = (port*slew + motor_port)/(slew + 1.0);
        stbd = (stbd*slew + motor_stbd)/(slew + 1.0);

        motor(0, port);
        motor(1, stbd);
        defer();
```

```
}
}
/*=================================================*/
/* The arbitrator function */
/*===================================================*/
/* The arbitrator function that compares the behaviour priorities and
then determines which behaviour determines controll of the robot
movement */
void arbitrator(void)
{
int i;
while(1)
{
i = 1;
old_max_pri = max_pri;
while(i < 10)
{       if((priorities[i]) > (priorities[max_pri]))
                max_pri = i;
        i++;
}
if(old_max_pri != max_pri)
        tone(500., 0.50);
motor_port = port_motor[max_pri];
motor_stbd = stbd_motor[max_pri];
defer();
}
}

/*###################################################*/
/* Sensor function to gather all function data and assign it to global
        variables for use by the arbitrator function and other functions
*/

void sensor()
{
while(1)
{
/* The motion detectors */
        port_move = analog(4);
        center_move = analog(5);
        stbd_move = analog(6);
/* The 32 KHz IR detector */
        poke(0x4000, 0x00);
        nose = analog(7);

/* The bump switch all about the perimeter */
        bin_in = bump_bar = peek(binary_in);

/* The 44 KHz IR detectors */
        port_ir = analog(0);
if((port_move > 190)&&(stbd_move > 190)&&(center_move > 190))
{
        port_move = analog(4);
        center_move = analog(5);
        stbd_move = analog(6);
}

        port_cntr_ir = analog(1);

if((port_move > 190)&&(stbd_move > 190)&&(center_move > 190))
{
        port_move = analog(4);
        center_move = analog(5);
        stbd_move = analog(6);
}
```

```
        stbd_cntr_ir = analog(2);

if((port_move > 190)&&(stbd_move > 190)&&(center_move > 190))
{
        port_move = analog(4);
        center_move = analog(5);
        stbd_move = analog(6);
}

        stbd_ir = analog(3);

if((port_move > 190)&&(stbd_move > 190)&&(center_move > 190))
{
        port_move = analog(4);
        center_move = analog(5);
        stbd_move = analog(6);
}

/* Sample all of the CDS cells from the analog mux */
/*        cds_hard_port = analog7(1);
        cds_mid_port = analog7(2);
        cds_center = analog7(3);
        cds_mid_stbd = analog7(4);
        cds_hard_stbd = analog7(5);
*/
/* Defer to the next process    */

        defer();
}
}

/*####################################################*/
void object_avoid(void)
{
int i;
while(1)
{
/* Put the IR readings into an array    */
        ir[0] = port_ir;
        ir[1] = port_cntr_ir;
        ir[2] = stbd_cntr_ir;
        ir[3] = stbd_ir;
/* Determine the largest value  */
i = 0;
while(i < 4)
{
        if(ir[i] > ir[biggest])
                biggest = i;
        i++;
}
/* Determine the priority of the IR signal      */
        if(ir[biggest] < 100)
                priorities[avoid_pri] = 50;
        else if(ir[biggest] < 110)
                priorities[avoid_pri] = 75;
        else if(ir[biggest] < 115)
                priorities[avoid_pri] = 100;
        else if(ir[biggest] < 120)
                priorities[avoid_pri] = 125;
        else    priorities[avoid_pri] = 135;

if(ir[biggest] < 100)
{       port_motor[avoid_pri] = fmax_motor_port;
        stbd_motor[avoid_pri] = fmax_motor_stbd;
```

```
}
else
{
if(biggest == 0)
{
        port_motor[avoid_pri] = fmax_motor_port;
        stbd_motor[avoid_pri] = (rmax_motor_port/3.0);
}

if(biggest == 1)
{
        port_motor[avoid_pri] = fmax_motor_port;
        stbd_motor[avoid_pri] = (rmax_motor_port/2.0);
}

if(biggest == 2)
{
        port_motor[avoid_pri] = (rmax_motor_port/2.0);
        stbd_motor[avoid_pri] = fmax_motor_stbd;
}

if(biggest ==3)
{
        port_motor[avoid_pri] = (rmax_motor_port/3.0);
        stbd_motor[avoid_pri] = fmax_motor_stbd;
}
}

defer();
}
}
/*=================================================*/
/* The bump algorithm to cause the robot to move back away from an
object
that it has bumped into.  The backward motion moves the front of the
robot
to point in a direction AWAY from the direction of the bumped into
object */
/*==================================================*/
void bump(void)
{
while(1)
{
        if(bump_bar > 0)
        {
                priorities[bump_pri] = 150;
                if((bump_bar == 1)||(bump_bar == 3))
                {
                        port_motor[bump_pri] = (rmax_motor_port/4.0);
                        stbd_motor[bump_pri] = (rmax_motor_stbd);
                }
                else if(bump_bar == 2)
                {
                        port_motor[bump_pri] = (rmax_motor_port/2.0);
                        stbd_motor[bump_pri] = (rmax_motor_stbd);
                }
                else if(bump_bar == 6)
                {
                        port_motor[bump_pri] = (rmax_motor_port);
                        stbd_motor[bump_pri] = (rmax_motor_stbd*0.75);
                }
                else if(bump_bar == 4)
                {
                        port_motor[bump_pri] = (rmax_motor_port);
                        stbd_motor[bump_pri] = (rmax_motor_stbd/2.0);
                }
                else if((bump_bar == 8)||(bump_bar == 12))
                {
                        port_motor[bump_pri] = (rmax_motor_port);
                        stbd_motor[bump_pri] = (rmax_motor_stbd*0.25);
                }
                else if(bump_bar > 15)
                {
                        priorities[bump_pri] = 0;

                }
                wait(4000);
        }
        priorities[bump_pri] = 0;
        defer();

}
}
/*=====================================================*/
/* pivot behaviour, pivots the robot based upon global variables so
other
behaviours can controll the direction of the pivot */
/*===================================================*/
void pivot(void)
{
while(1)
{
if(pivot_dir != 0)
{
  if(pivot_dir == -1)
      { port_motor[pivot_pri] = -100.0;
        stbd_motor[pivot_pri] = 100.0;
      }
  else if(pivot_dir == 1)
      { port_motor[pivot_pri] = 100.0;
        stbd_motor[pivot_pri] = -100.0;
      }
  wait(pivot_time);
  pivot_dir = 0;
}
/* relenquish control of the robot motors */
priorities[pivot_pri] = 0;
defer();
}
}

/*#####################################################*/
/* SEARCH : The motion following algorithm */
/*=====================================================*/
void search(void)
{
        priorities[search_pri] = 65;        /* set the searching
priority level */
        detect_p = 0;
        detect_c = 0;
        detect_s = 0;

while(1)
{
if((port >= 10.0)&&(stbd >= 10.0))
{       port_motor[search_pri] = 0.0;
        stbd_motor[search_pri] = 0.0;
}
else
{       detect = 0;
```

```
        detect_time = seconds() + 10.0;

        while(detect_time > seconds())
        {
        if((port_move > 190)&&(stbd_move > 190)&&(center_move > 190))
        {
                port_move = analog(4);
                center_move = analog(5);
                stbd_move = analog(6);
        }
        detect_p = (int)(port_move < 190);
        detect_c = (int)(center_move < 190);
        detect_s = (int)(stbd_move < 190);

        /* If motion to center then just lurch forward 1.5 seconds */
                if((detect_c)||((detect_p)&&(detect_s)))
                {       port_motor[search_pri] = fmax_motor_port;
                        stbd_motor[search_pri] = fmax_motor_stbd;
                        tone(1000., 0.5);
                        wait(1500);
                        detect = 1;
                }
                /* If motion to port, pivot to port and lurch forward */
                else if((detect_p)&&!(detect_s))
                {       pivot_dir = -1;
                        priorities[pivot_pri] = 200;
                        defer();
                        port_motor[search_pri] = fmax_motor_port;
                        stbd_motor[search_pri] = fmax_motor_stbd;
                        tone(1500., 0.5);

                /* Wait for turning and forward times */
                        wait((pivot_time + 1500));
                        detect = 1;
                }
                /* If motion to stbd then pivot to stbd and lurch
forward */
                else if((detect_s)&&!(detect_p))
                {       pivot_dir = 1;
                        priorities[pivot_pri] = 200;
                /* Wait while pivoting  */
                        wait(pivot_time);
                        port_motor[search_pri] = fmax_motor_port;
                        stbd_motor[search_pri] = fmax_motor_stbd;
                        tone(2000., 0.5);

                /* Wait for forward movement */
                        wait(1500);
                        detect = 1;
                }
        }       /* End the while loop   */
        if(!detect)
        {
                priorities[wander_pri] = 70;
        }
}       /* End the else */
defer();
}       /* End the while        */
}

/*#####################################################*/
/*########################################################*/
/* The wander behavior just wanders around for about 10 seconds while
the
```

```
robot tries to find a new location.  Since the default value of the
priority
is 70 it out prioritizes search, and succumbs to the avoid and bump
behav’s. */
/*========================================================*/
void wander(void)
{
while(1)
{
        if(priorities[wander_pri] == 70)
        {       if(searches < 7)
                {       searches++;
                        priorities[wander_pri] = 0;
                        pivot_dir = -1;
                        priorities[pivot_pri] = 145;
                        wait(pivot_time);
                }
                else
                {
           /* Setup the priority to pivot the robot */
                priorities[pivot_pri] = 145;

           /* Generate a psuedo random pivot direction */
                pivot_dir = ((peek(0x1000)%3)-1);

           /* Generate a psuedo random number of pivots */
                pivots = (peek(0x1000)%8);

           /* and pivot that number of times */
                 while(pivots)
                 {       pivots--;
                         wait(pivot_time);
                 }

                /* Now move forward and avoid obstacles for 10 seconds
*/
                  port_motor[wander_pri] = fmax_motor_port;
                  stbd_motor[wander_pri] = fmax_motor_stbd;
                  wait(10000);


                }
                /* Now end wandering */
                priorities[wander_pri] = 0;
                defer();

        }

}
}

/*########################################################*//*##########
##############################################*/
/* The attacking behaviour */
void attack(void)
{
        while(1)
{
        if(nose > 59)
        {       priorities[attack_pri] = 300;
                  port_motor[attack_pri] = fmax_motor_port;
                  stbd_motor[attack_pri] = fmax_motor_stbd;
                  wait(10000);
                priorities[attack_pri] = 0;
        }
}
}

/* The main program for the Predator Robot, BullPen by Darren Walton */
void main()
{
/* Init the IR leds     */
        poke(0x7000, 0xff);

/* Init the analog mux */
        poke(0x4000, 0x00);

/* Start reading in the sensor data */
        start_process(sensor());

/* Start the behaviour usage processes */
        start_process(motor_control());
        start_process(pivot());

/* The behaviours */

        start_process(search());
        start_process(object_avoid());
        start_process(bump());
        start_process(wander());
        start_process(attack());
/*        start_process(end());         */

/* The arbitrator to determine the controlling behaviour */
        start_process(arbitrator());

}
```