Jormungand

Final Report

Charles W. (Billy) Eno

December 9, 1998

University of Florida
Intelligent Machines Design Laboratory

Table of Contents

Abstract
Jormungand is an autonomous robotic snake that was designed to be maneuverable and rugged. He moves without the benefit of legs or wheels through serpentine motions. A pair of Mekatronix MSCC11 boards controls him, implementing servo control on one board and sensor control on the other. IR is used for obstacle avoidance. Bump switches are used to initiate the rattle and sidewinder behaviors. Mercury tilt switches allow the robot to tell if he is upright. The robot shows some good behaviors and is a good basis for future expansion.

Executive Summary

Jormungand is an autonomous robotic snake. The goal of the project was to provide a rugged and maneuverable platform for an array of sensors.

Jormungand was designed with ease of assembly and reuse of parts in mind. All his segments are basically identical. They are simply rotated 90 degrees from one another to provide both up-down and left-right movement. The head is designed to easily mount many different sensors and the tail has a rattle to let the outside world know when the snake is upset.

Two Mekatronix MSCC11 boards control the robot. One board provides servo control and communicates via the SCI system with the other board which controls sensor and higher functions.

There are three sets of sensors. IR is used for obstacle avoidance. Bump switches initiate behaviors such as the rattle and the sidewinder move. The tilt sensors are mercury switches and let the snake know when it is not upright.

The movements of the snake are similar to an inchworm and provide a slow but steady pace for the forward progress. The robot turns fairly well, within a five-foot circle. The sidewinder move allows the snake to move straight sideways but is not fully integrated due to limited program space on the boards.

Much of the emphasis of the project has been on the construction of the platform and the

development of some basic movements for the robot. Future work will expand the sensor array and increase the complexity of the behaviors.

1. Introduction

In the Intelligent Machines Design Laboratory (IMDL) the best part, in my opinion, is the encouragement one gets to build on other peoples designs and ideas as well as come up with your own, new designs ideas. No one person would be able to do as much in one semester if this chance to explore previous work was unavailable. This is why the robots in IMDL keep getting better and better the longer it is in existence. I started the class wanting to do a snake robot. There had been one previous attempt at a snake (Monty, Melissa Jones, Fall '97) and I felt I could do an even better snake. The snake design I came up with was completely different from her design, but I was able to learn from some of her mistakes. The design incorporates original code, code from the standard IMDL library, as well as from other sources. Many parts of the snake are inspired by previous work, from the IR sensors to the bump switches.

The project presented a challenge: Create a highly flexible, maneuverable robot that could incorporate interesting behaviors without legs or wheels. My design went through several stages where I refined the platform and developed the techniques to control and move my robot.

In this paper I will go through a general overview of the project and proceed to specifics about each individual aspect. After the conclusion I will present my thoughts on how I plan to improve on my work in the future.

2. Integrated System

Jormungand is an autonomous snake robot and has no legs or wheels. He is built in an open frame structure that is rugged and flexible. His movement is controlled by nine servos, one at each joint of his body. He accomplishes movement through various serpentine movements. His multiple joints provide many ways to manipulate his body.

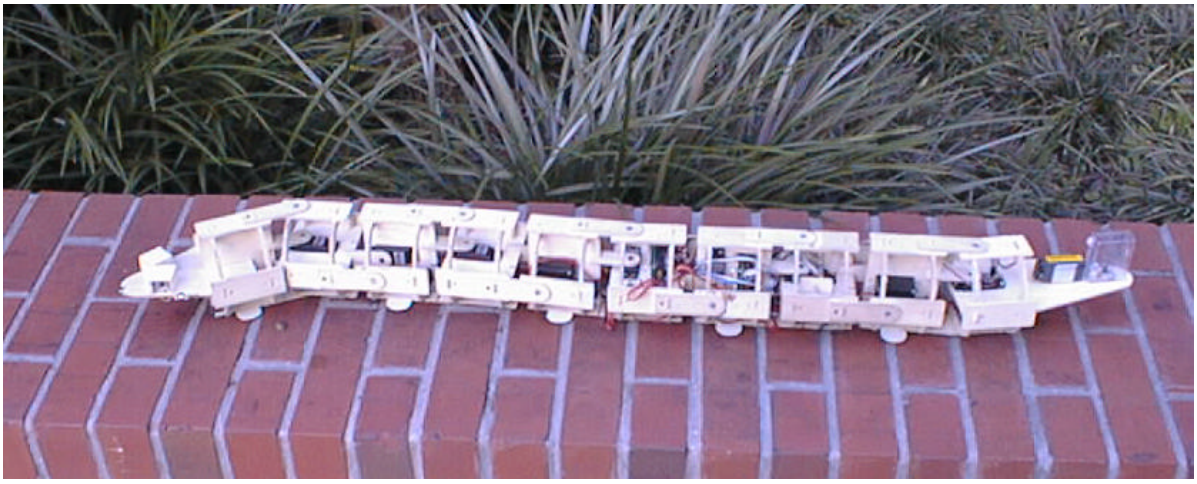
Jormungand is controlled by two MC68HC11-E2 single chip boards. A servo controller board manipulates the servos and the brain board controls the sensors and tells the servo controller how to position the servos.

He perceives the world through three sets of sensors. IR provides basic obstacle avoidance. Bump sensors initiate various actions. Tilt sensors help him tell when he is on his side. Jormungand's programming resides in a continuous loop that keeps him moving forward while avoiding obstacles and staying upright.

3. Mobile Platform

3.1 Overview of the Platform

The snake platform was designed to be simple to assemble with many repeated parts. Each segment of Jormungand is essentially identical. The overall platform was designed to provide the maximum amount of freedom of movement with a minimum amount of complexity. Along with this was the need for an extremely rugged and durable snake. (see picture below)



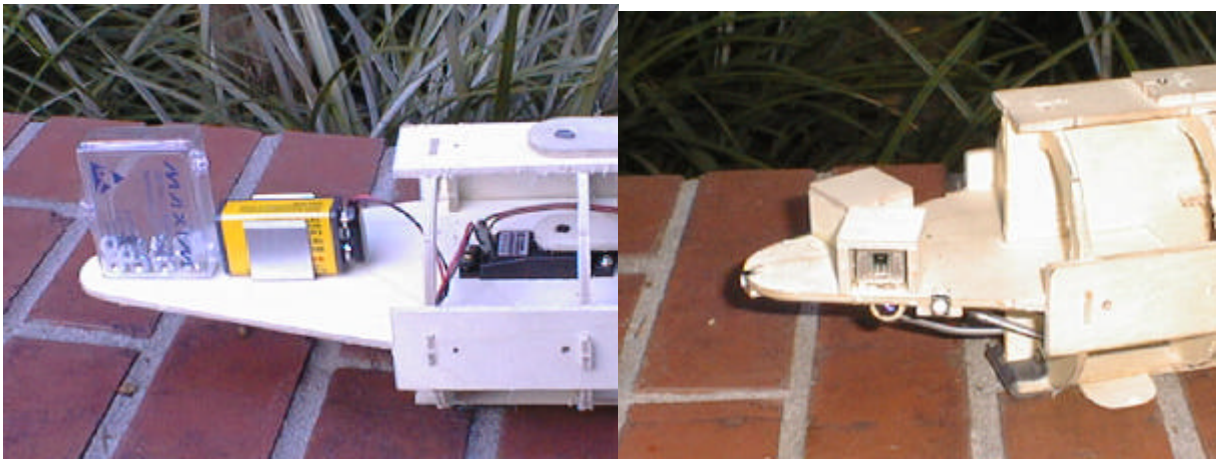
Overall Picture

3.2 Overall Design

I started drawing my snake long before the class actually started. My original thought was to have either thin walled aluminum or plastic pipe to make up each segment. As I sketched the designs it became apparent that connecting these segments and allowing them sufficient free movement would be difficult. I realized that tubes would require internal structures to mount servos, batteries and other components such as sensors. This internal structure would be hard to mount components to, and once installed, would be nearly impossible to remove or adjust as needed. The solution was to remove the outer tube and just leave the internal structure. This would provide easy access to components and allow for easy assembly.

3.3 Head and Tail Design

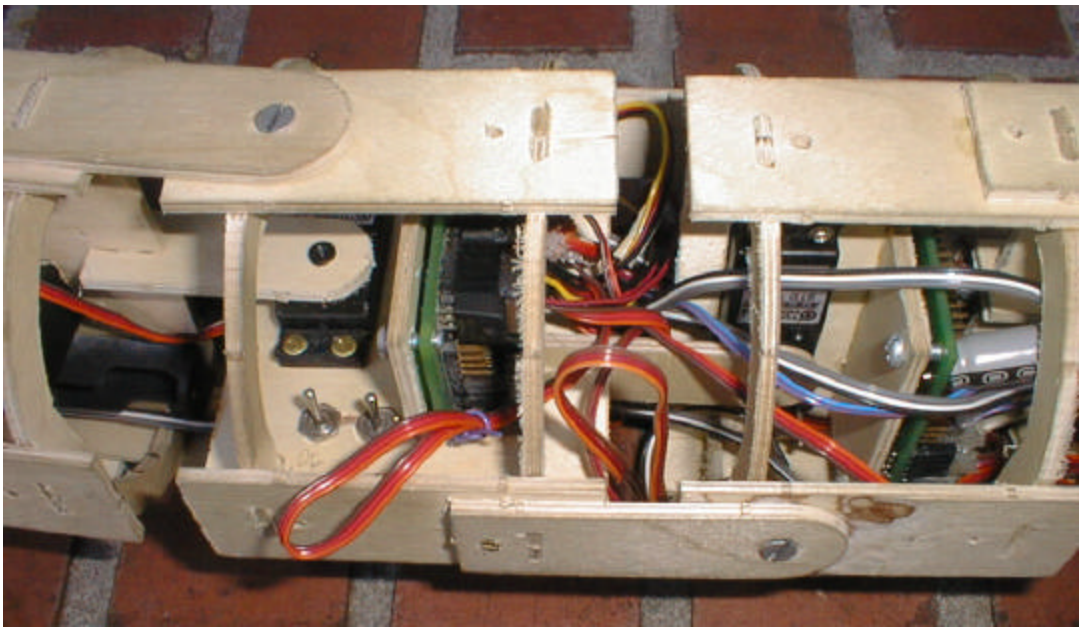
The head was designed to provide a good platform for the sensors. Both the IR and bump sensors are mounted on the head with plenty of room for other sensors. The tail is similar to the head and has the battery for the computers mounted on it as well as a rattle to warn the outside world when Jormungand is upset.



Tail and Head

3.4 Processor Segment Design

The minimum diameter of the snake was dictated by the size of the microprocessor boards. After experimenting with some smaller designs I realized I wasn't up to the task of modifying existing boards or making my own, I wanted to concentrate on making an interesting robot, not reinventing a single-chip board. This resulted in a snake 4 inches in diameter. The boards are mounted perpendicularly to the axis of the snake in two special segments. While the design involving the boards functions well, they would be impossible to replace, in the event of failure, without breaking the segment, and it is difficult to attach plugs to the header, due to the close spacing involved.



Processor Segments

3.5 Joint Design

I intended to have each joint have two degrees of freedom. After I designed such a joint I realized it would be needlessly complex. I settled on alternating directions of movement on each segment. This reduced the number of servos needed, an important factor in cost. In addition, it reduced the weight in each segment and allowed the overall snake to be longer.

3.6 Construction

Jormungand is assembled from 1/8" birch plywood. The pieces were drawn in AutoCAD and cut out on the T-Tech machine. Assembly was done initially with wood glue and later with super glue. Super glue is by far easier and faster to use. The only hardware involve were small screws to mount the servos and bolts and lock nuts used for the hinges. Some hinge arms are attached with screws as well as glue, but with experimentation I realized the screws were unnecessary and they do not appear on all the segments.

4. Actuation

4.1 Segment Actuation

Actuation is provided by servos. The servos use were Tower Hobbies' STD TS-53, a 42 oz-in servo that only costs about $12. They were chosen for their inexpensiveness, but aside from one that never worked, they proved to be reliable. Each segments consists of a body frame and a servo. The servo attaches via a wooden arm to the next segment. Each segment is rotated ninety degrees from the previous and next segments to provide for both up-down and left-right. The segments hinge on wooden arms protruding from the previous section. The hinge lines up with the pivot of the servo. The Hinges reduce strain on the servo itself and provides for solid connection between segments

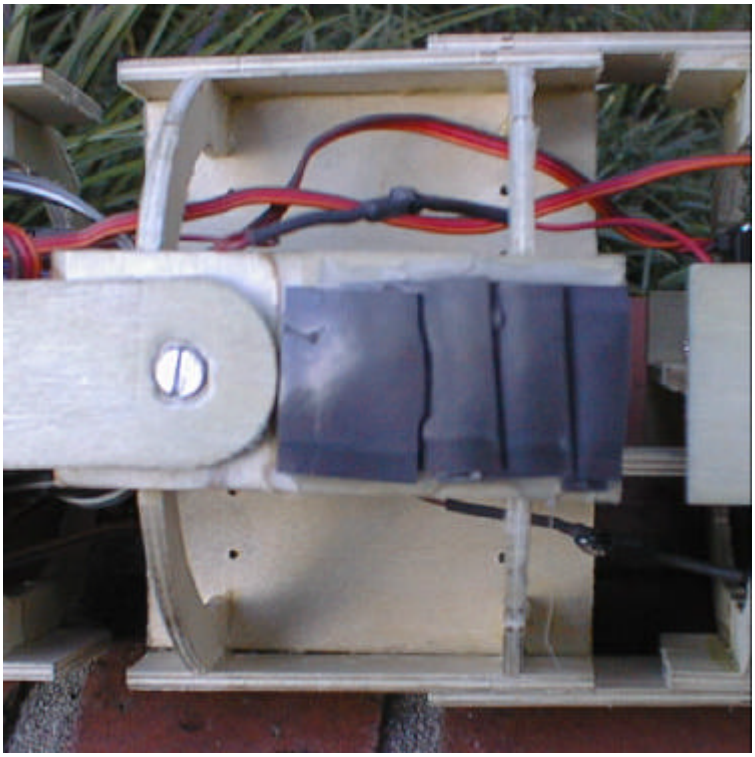
4.2 Servo Control

Servos are attached to a MSCC11 single chip board from Mekatronix. The servos use code written by Drew Bagnell and a serial interface I wrote (all code is in Appendix A). The servos require fairly strict time dependent signals to operate correctly. This is why I chose to have a separate servo controller. This also freed up space on the main board for additional movements and behaviors.

4.3 Basic Movement

One of the primary goals for Jormungand was to have him move like a snake. Serpentine movements are interesting and hard to duplicate I found. Of course, snakes are much more flexible than a robot can be. Jormungand's movement is primarily an inch-worm style

movement. He lifts his tail, pulls it forward, sets it down, pushes back and then propagates the "wave" created in his body forward to the front of his body. This pushes him forward at a slow but steady pace. This movement, while not tremendously serpent-like, is like a scaled up version of what snakes do to move. To turn, he merely bends all of the left-right segments in the direction he wants to go. This has the effect of guiding the movement in the desired direction. The scales on the snake's stomach maintain traction. These allow the snake to slip forward, but keep him from moving backwards.



Scale Detail

4.4 Sidewinder Movement

The sidewinder movement allows the snake to move straight sideways. He does this by lifting his tail, pulling it to the left or right, setting it down and then moving up his body doing the same with the rest of his segments. This movement is not well integrated into the overall movement because of the memory constraints of the MSCC11 board's 68'llE2 chip.

5. Sensors

5.1 Sensor Overview

The sensor array on Jormungand is somewhat limited. His current sensor array consists of two IR emitter/detector pairs, three bump sensors, and two tilt sensors. Due to time constriants I was unable to complete a one of my original objectives. I originally wanted Jormungand to have sonar “ears” to track a stationary or moving beacon, he would then “capture” the beacon by encircling it. The sensors I was able to complete, however, do provide sufficient input for interesting behaviors.

5.2 IR Sensors

The IR emitter/detector pairs are the standard hacked Sharp detector setup. They are the eyes for Jormungand. They can see approximately 2 feet and provide sufficient range for the snake to avoid obstacles. They are mounted on either side of the head looking forward and to the side. This arrangement provides for a good view to guide the robot by.

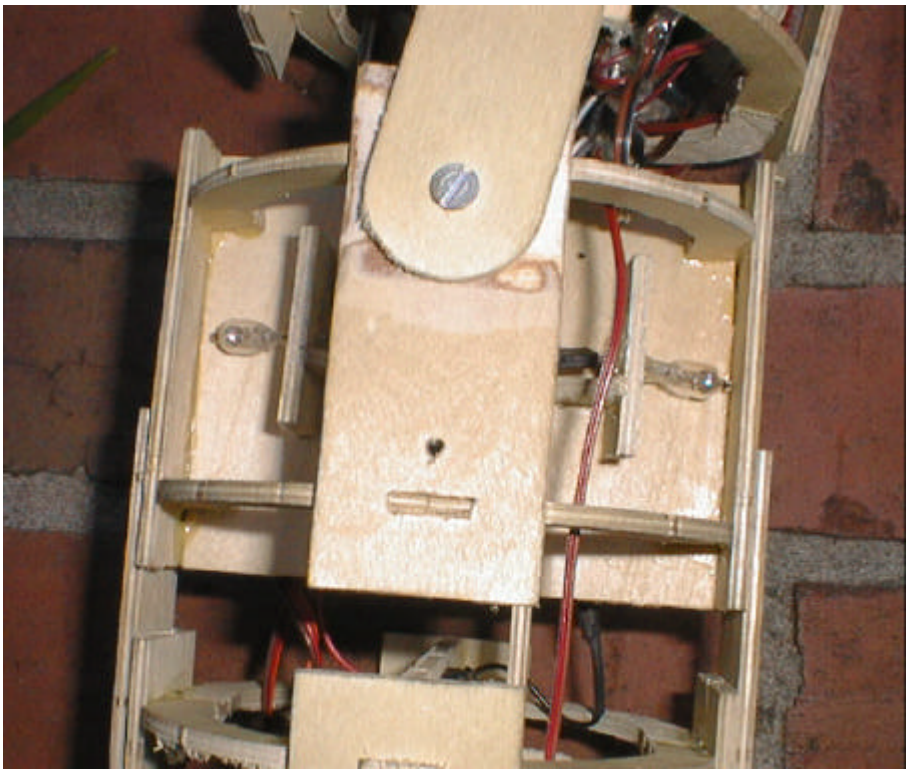
5.3 Bump Sensors

The bump switches are simple push buttons, they are the same as is used on the TJ and Talrik platform. The bump switches initiate two separate actions. The nose button causes the snake to become annoyed and rattle his tail. The nose button can be either held down by hand or activated if the snake moves straight into a wall because it was unable to see far enough forward. The cheek buttons cause the snake to initiate a sidewinder movement in the opposite direction. The bump switches were wired with a simple pull up scheme to provide the signal to the

processor.

5.4 Tilt Sensors

The tilt sensors are mercury switches. I obtained these from a member of the lab, Aamir. He had gotten them out of state a few years ago. Because of recent restrictions on the sale of items containing mercury he was not sure if it was possible to get them any more. They are mounted at an angle inside one of the segments. They are set up so it is possible to tell which side the snake has rolled over on. When the snake tilts over, the mercury in one of the switches slides over onto the two contacts and closes the circuit. The snake then takes appropriate action. The tilt switches were wired up exactly like the bump switches, with a simple pull up scheme.



Tilt Sensors

6. Behaviors

6.1 Behavior Overview

My goal for behaviors was to provide a good set of behaviors to make an interesting robot. The movement behaviors include an inchworm move as well as a sidewinder move. The additional behaviors are the self-calibration, obstacle avoidance, bump and the tilt recovery behavior.

6.2 Self-calibration

When the snake is turned on, the first action is a reading from the left IR sensors. If you place the snake's left eye a certain distance from a wall, that distance is as close to the wall it get before it wants to turn. If you leave the snake out in the open, then as soon it even detects a wall it will avoid it.

6.3 Obstacle Avoidance

The snake will avoid walls it sees with the IR sensors. As soon as the snake detects a wall it turns away. Then on the next check the snake will turn back if it no longer sees a wall, but at a rate 2/3rds that of the turning away. This minimizes the amount of jerking movement as the snake avoids obstacles. If there are walls on either side, the snake will try to maintain an equal distance from both walls.

6.4 Bump

When the nose button is pressed the snake will initiate its rattle. When a cheek button is pressed the snake will perform the sidewinder move.

6.5 Tilt Recovery

When the snake realizes it is not upright it initiates a series of moves to right itself. First it curls up a little bit and then uncurls quickly. This has the effect of tossing the robot around a little, and after a few tries generally gets the snake back upright.

7. Conclusion

7.1 Summary

Jormungand was all in all, a success. He accomplished my goal of a mobile, highly maneuverable robot. Jormungand is a rugged robot with room for expanded sensors and behaviors. His movements are more snake like than I had hoped and he is quite mobile. His one deficiency, I feel, is his limited range of sensors. With increased sensing ability I think he could perform more useful actions and exhibit much more complex behaviors. While, like everyone in the IMDL, I was unable to complete all I had hoped to do with the robot, I feel I have provided myself, and others who may wish to do so, a basis to build on.

7.2 Future Work

Replace brain board with TJ Pro board - I was extremely limited by the amount of program space available on the 68'11-E2's
Need Stronger Servos - Current 42 oz-in servos are cheap but Stronger servos means better moves
Bigger Batteries -Not very much run time with current internal batteries (6 AA's)

Enlarge diameter - Accommodate larger servos and C-cells
Shorten segments, increase number -More joints per foot = more flexible
More sensors - touch sensors to sense uneven ground, sonar for hunting behavior
Make the boards easier to get to- Tweezers and hemostats required for current setup

Appendix A - Code

Filename: INCHY13.C
This is the main brain-board program
Notes: Uses the TJ code for Analog (IR) ports as well as serial code from the MIL library

```
/*inchy 1 - added down force on servo 2 */
/*inchy 2 - turn all servos */
/*inchy 3 - include nose bumber on portc(1) */
/*inchy 4 - full inch mode, doit subroutine */
/*inchy 5 - initial wall distance, rattle for longer */
/*inchy 6 - experiment with taking out down force and smoothing movement */
/*inchy 7 - turn faster, turn back slower */
/*inchy 8 - include the tilt sensors, PC2=on left side, PC3=on right side */
/*inchy 9 - more aggresive approach to the righting procedure */
/*      9.a - changed turn back to equal turn rate */
/*inchy 10 - added eye sockets, taking out inital dist find */
/*inchy 11 - added left and right touch sensors PC4,PC5 */
/*inchy 12 - new flipping routine */
/*inchy 13 - new servo controller - servoc13.c
         - implements sidewinder movement using servo controller board
            call servo 99 with pw=1 for left, pw!=1 for right
            added sidewinder for left and right sensors*/

#define DELAY    20000  /* Delay between moves (in general) */
#define TURN     15     /* Turn rate away from wall */
#define TURNBACK 15     /* Turn back rate when no wall sensed */
#define EQUALVAL 5      /* left IR and right IR close enough to be considered
the same */


#define NEGMAX 1
#define HALFNEG 32
#define POSMAX 127
#define HALFPOS 96
#define HALF 64

#include <hc11.h>
#include <mil.h>
#include <irtj.h>
#include <analog.h>
#include <vectors.h>
#include <serial.h>


/* currpos holds the current position of the servos */
char cp[10];
int dist;  /* initial distance from wall */

/*prototype declerations */
void servoit(int sn, int pw);
void doservos(void);
void waitabit(int waittime);
doit(int waittime);
void zeroservos(void);
void rattle(int headturn);
```

```
void rightit(int tp1,int tp2);


int main(void)
{
 int i, lval,rval,diff, curr;
 init_analog();
 init_ir();
 init_serial();
 ir_mode(0);

 DDRC = 0x00;           /* Set PORTC (bumper) for input */

 waitabit(15000);       /* let things warm up */
 dist=ir_value(6);           /* get initial distance from wall */
 waitabit(15000);

 curr=0;
 diff=0;
 zeroservos();
 while(1)
 {
 /* see if the robot is on its side */
  if(PORTC & 0x04) rightit(86,100);
  else if(PORTC & 0x01) rightit(42,28);
/*check bumper switches */
  if(PORTC & 0x02) rattle(64);         /* nose touched */
  else if (PORTC & 0x10) {zeroservos(); servoit(99,2);} /* PC4, left sensor
touched */
  else if (PORTC & 0x20) {zeroservos(); servoit(99,1);}   /*PC5, right sensor
touched */
  else  /* normal movement pattern */
   {
  lval=ir_value(6); /* read IR*/
  rval=ir_value(7);
  diff=lval-rval;           /*check relative diff */
  if ((abs(diff)>=EQUALVAL) && ((lval>dist) || (rval>dist)) )
    {
     if( ((diff)>0) && (curr>(-64+TURN)) ) curr-=TURN;
     else if(curr<(64-TURN)) curr+=TURN;
    }
  else
    {
     if(curr<(-TURNBACK)) curr+=TURNBACK;
     else if (curr>TURNBACK) curr-=TURNBACK;
     else curr=0;
    }
  i=64+curr;
  cp[0]=i; cp[2]=i; cp[4]=i; cp[6]=i; cp[8]=i;
  doit(DELAY);
  cp[9]=32; cp[7]=1;  cp[5]=32;
  doit(DELAY);
  cp[9]=64;  cp[7]=96; cp[5]=127; cp[3]=96;
  doit(DELAY);
  cp[7]=64;  cp[5]=32; cp[3]=1;   cp[1]=32;
  doit(DELAY);
  cp[5]=64;  cp[3]=96; cp[1]=127;
```

```
  doit(DELAY);
  cp[3]=64;  cp[1]=64;
  doit(DELAY);
  }
 }
}

/*servoit() outputs the necessatry strings to tell the servo board to change
a servo postition */
/* CCW is postitive */
/* 64 is centered, 1 is ~45degrees all the way right,
   127 is all the way left   */
void servoit(int sn, int pw)
{
 put_char(0);
 put_char(sn);
 put_char(pw);
}
/*do all the servos*/
void doservos(void)
{
 int i;
 for (i=0;i<10;i++){servoit(i,cp[i]);}
}

/*delay routine*/
void waitabit(int waittime)
{
 int i;
 for (i=1;i<waittime;i++){}
 return;
}
/*do servos and wait*/
doit(int waittime)
{
 doservos();
 waitabit(waittime);
}
/* position all servos at zero (64) postition
void zeroservos(void)
{
 int i;
 for(i=0;i<=9;i++) {cp[i]=64;}
 doit(2*DELAY);
}
/*rattle routine*/
void rattle(int headturn)
{
int i;
cp[0]=headturn;
for(i=1;i<40;i++)  /*rattle if nose is touched */
     {
      cp[7]=96;
      cp[8]=69;
      doit(1000);
      cp[8]=59;
      doit(1000);
```

```
     }
}
/*sequence that helps the snake right itself*/
void rightit(int tp1,int tp2)
{
 cp[0]=tp1; cp[1]=96; cp[2]=tp1; cp[3]=32; cp[4]=tp1; cp[5]=96; cp[6]=tp1;
cp[7]=32; cp[8]=tp1;
 doit(DELAY);
 cp[0]=tp2; cp[1]=64; cp[2]=tp2; cp[3]=64; cp[4]=tp2; cp[5]=64; cp[6]=tp2;
cp[7]=64; cp[8]=tp2;
 doit(DELAY);
}
```

Filename: SERVOC13.C
This is the servo controller board code
Notes: This uses Drew Bagnell's Servo Control code XSR16.C

```
/* this is the servo controller used for inchy13.c */
#define DELAY 400
#define STEP 10
#define STEPS 100
#define HALFSTEP 5
#define MAX 900
#define HALF 450

#include "xsr16.h"
#include <serial.h>

void incher(int s1,int pw11,int pw12,int s2,int pw21,int pw22,int s3,int
pw31,int pw32,int s4,int pw41,int pw42,int s5,int pw51,int pw52,int s6,int
pw61,int pw62,int steps);

void
main(void)
{
   int actualpw,sn,pw,i,sideMAX,sideHALF;
   init_serial();
   init_servos();
   while(1)
    {
     sn=get_char();
     if (sn==0)
       {
        sn=get_char();
        pw=get_char();
        if(sn!=99)  /*servo number 99 means do sidewinder move*/
         {
          actualpw=2100+(14*pw);
          servo(sn,actualpw);
         }
        else /* had to put sidewinder move here, ran out of space on brain
board*/
         {
          if(pw==1) { sideMAX=MAX; sideHALF=HALF; }
           else {sideMAX=-MAX;sideHALF=-HALF;}
          incher(8,0,-sideMAX,6,0,sideMAX,-1,0,0,-1,0,0,-1,0,0,-1,0,0,STEPS);
          for(i=8;i>=2;i-=2)
            {

incher(i,-sideMAX,-sideHALF,i-1,0,HALF,i-3,0,MAX,i-5,0,HALF,i-2,sideMAX,sideH
ALF,i-4,0,sideHALF,STEPS);

incher(i,-sideHALF,0,i-1,HALF,0,i-3,MAX,0,i-5,HALF,0,i-2,sideHALF,-sideMAX,i-
4,sideHALF,sideMAX,STEPS);
            }
           init_servos();

/*incher(0,-sideMAX,0,-1,MAX,0,-1,0,0,-1,0,0,-1,0,0,-1,0,0,STEPS);*/
```

```
        }
      }
    }
}

void incher(int s1,int pw11,int pw12,int s2,int pw21,int pw22,int s3,int
pw31,int pw32,int s4,int pw41,int pw42,int s5,int pw51,int pw52,int s6,int
pw61,int pw62,int steps)
{
 int a,d,i,j,k,l,m,n,stepi,stepj,stepk,stepl,stepm,stepn;

 stepi=((pw12-pw11)/steps);
 stepj=((pw22-pw21)/steps);
 stepk=((pw32-pw31)/steps);
 stepl=((pw42-pw41)/steps);
 stepm=((pw52-pw51)/steps);
 stepn=((pw62-pw61)/steps);

 i=pw11; j=pw21; k=pw31; l=pw41; m=pw51; n=pw61;

 for(a=0;a<steps;a++)
  {
   i+=stepi;j+=stepj; k+=stepk; l+=stepl;m+=stepm;n+=stepn;

   if (s1>=0) {servo(s1,3000+i);}
   if (s2>=0) {servo(s2,3000+j);}
   if (s3>=0) {servo(s3,3000+k);}
   if (s4>=0) {servo(s4,3000+l);}
   if (s5>=0) {servo(s5,3000+m);}
   if (s6>=0) {servo(s6,3000+n);}
   for(d=0;d<=DELAY;d++){}
  }
}
```

Filename: XSR16.C
Drew Bagnell's servo code

```
//-- Copyright Drew Bagnell, 1998


//Experimental 16 servo motor driver with
//high precision control (within 2 degrees)


//One potentially useful improvement to make sure you only call servo changes
when nothing
//is happening to the servo... (only in the first 1000 e clocks after a
switch.) Note,
//the walk program does this by checking for a change in clock_counter, which
happens
//immediately after the ISR.
/***************************** Includes *********************************/
#include <hc11.h>
#include <mil.h>

/************************************************************************/

/************************** Prototypes **********************************/
#pragma interrupt_handler servo_handON
#pragma interrupt_handler servo_handB
#pragma interrupt_handler servo_handC

void init_servos(void);
void servo(int, int);
void servo_handB();
void servo_handC();
/************************************************************************/


/***************************** Globals **********************************/
unsigned int ontimeB[8];
unsigned int ontimeC[8];

unsigned char indexB,indexC;
unsigned char servomaskB[8],servomaskC[8];

int clock_counter;
/************************************************************************/


void
init_servos(void)
{

  int i;
  INTR_OFF();
  CLEAR_BIT(TCTL1,0x07);     //interrupt will affect no pins in port A

  for(i = 0 ; i < 8 ; i++)
  {
```

```
    ontimeB[i] = 3000;
    ontimeC[i] = 3000;

    servomaskB[i] =  1 << i;
    servomaskC[i] =  1 << i;
  }

  TOC1 = 0; //turn them all on at the rollover
  TOC2 = TOC3 = 3000; //start turning off at mid-position

  SET_BIT(TMSK1,0xE0);            /* Enable OC1,2,3 interrupt */

  DDRC = 0xff;

  clock_counter = 0;
  INTR_ON();
}

/*
   Applies or removes on from servo by turning the servomask
   to zero. In particular, if state= 0 , servo is turned off
   and state=1 turns it back on.
*/
void
servo_power(int offset, int state)
{

        if (offset < 8 )
        {
        servomaskB[offset] = state << offset;
        }
        else if (offset < 16 )
        {
        servomaskC[offset] = state << offset;
        }
}

void
servo(int offset, int newwidth)
{
  int i;
  int deltaWidth;

  if ( offset < 8 )
  {
  deltaWidth = newwidth - ontimeB[offset];
  ontimeB[offset] += deltaWidth;
  }
  else if ( offset < 16 )
  {
  deltaWidth = newwidth - ontimeC[offset-8];
  ontimeC[(offset)-8] += deltaWidth;
  }


}
```

```
void
servo_handON()
{
        PORTC = *(servomaskC+indexC);
        PORTB = *(servomaskB+indexB);
        TOC1 += 5000;
        CLEAR_FLAG(TFLG1,0x80);          /* Clear OC1 flag */
        indexB = indexC = (indexC + 1)&0x07;
        clock_counter++;
}



void
servo_handB()
{
    PORTB = 0;
    CLEAR_FLAG(TFLG1,0x40);          /* Clear OC2 flag */
    INTR_ON();
    TOC2 += *(ontimeB+indexB) + (TOC1 - TOC2);
}

void
servo_handC()
{
    PORTC = 0;
    CLEAR_FLAG(TFLG1,0x20);          /* Clear OC3 flag */
    INTR_ON();
    TOC3 += *(ontimeC+indexC) + (TOC1 - TOC3);
}
```

Filename: XSR16.H
Header file for XSR16.C

```
/*********************************************************
* Header File for 16 servo controllers
* Programmer: Drew Bagnell
* copyright Drew Bagnell 1998
*
*********************************************************/

#ifndef _XSR16_H
#define _XSR16_H


/*******Function Prototypes******************/
/* These handlers are visible only to provide access to vectors.c*/

void servo_handON(void);
void servo_handB(void);
void servo_handC(void);


void servo_power(int offset, int state);
void init_servo(void);
void servo(int offset, unsigned newwidth);


extern int clock_counter;
#endif
```

Appendix B – Description of CAD and Plot files

*note - layout implies that the pieces are laid out and ready for plotting

most of the time the original un-offset pieces are on different layers

that have been frozen
All drawings are in AutoCAD R14

arms.dwg - copy of a couple dozen replacement arms
arms.plt - plotfile of same

base.dwg - drawing of the base pieces
baselay.dwg - layout of two different lengths of base pieces
baselay.plt - plotfile of same

newsnake.dwg - drawing of the pieces of the original (smaller) segment
cutlay.dwg - layout of original (smaller) segment
cutlay.plt - plotfile of same
cutlay1.dwg - has both offset and original sized pieces for the segments
cutlay2.dwg - layout of two segments
cutlay2.plt - plotfile of same

eyesock.dwg - drawings of the eye sockets
eyelay.dwg - layout of two eye sockets
eyelay.plt - plotfile of same

snakhead.dwg - drawing of the head pieces of the snake
headlay.dwg - layout of a single head piece, also pieces that complete the processor segment
headlay.plt - plotfile of same

snakproc.dwg - drawings of the processor segment
layproc14.dwg - layout of two processor segments
uses the spare pieces from layhead.plt to complete both segments
taillay.dwg - layout of tail piece (with extra base piece)
taillay.plt - plotfile of same

Appendix C – Construction Notes

First, I have left two segments assembled and connected on the robot shelf in the lab (the one with all the old robots on it.) Look at it, it will help you assemble it properly.

For normal segments –
Cut out CUTLAY2.PLT on the t-tech - this has all the pieces you need for two segments.
There should be:
2 - Main plates (with holes for servos)
8 - semicircular rib pieces (note - four have extra long tabs in the center, separate these out)
8 - base pieces (rectangular pieces that hold everything together)
2 – servo arms
4 – four hinge arms

On main plates and the segments, servo holea goes to the front,
On the base pieces, the larger holes go towards the front.
The rib pieces with the longer tabs go towards the back.

Nothing fits together too well at first so have a razor knife handy.
- Widen the slots in the base pieces with a razor knife or dremel tool.
- Knock the corners off the rib pieces so they fit in the slot in the base pieces. When I assemble them I get them close enough and then take a pair of needle nose pliers and force them together.
- Widen the slots on the sides of the base pieces and their matching slots on the rib pieces
- Fit the rib pieces into the slots on the main plate. (make sure the two with short tabs are to the front and the two with long tabs are to the back)
-Fit the base pieces that are perpendicular to the main plate. (make sure these are flush with the side of the plate, if they aren't, trim out the tabs or the slots or both.
-After the side pieces are flush fit the base pieces on the top and bottom, this ties together the side rib pieces.
-After making sure everything is oriented properly, run superglue along all the joints to glue it together
-now, fit the hinge arms over the long tabs, they should stick backwards. Glue this.
-The servo arms fit directly over the knob on the servo, no servo horn is needed. You will have to widen the hole on the servo arm. Don't glue this, it isn't needed and may foul the servo. These arms do wear out (especially if the fit isn't particularly tight) that is why I drew ARMS.PLT, a plot file with extra arms.

Processor Segments and head-
Cut out LAYPROC14.PLT and HEADLAY.PLT
This provides you all the pieces for the head, two processor segments with a couple extra pieces.
- Assembly is almost the same except the processor boards must be mounted before the segment is fully assembled.
- The single-chip boards attach to the smaller square piece and then slip in perpendicular to the main plate.
- Assembly requires a lot of shaping, the slots for the processor board are not quite right and the holes don't line up too welll.

The tail is in TAILLAY.PLT and assembles similarly to the rest.

Also, feel free to track me down and ask me for help!! - Billy