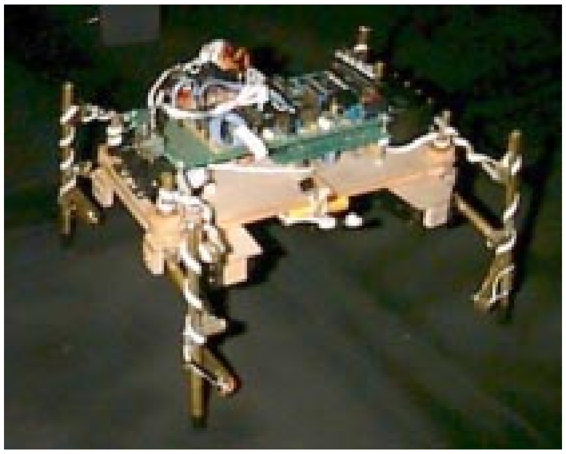
Intelligent Machines Design Laboratory
Dr. Keith L. Doty, director.
EEL 5666

SPOT

An autonomous mobile platform



Rodrigo J. Osorio

December 12, 1997

Table of Contents

Abstract

Spot is a mobile platform with four legs for autonomous agents. Each leg has two degrees of freedom and two sensors. Walking is achieved by maintaining static balance. Static balance is maintained by pivoting the body of the platform. Legs are controlled with two Motorola microcontrollers communicating through the serial port.

Executive Summary

Spot is an attempt to create a platform for autonomous agents capable of traveling over various surfaces and obstacles. Mobility is achieved with four legs, each actuated by two servos. In order to walk static balance must be maintained. This is done by tilting body from side to side. Legs each have a pressure sensor to permit control static balance. Servos are driven by an M68HC11E2, which in turn is controlled by an M68HC11E9 with expanded memory.

Due to friction, pressure sensors did not perform adequately. As a result, the platform was only able to walk with hard-coded balance control. Infrared obstacle avoidance was implemented to demonstrate walking abilities.

1. Introduction

Mobile platforms for autonomous agents are usually based on wheeled designs, and sometimes on legged designs. There are many reasons for a preference toward wheeled platforms. Foremost probably is the ease of construction and control, thereby permitting the experimenter to devote the majority of his time to behavior development. Most wheeled autonomous agents consist of nothing more than a collection of processors and sensors that have been joined to a commercially available geared motor. In its simplest incarnation, the wheeled implementation is merely a modified servomotor controlled with Pulse Wave Modulation (PWM). Other benefits that cannot be ignored are adequate travel speed, and greater energy efficiency and reliability than other translation methods.

Wheeled platforms do have some limitations. They usually must remain within the smooth and level environment of the laboratory floor, since the wheels cannot forge any but the most trivial obstacles. This limitation is usually of little concern, since there is much research that can be performed on the laboratory floor. Nevertheless, there is much to be learned about autonomous agents and machine intelligence by designing platforms if greater mobility than that of wheeled platforms.

There have been some attempts to increase mobility by constructing legged autonomous agents. These platforms to some degree and with varying success seek to reproduce the mobility of biological organisms. Leg implementation introduces complexities and problems that cannot be ignored. Mechanically, legs are more difficult to construct and are unquestionably less reliable than wheels. Legs are also less efficient and much slower over level surfaces than wheels. Legs, on the other hand, are particularly adept at traversing uneven terrain and at forging obstacles.

The complexity of construction and control of legged platforms does not necessarily depend on the number of legs used. The legged spectrum ranges from one-legged platforms to platforms that have six or more legs. The complexity spectrum, however, is not congruent with the leg-number spectrum. For example, the active balance required by an one-legged platform necessitates high-speed actuators and complex sensors and control algorithms.   Likewise, platforms with six or more legs are complex because of the total number of legs, but their inherent stability obviates the need for active balance control.

Agents with four legs present an interesting balance between mechanical complexity and control complexity. Mechanically, four legs are more reliable and more easily constructed than six or more legs. In practice, although the inherent stability possessed by six-legged agents is lost, the active balance of one- and two-legged agents is not absolutely required. Essentially, if the limitation that at any particular time at least three legs must be on the ground, balance control consists of nothing more than ensuring the center of gravity of the agent is within the support tripod of the legs.

This is an experiment into the viability of such a design. Without complex sensors, actuators, or control algorithms, a four-legged platform should be able to move robustly through terrain and over obstacles that would immobilize wheeled platforms.

2. Integrated System

Spot is an autonomous agent platform supported and propelled by four legs. The legs are actuated by servomotors, which in turn are controlled with two Motorola MC68HC11 microprocessors. Legs are constructed of hollow brass tubes, while the body is made of lightweight plywood.

Each of the legs is actuated by two servomotors to effect rotations in the z-axis as well another axis perpendicular to the z-axis (see fig.1 and fig. 2). Each leg, therefore, has two degrees of freedom. In total, the platform uses eight servos. Within each leg and protruding on the lower side of each is a pressure sensor to measure the amount of weight resting on each leg. The sensor consists of a spring-loaded plunger connected to a potentiometer (see fig. 4). Using a voltage divider, the potentiometer's position is digitized for use in the processor.

Walking gait is hard-coded and ensures that at any time only one leg is off the ground. In order to maintain static stability, before a leg is raised the center of gravity of the body must be within the support perimeter of the remaining three legs. The range of rotation of each leg on the z-axis is divided into four positions. At any one time each leg is assigned a different position. To move, the three legs supporting the body advance one position, while the remaining leg advances four positions. This leg is subsequently lowered, another leg is raised, and the cycle continues.

To control balance and permit the platform to raise one leg while remaining statically stable, the center of gravity of the body must remain over the perimeter of the triangle defined by the contact points of the supporting legs. The center of gravity of the body, which is above the axis of rotation of the body, is shifted with respect to the legs by tilting the body sideways. This enables balance control without using additional components or weights. Although weight shifting is currently hard-coded, the ultimate goal is to control balance with the pressure sensors.

Two microcontrollers are used to drive the legs and control the platform. The PWM waves for the eight servos are generated by a single M6811E2 operating in single-chip mode. This microcontroller receives signals through the serial port from an M6811E9 with additional RAM. All sensory inputs and behavior logic reside in the E9 chip.

3. Actuation

Standard Radio Control servos actuate all movements on the platform. These components were chosen as actuators because of their ease of integration, their ease of control, their power output, and their low cost. Being self-contained, gearhead motors, mechanical integration usually consists on properly fixing the servos to the platform. This greatly reduces design time and construction effort. In addition, the length of a PWM wave defines servo position. These waves are easily generated with a microcontroller with timer functions, and obviate the need for leg-position encoders. Finally, with torque outputs ranging from 37-in-oz. to 42 in.-oz., the servos are powerful enough to actuate small and lightweight platforms.

Eight distinct PWM waves are generated by the single-chip MCU to control all leg movement. All waves depend on only one output-compare register. To do so, each servo is assigned a 5000 cycle segment of a 40,000-cycle routine. In succession, the rising edge for each servo starts in 5000 cycle intervals. In this manner, each servo receives a signal every 40,000 cycles, or every 20 milliseconds. Upon start of the rising-edge, pulse duration is scheduled according to desired position for the servo. Although the MCU is constantly minding the PWM's, the interrupt-driven nature of the program reduces the required processing time and ensures accurate wave signals for the servos.

The single-chip MCU receives leg commands from another MCU through the serial SPI port. No framing data is used to speed communications and to reduce processor overhead. Instead, position for each servo is communicated by assigning a 32 level integer and sending this integer through the SPI port. That is, integers from zero to 31 refer to servo number one; while integers from 32 to 63 refer to servo number two. This scheme is likewise repeated for the remaining servos. In this manner, any number between zero and 255 sent through the serial port refers to a position command for a specific servo.

The leg control commands sent through the serial port are generated by the expanded MCU, which generates movement commands from predefined walk tables.

4. Sensors

Sensors on each leg are necessary to control walking behavior and to provide "walking reflexes" from which more complex behaviors may emerge. Minimally, these sensors must measure pressure exerted on the ground by each leg, and must detect contact with obstacles by prominent points on the side of the leg.

Pressure sensors for each leg consisted of a spring-loaded plunger connected to a potentiometer. A voltage divider circuit produces voltages that are digitized by the MCU. Plungers were machined from aluminum, and the potentiometers were removed from servos.

Side-contact sensors were not constructed because pressure sensors did not work as expected. The combination of the aluminum plungers, springs, and potentiometers with their associated linkages introduced too much friction into the operation of each sensor. As a result, the sensors did not have the sensitivity required to detect slight movements in the relative position of the body's center of gravity. The parallelogram leg geometry and the necessity to tilt the body to shift the center of gravity aggravated this situation. Tilting the body to shift the center of gravity also tilts the legs, thereby decreasing the force perpendicular to the spring, and increasing friction by increasing side loads on the plunger.

5. Behaviors

Initially, the primary low-level behavior envisioned was to maintain a predefined force on each leg. From this ability it was hoped to derive the ability to maintain the center of gravity within any three legs. Combining the control of static balance with obstacle detection on each leg, higher behaviors are easily envisioned. The platform could have walked along terrain of varying grade while maintaining balance. In addition, with obstacle detection on each leg, the leading leg could determine if obstacles could be climbed or if it was necessary to walk around.

Unfortunately and as previously stated, the pressure sensors did not work. Without them it is not possible to control balance while climbing obstacles.

To demonstrate the capability of the leg design, open loop walking over smooth and level surfaces was achieved using walk tables. To do so, the platform sifts its weight on to three legs and lifts up the remaining leg. As the legs on the ground advance through ¼ of their total travel, the leg in the air advances fully. At the end of its travel, this leg is lowered and the process is repeated with another leg.

Using this scheme it was possible to walk straight both forwards and backwards, to turn while walking, and to turn in place. Walking and turning was accomplished by advancing the legs on one side less than the legs on the other side. Turning in place was accomplished by advancing the legs on one side while retarding the legs on the other side.

In order to demonstrate the walking abilities, a simple infrared sensor/emitter combination was installed on the front of the platform. Based on the data received from the sensor, the platform retreats and changes direction.

6. Conclusion

I believe that in its current incarnation the device I have constructed does not qualify as an autonomous agent. Although a sensor controls walking direction, the walking behavior is fixed and independent of the environment. The fact that it walks depends on a carefully synchronized sequence of movements and on a relatively level surface. Although I am satisfied with the mechanical accomplishments, the device lacks the “walking reflexes” I highly desired to develop. Nevertheless, I still have hope of obtaining suitable pressure transducers that will permit the control of static balance.

In retrospect, I believe a superior leg design would poses three degrees of freedom and three actuation servos. Such a leg can be controlled to move its contact point linearly, something the legs on my current design cannot do. Furthermore, although it is apparent that a four-legged platform is feasible, a six-legged platform is more stable and highly more desirable in spite of the added complexity.

Appendix A

```
/*      Servo Control program for SPOT                              */
/*      written for ICC and compiled for single chip board          */
/*      NOTE:  6811E2 chip running in BOOTSTRAP mode                 */
/*      program receives data through serial port and generates PWM's*/
/*      Written by:  Rodrigo J. Osorio for EEL5666                   */
/*      Last revision:  December 10, 1997                            */
/*                                                                   */


#include<hc11.h>
#include<mil.h>

#define TOC2_ISR servo_driver
#define SCI_ISR serial_driver
#pragma interrupt_handler servo_driver
#pragma interrupt_handler serial_driver

void servo_driver();
void serial_driver();
void init_serial();
void init_servos();
void init_board();

static int state,i;

int mask[8]={0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};
int data[8];

main(void){
  init_board();
  INTR_OFF();
  init_servos();                  // Initialize servos & start servo
interrupts
  init_serial();                  // Initialize serial ports & start
interrupts
  INTR_ON();

  while(1);

}

void servo_driver()
{
      if(state)               // Turn-off port
      {
        PORTC=0x00;
        PORTB=0x00;
        TOC2=TCNT+5000;       // Progran next rising-edge
        state=0;
      }
      else
      {
        PORTC=mask[i];        //
        PORTB=mask[i];        //  Turn-on port
        TOC2=TCNT+data[i];    //  Progran pulse width
        state=1;
        i++;                //  i indexes servo widths and port masks
        i%=8;               //  i must be between zero and seven
      }
```

```
    CLEAR_FLAG(TFLG1,0x40);   //  Clear TOC2 flag
}

void serial_driver(){
   int com_data;
   com_data=SCDR;
   if(com_data<32) data[0]=-com_data*63+3874;          //4000
   else if(com_data<64) data[4]=(com_data-32)*63+2000;

   else if(com_data<96) data[1]=(com_data-64)*63+2126;  //2000
   else if(com_data<128) data[5]=-(com_data-96)*63+4000;

   else if(com_data<160) data[2]=-(com_data-128)*63+4000;
   else if(com_data<192) data[6]=-(com_data-160)*63+4000;

   else if(com_data<224) data[3]=(com_data-192)*63+2000;
   else  data[7]=(com_data-224)*63+2000;
   CLEAR_FLAG(SCSR,0x20);       // clear receive-data-register-full
flag
}

void init_serial(){
      BAUD=0xb0;                   // Baud=9600
   SCCR2=0x2c;                  // Enable Transmit and Receive
                                // Enable interrupts on receive-full
}

void init_board(){
    CLEAR_BIT(HPRIO,0x40);          //Remap interrupts to 0xffd6
    DDRC=0xff;                      //  Configure PORTC for output
}

void init_servos(){
  TOC2 = 0;                       // First OC2 int will occur at TCNT=0
*/
  SET_BIT(TMSK1,0x40);            // Enable OC2 interrupt
  data[0]=3000;data[1]=3000;data[2]=3000;data[3]=3000;
  data[4]=3000;data[5]=3000;data[6]=3000;data[7]=3000;
  state=0;

}


/*      Leg Control program for SPOT                                */
/*      written for Interactive C                                   */
/*      program communicates with single-chip board using serial port*/
/*      Written by:  Rodrigo J. Osorio for EEL5666                  */
/*      Last revision:  December 10, 1997                           */
/*                                                                  */


int px[4];    /* these arrays keep track of servo possition */
int pz[4];
int hold_flag;
int k,l,m,n;  /* these vars. keep track of location on walk table */
/*   Walk Table      v[] is leg roll
                                      h[] is leg yaw
```

```
                     h5[] is yaw cycling backwards
*/
int v[20] ={10,10,10,21,21,21,21,10,10,10,10,10,10, 3,19,19,
3,10,10,10};
int h[20]
={18,18,18,18,18,14,14,14,14,14,10,10,10,10,10,22,22,22,22,22};
int
h5[20]={14,14,14,14,14,18,18,18,18,18,22,22,22,22,22,10,10,10,10,10};
int s[20] ={ 2,4,4,4,4, 2,4,4,4,4, 2,4,4,4, 2, 2, 2,4,4,4};


int main(void){

int i;
int ir=0;
int ir_left=0;
int ir_right=0;

   init_serial();
   center_legs();
   pause(1000);

   while(1){
      if(k==2 || k == 7 || k == 12 || k == 17) ir=analog(1);
      if(ir<95) forward();
      if(ir>=95) {
         if(ir>110) backward();
         else{
                  for(i=0;i<=25;i++) turn_left();
                        ir=0;
                  ir_left=analog(1);
                  for(i=0;i<=50;i++) turn_right();
                  ir_right=analog(1);
                  if(ir_left<ir_right)    for(i=0;i<=100;i++)
turn_right();
                  else for(i=0;i<=100;i++) turn_left();
         }
      }
   }

return 0;
}

/* launch processes to move specific leg from current to target
   number is leg number (1-4)
   z      is leg height
   x      is leg rotation
   when received, numbers between  0 and 31 are decoded for leg 1 roll,
                  numbers between 32 and 63 are decoded for leg 1 yaw,
                  numbers between 64 and 95 are decoded for leg 2 roll,
etc.
*/
void move_leg(int number, int z, int x, int s){
   int i;
   i=(number-1)*64;
   start_process(move_servo(px[number-1]+i   ,x+i   ,s)); /* launch
roll */
```

```
   start_process(move_servo(pz[number-1]+i+32,z+i+32,s)); /* launch yaw
*/
   px[number-1]=x;
   pz[number-1]=z;
}

/* process to move servo at specified speed from current to target
   these are run as simultaneous tasks, up to eight at a time
   p is present servo possition
   x is target servo possition
   s is inverse of speed (delay) */
void move_servo(int p,int x,int s){

   int i;
   hold_flag++;                         /* hold flag keeps track of
number of

       processes running */
   if(p<x){
      for(i=p;i<=x;i++){
            put_char(i);                   /* send command to serial
port */
            pause(s);
      }
   }
   else if(p>x){
       for(i=p;i>=x;i--){
            put_char(i);
         pause(s);
       }
   }
   hold_flag--;
   defer();
}
                             /* center legs and initialize position
vector */
void center_legs(){
      put_char(15); put_char(47);
   put_char(79); put_char(111);
   put_char(143);put_char(175);
   put_char(207);put_char(239);
   px[0]=15; pz[0]=15;
   px[1]=15; pz[1]=15;
   px[2]=15; pz[2]=15;
   px[3]=15; pz[3]=15;
   hold_flag=0;
   k=0;l=5;m=10;n=15;

}

void pause(int a){           /* todo: replace pause with some timer
function */
   int b;
      for(b=0;b<=a;b++);
}
```

```
void forward(){
            k++;l++;m++;n++;            /* advance walk table indices
*/
      if(k>19) k=0;
      if(l>19) l=0;
      if(m>19) m=0;
      if(n>19) n=0;
      move_leg(1,h[k],v[k],s[k]);  /* move leg 1 to next position on
table */
      move_leg(2,h[l],v[l],s[l]);
      move_leg(3,h[m],v[m],s[m]);
      move_leg(4,h[n],v[n],s[n]);
      while(hold_flag >0);   /* wait until all processes launched have
ended */
}

void turn_left(){
            k++;l++;m++;n++;
      if(k>19) k=0;
      if(l>19) l=0;
      if(m>19) m=0;
      if(n>19) n=0;
      move_leg(1,h5[k],v[k],s[k]); /* h5[] cycles legs backwards   */
      move_leg(2,h[l],v[l],s[l]);  /*       leg 1 ------- leg 2    */
      move_leg(3,h[m],v[m],s[m]);  /*             |front|          */
      move_leg(4,h5[n],v[n],s[n]); /*             |     |          */
      while(hold_flag >0);         /*             |back |          */
}                                  /*       leg 3 ------- leg 4    */

void turn_right(){
            k++;l++;m++;n++;
      if(k>19) k=0;
      if(l>19) l=0;
      if(m>19) m=0;
      if(n>19) n=0;
      move_leg(1,h[k],v[k],s[k]);
      move_leg(2,h5[l],v[l],s[l]);
      move_leg(3,h5[m],v[m],s[m]);
      move_leg(4,h[n],v[n],s[n]);
      while(hold_flag >0);
}

void backward(){
            k++;l++;m++;n++;
      if(k>19) k=0;
      if(l>19) l=0;
      if(m>19) m=0;
      if(n>19) n=0;
      move_leg(1,h5[k],v[k],s[k]);
      move_leg(2,h5[l],v[l],s[l]);
      move_leg(3,h5[m],v[m],s[m]);
      move_leg(4,h5[n],v[n],s[n]);
      while(hold_flag >0);
}
```

Appendix B

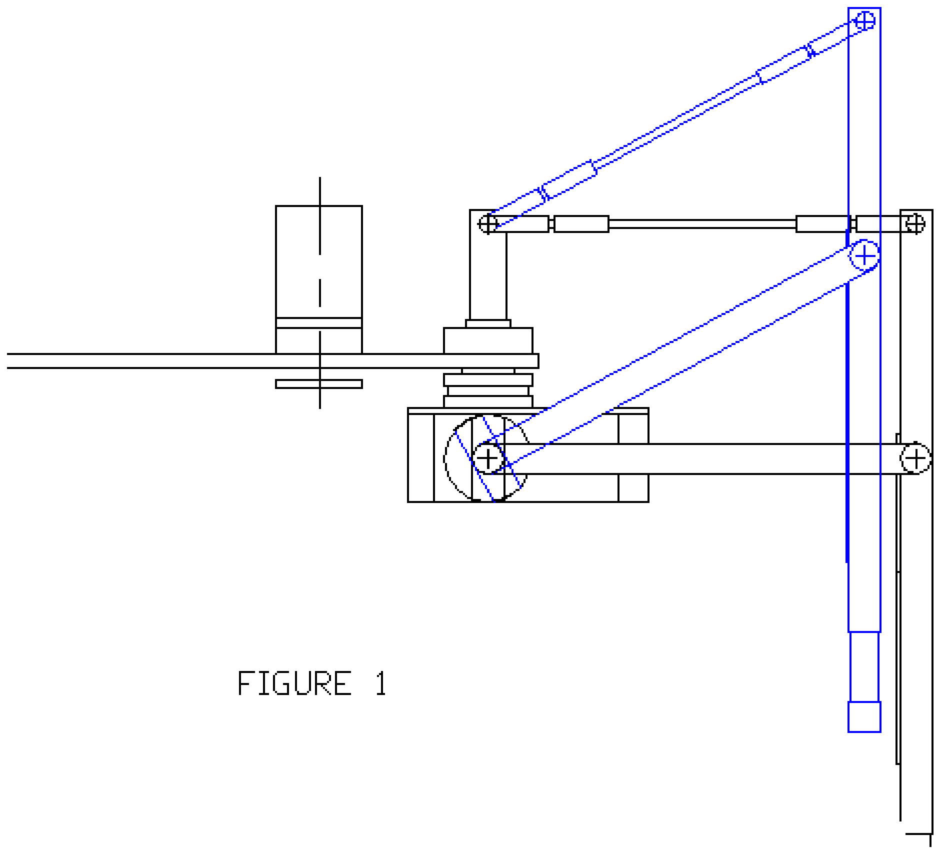


FIGURE 1

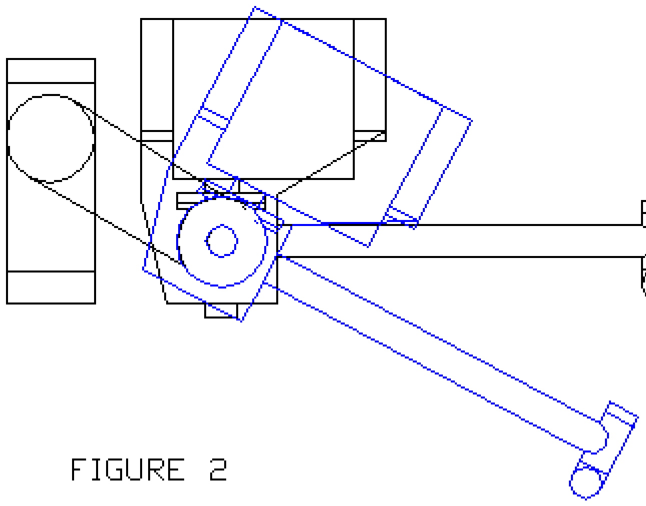


FIGURE 2

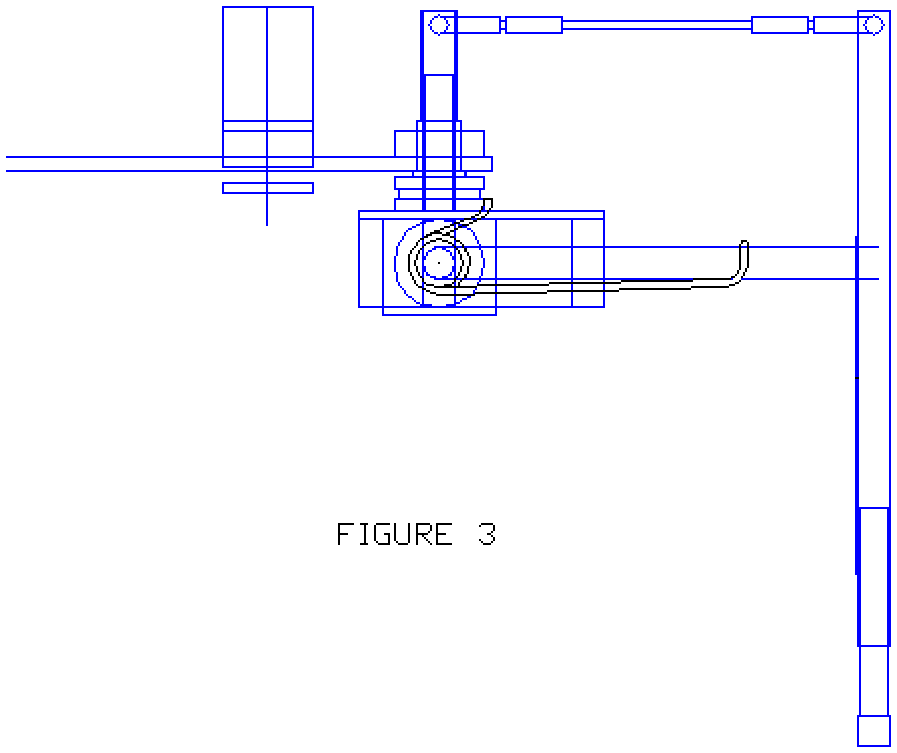


FIGURE 3

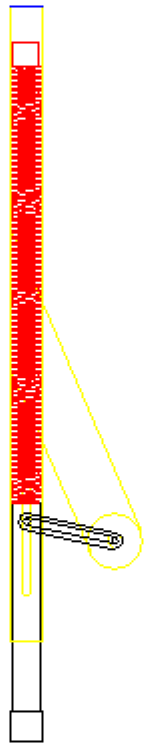


FIGURE 4

Appendix C

Hardware

- Actuators
  - four NES-L501 servos from JR Remote Controls of Japan
  - four Titan Shorty servos from Royal
- Controllers
  - Motorola M68HC11EVBU microcontroller board
  - Novasoft ME11U memory expansion board http://www.mil.ufl.edu/novasoft/
  - BOTBoard single-chip 6811 board http://www.rdrop.com/users/marvin/
  - M68HC11E2 chip
- Materials (available at most hobby stores)
  - 1/8” aircraft grade plywood
  - K&S brass tubing of various outer diameters
- Sensors
  - Potentiometers removed from surplus servos
  - Sharp 40 KHz Digital Infared Receiver http://www.mil.ufl.edu/novasoft/

Software

- Interactive-C 3.1 http://www.mil.ufl.edu/novasoft/
- ICC11 C-Compiler for the MC68HC11 http://www.mil.ufl.edu/novasoft/
- PCBUG11