DOGBOT

EEL5666 Intelligent Machines Design Laboratory

Andrew Reddish

T A B L E O F C O N T E N T S

A B S T R A C T

This IMDL project was to build and program a four-legged walking robot. The robot's primary objective is to achieve four-legged locomotion with obstacle avoidance. The robot's gait was modeled after four-legged animals. One MOTOROLA M68HC11 EVBU board with some extra de-multiplexing hardware for servo servicing controls the robot. The legs exhibit two-degree of freedom motion and are actuated by three servos.

E X E C U T I V E S U M M A R Y

Dogbot is a four-legged autonomous walking agent. It was designed and built as my IMDL project. The platform was constructed out of 5-ply wood cut out using the T-Tech machine and controlled with one MC68HC11 EVBU board. It uses four IR detectors and emitters for obstacle avoidance and bump sensors for detecting contact between leg and ground. The goal of the project is to create a four-legged walking robot capable of avoiding obstacles.

The legs are actuated by 12 servos total. Two of which move the leg up and down for support and lift. Four of the servos function as hip joints that rotate the shoulders that the legs are mounted. These 12 servos are controlled by de-multiplexing the OC2-OC5 lines of the EVBU board. The servos are supplied with a 7.2 radio controlled car battery.

Walking is achieved by moving opposite legs as a pair. A pair is lifted then rotated forward as the other is lowered and rotated backwards. This propels the robot forward by shifting the weight. This process is repeated over and over to allow for continuous forward movement.

I N T R O D U C T I O N

Legged locomotion is a more superior form of mobility when different terrain must be navigated with no intervention or re-programming. This is demonstrated by the fact that frictional changes and uneven spots cause wheeled locomotion to fail, or behave incorrectly. For instance, a two-wheel drive automobile is incapable of driving through a bed of rocks with large drops and rises. Thus it was the purpose of this project to develop a legged autonomous robot capable of walking.

D E S I G N C O N S I D E R A T I O N S

In designing a four-legged walking robot, many factors were taken into consideration. Of these factors only two presented a significant mechanical challenge: balance and propulsion. For any autonomous walking agent to be able to walk it must first stand. Standing requires that the robot is statically stable. This is achieved easily with four legs positioned around a rectangle as I have done in the design of this robot.

However, the robot must also maintain balance while it moves the legs while walking. Since the algorithm for walking dictates that I move two legs at a time there are only two legs on the ground at any given time. This creates a significant challenge since the robot will almost always fall to one side when only two legs are supporting it. This fact is somewhat offset by the rotation of the legs in the air, but still remains a small problem. It was found during testing of the robot that the falling to one side did not happen if the

robot's legs were moved at a fast enough rates. But, when it did happen, it only effected the ability of the robot to go straight, since the side to which it fell came in contact with the ground more and created a small pivot point. Therefor this type of balancing was not accounted for and would be a point of extension to this project later on.

P L A T F O R M D E S I G N

The platform design was modeled after a small dog. The body was rectangular (14"X6") with legs mounted under the corners of the body on a shoulder support. A shoulder support was created to give the legs a mounting point that could be pivoted by another servo attached to the top of the support. The body and shoulder supports are shown in figure 1.

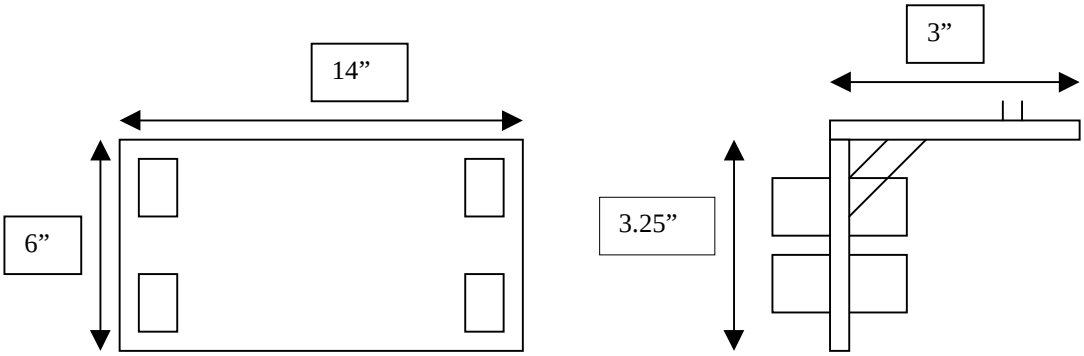

Figure 1.

The leg design was the most critical part of the robot after it's body and shoulders were designed. The legs had to support the weight of the robot and be able to move it forward. Initially the legs were constructed from 1" PVC pipe that was 8" long. However, due to construction problems in creating a mounting point for the linkages that connect the leg to the shoulder, this design was replaced with wood. The wood legs were cut out using the T-Tech machine from an AutoCAD drawing of the leg. An assembled leg and leg

design specifications are shown in figure 2. The legs are 6” high and 5/8” thick with mounting points for the linkage 1.75” apart. The bottom of the leg is capped with rubber, which is glued on, then taped to keep it secure. The rubber forms a small pad that helps the robot to walk on smooth surfaces where it would otherwise slip. Under the rubber is a bump sensor used to detect when the foot is on the ground.

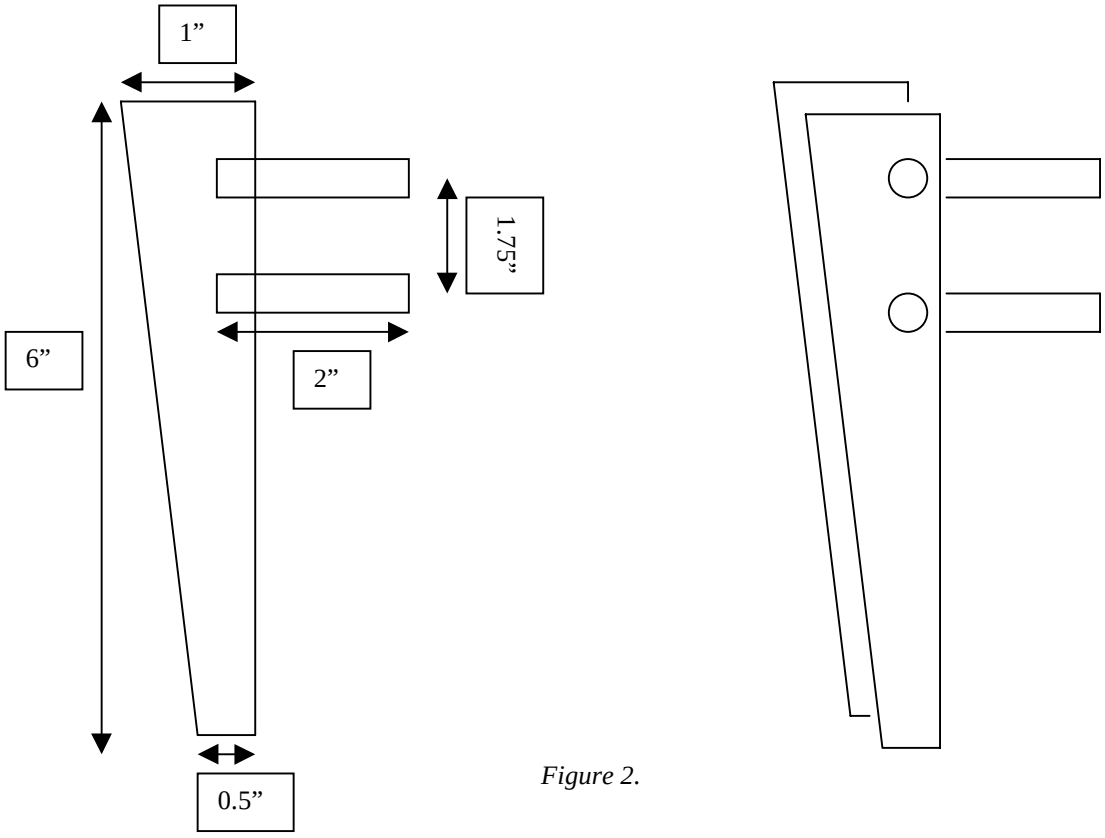

Figure 2.

The circuitry that controls the robot and servos was mounted underneath the body. The battery pack for the servos was mounted in the center, on top of the body. In front of the servo battery pack the EVBU board battery was mounted. Control switches, buttons, and LED indicators were mounted between the rear servos and the servo battery pack.

A C T U A T I O N

To move the robot a model of walking similar to that used by four-legged animals was used. The algorithm to walk consists of moving opposite legs forward and up, while the opposite pair moves backward and down. The first set of legs is lifted while the second is pushing down to give the set of legs up enough space and time to move to their new positions. While in this configuration, the set of legs off the ground rotate forward while the set on the ground rotates backwards. The rotation backward of the legs in contact with the ground causes the robot to fall forward and move in that direction. Hence creating forward propulsion and shifting the weight back onto the newly positioned legs. This process is repeated over and over by changing which pair of legs is rotating in which direction. A flowchart for this procedure is shown in diagram 1.

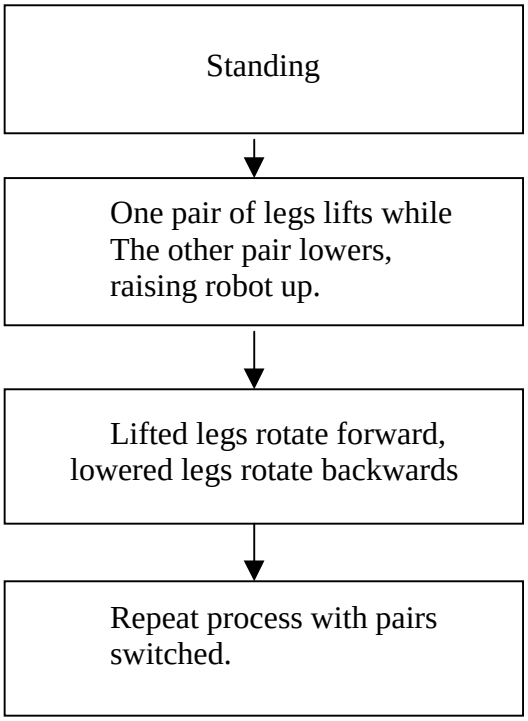

Diagram 1.

This was achieved using 12 servos. Four were used as the hip joint that rotated the shoulder and with it the legs. Two servos each were mounted on the shoulders and were used to lift the robot up and support the weight. The servos used were manufactured by Aristo and capable of supplying 42 oz-in of torque. The servos were controlled by de-multiplexing the output compare lines of the MC68HC11 EVBU board. The circuitry block diagram for the de-multiplexing hardware is shown in figure 3.

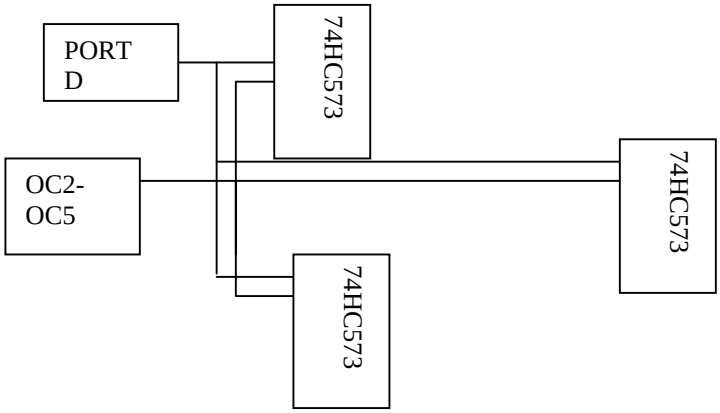

Figure 3.

An algorithm for controlling the servos was developed to use all five of the output compare lines. Since the first output compare line was capable of controlling all five, it was used as a timer and a master set. The other four were used only bring the output low at the appropriate time. Interrupts were used to allow for processing of the sensors and other calculations to be performed without having to service the servos directly. Output compare 1 (OC1) was set to interrupt three times during the 20ms period of the signal generated for the servos. OC1 would interrupt at times 0ms, 2.5ms, and 5ms of the 20ms waveform. At each of these instances it would select the correct set of motors through port D of the processor, set all of the OC2-OC5 lines high and enable their interrupts, and place the correct shut-off time in the OC2-5 compare registers. After servicing the last set of motors, it would set itself to interrupt 12.5ms later so that it could repeat the

process. This algorithm would allow for up to 32 motors to be serviced, since you could update five more sets of motors in the 12.5 ms time.

S E N S O R S

The sensor system for Dogbot was designed to achieve obstacle avoidance and sense of ground detection. Infrared emitters and detectors were used to achieve obstacle avoidance. There were four pairs of the emitters and detectors positioned on the body of the robot. Three were positioned in the front of the robot to detect obstacles in the path of the robot and in the path of the legs. There is also one detector and emitter placed at the rear of the robot for detection of objects in the rear of the robot for when it is backing up. The IR sensor layout is shown in figure 4. The IR detectors used were SHARP GP1U58Y and the emitters were infrared LEDs.

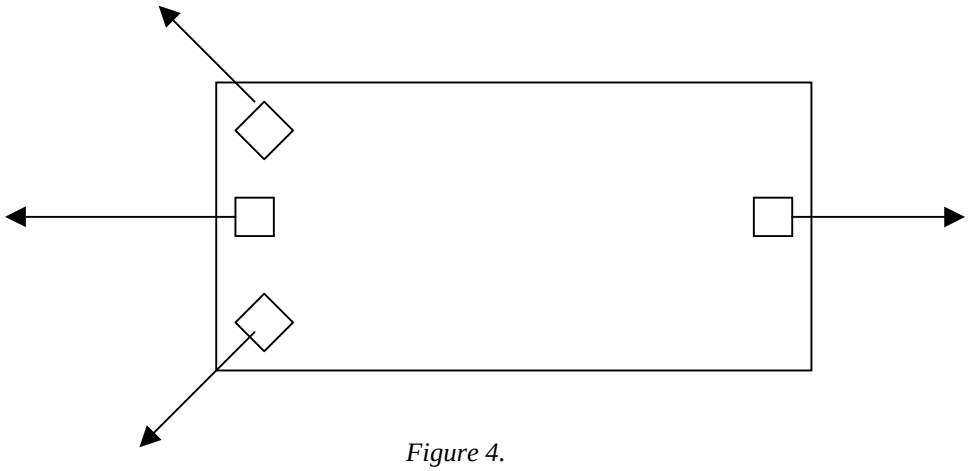

Figure 4.

Bump sensors were mounted under the rubber pad of the leg. These were to be used for detection of leg-ground contact and later for learning and a self-adjusting gait. Since

information about when the leg hits ground would allow for uneven terrain navigation by updating on-the-fly how far the legs lower or raise. However, this sensor was not implemented in software and is not used by the robot during its predefined method of walking. I only mention it to show a possible extension to the current platform.

B E H A V I O R S

Currently, the only behaviors the robot has implemented are walking and obstacle avoidance. Much more was envisioned but. due to the lack of time and extreme difficulties in getting the servos to work the learning aspects were not investigated. The building of the actual platform took more time that initially determined, which set back the reset of the project. The most restrictive to the development of other behaviors was the battery power required by the servos. Since it was unbeknownst to me that the servos would behave the same with low battery current as sending the wrong signal to the motor, I spent weeks writing and re-writing the servo control routines thinking they were incorrect. It was only determined at the end of the project due to a suggestion by Zer Liang that a battery capable of supply more current was needed.

C O N C L U S I O N S

While I did build and programming a functional walking robot, Dogbot is limited to walking on flat semi-smooth surfaces only. Some rough terrain will not allow for the motors to propel the body forward. Originally this limitation was meant to be overcome, but problems and setbacks made it impossible to even begin to explore this area during the course of the semester. Continuing development will be done to fix this limitation and create a more adroit representation of a four-legged animal.

Coding and getting the robot to walk were thought to be some of the more challenging concerns initially, but it was found to be quite simple when taking the mechanical design in to consideration. The only real difficult problem not overcome in this project till the end was the servo control due to battery power. Too much time was spent coding and translating from assembler to ICC11 and tweaking the algorithms. But overall, I have learned a great deal in the areas of mechanical design, electrical engineering, and autonomous agents. I am pleased with what I have accomplished in the limited time and with the setbacks due to my own ignorance.

Future work to be done on the robot is to improve the walking and add more sensors to gain more information about the surroundings. Mounting a CCD camera on top of it for remote exploration would be interesting, and with a faster processor it would be an excellent research topic to develop a walking style from visual information. Much more could be thought of to add to the robot, since anything can be expanded on, but the near

future needs a better walking style with more information and a agent capable of adjusting its gait to any condition that would allow for four-legged locomotion.

In summary, this project was challenging, rewarding, and frustrating. My hands on knowledge of what it takes to go from theory and design to actual implementation has increased greatly. The reward came in working for many hours to achieve a goal which was met when the robot finally did walk (although in limited capacity). I definitely took away a positive experience from the project and recommend the IMDL class to anyone.

A P P E N D I X

Dogbot.c

```
/*
˝
      Dogbot - a four-legged walking robot
˝

˝
      Andrew Reddish

 */
#include "hc11.h"
#include "analog.h"

#pragma interrupt_handler tov_hand oc1_hand oc2_hand oc3_hand oc4_hand
oc5_hand

#define bit0 0x01
#define bit1 0x02
#define bit2 0x04
#define bit3 0x08
#define bit4 0x10
#define bit5 0x20
#define bit6 0x40
#define bit7 0x80

#define ONE_SEC         32
#define HALF_SEC        16

/* motor constants */
#define MAX_PWM 4680
#define MIN_PWM 1400
#define RANGE_PWM (MAX_PWM-MIN_PWM)

/* offsets into the array for each leg */
#define LEFT_FRONT              0
#define RIGHT_FRONT             2
#define RIGHT_BACK              1
#define LEFT_BACK               3
#define HIP_JOINT               0
#define LIFT_JOINT              4
#define ELBOW_JOINT             8

#define LEFT_FRONT_HIP          0
#define RIGHT_FRONT_HIP         1
#define LEFT_BACK_HIP           3
#define RIGHT_BACK_HIP          2

#define LEFT_FRONT_LIFT         4
#define RIGHT_FRONT_LIFT        5
#define LEFT_BACK_LIFT          7
#define RIGHT_BACK_LIFT         6

#define LEFT_FRONT_ELBOW        8
```

```
#define RIGHT_FRONT_ELBOW       9
#define LEFT_BACK_ELBOW         10
#define RIGHT_BACK_ELBOW        11

/* used for selecting which joint to move */
#define HIP_JOINT_SELECT        0x01
#define LIFT_JOINT_SELECT       0x02
#define ELBOW_JOINT_SELECT      0x04

/* leg degrees */
#define HIP_FORWARD          3500
#define HIP_BACKWARD         2500
#define LIFT_UP              3500
#define LIFT_DOWN            2500
#define CENTER               3000

/* IR constants */

#define FRONT_LEFT             0
#define FRONT_RIGHT            1
#define FRONT_CENTER           2
#define REAR                   3
#define THRESHOLD              110

/* Walk constants */
#define STAND           0
#define FORWARD         1
#define TURN_RIGHT      2
#define TURN_LEFT       3
#define BACKWARD        4

/* Global variables */

char state;                  /* 0 = setup lines, 1 = wait till refresh
*/
char which_joint;            /* current joint selected */
char which_joint_mask[3];
char current_joint;          /* joint offset into table we are working
on */
int servo_current_position[12];     /* servo PWM times */
int servo_new_position[12];  /* new servo PWM times */
char overflows;              /* # of overflows of TCNT */
char WalkState;              /* current state of walking */
char IRSensors[4];           /* IR readings */

#define TOC1_ISR oc1_hand
#define TOC2_ISR oc2_hand
#define TOC3_ISR oc3_hand
#define TOC4_ISR oc4_hand
#define TOC5_ISR oc5_hand
#define TOF_ISR tov_hand

void Init();
void InitOC();
void Walk();
void CheckIR();
void UpdateServos();
```

```
void SmoothServos();
void SetServo(char which, int value);
int DegToPWM(char deg);

void tov_hand()
/* This is the interrupt routine that increments the # of overflows */
{
      overflows++;
      TFLG2 &= 0x80;
}

void Delay(char value)
/* 32.7*value ms delay */
{
    overflows = 0;

    TFLG2 &= 0x80;
    TMSK2 |= 0x80;

    while(overflows < value)
      asm("NOP");

    TMSK2 &= ~0x80;
}

void oc1_hand ()
/*
      This is the interrupt handler for OC1 and the main control block
      of the servo control algortihm.     Depending on the value of
state,
      this routine either sets up one of the four banks of servos, or
      waits till the next refresh cycle.
 */
{
      PORTD = 0;

      if(state == 0)  /* refresh */
      {
                  PORTD = which_joint_mask[which_joint]; /* select bank
*/

                  OC1M = 0x78;
                  OC1D = 0x78;
                  CFORC = 0x80;   /* force lines high */

                  which_joint++;
                  which_joint %=3;


                  TCTL1 = 0xAA;   /* set to clear on match */

                  TOC2 = TOC1 +
servo_current_position[current_joint+LEFT_FRONT];
                  TOC3 = TOC1 +
servo_current_position[current_joint+RIGHT_FRONT];
                  TOC4 = TOC1 +
servo_current_position[current_joint+RIGHT_BACK];
```

```
                  TOC5 = TOC1 +
servo_current_position[current_joint+LEFT_BACK];
                  TOC1 += 5000; /* set for next setup time at 2.5ms */

                  TFLG1 &= 0x78;  /* clear all flags for OC2-OC5 before
enabling them */
                  TMSK1 |= 0xf8;
                  OC1M = 0x00;

                  current_joint+=4; /* setup which joint is to be
serviced next*/

                  if(current_joint > ELBOW_JOINT)
                  {
                        current_joint=HIP_JOINT;
                        state = 1;
                  }
      }
      else
      {
            /* wait for 12.5 ms till next refresh */
            TOC1 += 25000;
            state = 0;
      };


      TFLG1 &= 0x80;           /* clear the flag */

}

void oc2_hand ()
/* This routine disables its interrupt and clears the flag */
{
      TFLG1 &= 0x40;
      TMSK1 &= ~0x40;

}

void oc3_hand ()
/* This routine disables its interrupt and clears the flag */
{
      TFLG1 &= 0x20;
      TMSK1 &= ~0x20;
}

void oc4_hand ()
/* This routine disables its interrupt and clears the flag */
{
      TFLG1 &= 0x10;
      TMSK1 &= ~0x10;
}

void oc5_hand ()
/* This routine disables its interrupt and clears the flag */
{
      TFLG1 &= 0x08;
      TMSK1 &= ~0x08;
```

```
}

void InitOC()
/* Initializes the registers on the HC11 */
{
      DDRD = 0x1f;
      PORTD = 0;

      PACTL = 0x88;

      TOC1 = 4000;
      OC1M = 0x78;

      TFLG1 &= 0x80;
      TMSK1 |= bit7;
}

void Init()
/* Setup the initail values of all state variables, etc */
{
      int i;
      for(i=0;i<12;i++)
            servo_current_position[i] = servo_new_position[i] = CENTER;

      which_joint_mask[0] = HIP_JOINT_SELECT;
      which_joint_mask[1] = LIFT_JOINT_SELECT;
      which_joint_mask[2] = ELBOW_JOINT_SELECT;
      which_joint = state = 0;
      current_joint = HIP_JOINT;

      WalkState = FORWARD;

      *(unsigned char *)(0x7000)=0xFF; /* turn on IR */
}

void main(void)
{
      init_analog();
      Init();                 /* Initialize variables */
      InitOC();

      asm(" cli");

      Delay(ONE_SEC); /* wait 1 seconds before we begin to move */

      while(1)
      {
            CheckIR();
            Walk();
      }
}

void CheckIR()
/* This procedure checks the IR and corrects the walking state
   if an obstacle is in the path
*/
{
```

```
      IRSensors[FRONT_LEFT]=analog(0);
      IRSensors[FRONT_RIGHT]=analog(2);
      IRSensors[FRONT_CENTER]=analog(1);
      IRSensors[REAR]=analog(3);

      WalkState=FORWARD;

      if(IRSensors[FRONT_CENTER] > THRESHOLD ||
         (IRSensors[FRONT_LEFT] > THRESHOLD && IRSensors[FRONT_RIGHT] >
THRESHOLD)) {
            WalkState=BACKWARD;
      }
      if(IRSensors[FRONT_LEFT] > THRESHOLD) {
            WalkState=TURN_RIGHT;
      }
      if(IRSensors[FRONT_RIGHT] > THRESHOLD) {
            WalkState=TURN_LEFT;
      }
      if(IRSensors[REAR] > THRESHOLD && WalkState==BACKWARD)
      {
            WalkState=FORWARD;
      }

}

void Walk()
/* The main walking routine, this sets the PWM of each servo
   to acheive the walking
*/
{
      switch(WalkState)
      {
      case STAND:
            SetServo(LEFT_FRONT_LIFT,CENTER);
            SetServo(LEFT_BACK_LIFT,CENTER);
            SetServo(RIGHT_BACK_LIFT,CENTER);
            SetServo(RIGHT_FRONT_LIFT,CENTER);
            SetServo(LEFT_FRONT_ELBOW,CENTER);
            SetServo(LEFT_BACK_ELBOW,CENTER);
            SetServo(RIGHT_BACK_ELBOW,CENTER);
            SetServo(RIGHT_FRONT_ELBOW,CENTER);
            SetServo(LEFT_FRONT_HIP,CENTER);
            SetServo(RIGHT_BACK_HIP,CENTER);
            SetServo(LEFT_BACK_HIP,CENTER);
            SetServo(RIGHT_FRONT_HIP,CENTER);
            UpdateServos();
            Delay(HALF_SEC); /* delay 1 second */

            break;
      case BACKWARD:
            SetServo(LEFT_FRONT_LIFT,LIFT_UP);
            SetServo(LEFT_BACK_LIFT,LIFT_UP);
            SetServo(RIGHT_BACK_LIFT,LIFT_UP);
            SetServo(RIGHT_FRONT_LIFT,LIFT_UP);
            SetServo(LEFT_FRONT_ELBOW,LIFT_UP);
            SetServo(LEFT_BACK_ELBOW,LIFT_UP);
            SetServo(RIGHT_BACK_ELBOW,LIFT_UP);
```

```
        SetServo(RIGHT_FRONT_ELBOW,LIFT_UP);
        UpdateServos();
        Delay(HALF_SEC); /* delay 1 second */

        SetServo(LEFT_FRONT_HIP,HIP_FORWARD);
        SetServo(RIGHT_BACK_HIP,HIP_BACKWARD);
        SetServo(LEFT_BACK_HIP,HIP_BACKWARD);
        SetServo(RIGHT_FRONT_HIP,HIP_FORWARD);
        UpdateServos();
        Delay(HALF_SEC); /* delay */

        SetServo(LEFT_FRONT_LIFT,LIFT_DOWN);
        SetServo(LEFT_BACK_LIFT,LIFT_DOWN);
        SetServo(RIGHT_BACK_LIFT,LIFT_DOWN);
        SetServo(RIGHT_FRONT_LIFT,LIFT_DOWN);
        SetServo(LEFT_FRONT_ELBOW,LIFT_DOWN);
        SetServo(LEFT_BACK_ELBOW,LIFT_DOWN);
        SetServo(RIGHT_BACK_ELBOW,LIFT_DOWN);
        SetServo(RIGHT_FRONT_ELBOW,LIFT_DOWN);

        UpdateServos();
        Delay(HALF_SEC); /* delay */


        SetServo(LEFT_FRONT_HIP,HIP_BACKWARD);
        SetServo(RIGHT_BACK_HIP,HIP_FORWARD);
        SetServo(LEFT_BACK_HIP,HIP_FORWARD);
        SetServo(RIGHT_FRONT_HIP,HIP_BACKWARD);
        UpdateServos();
        Delay(HALF_SEC); /* delay */
        break;

case TURN_LEFT:
        SetServo(LEFT_FRONT_LIFT,LIFT_DOWN);
        SetServo(LEFT_BACK_LIFT,LIFT_DOWN);
        SetServo(RIGHT_BACK_LIFT,LIFT_DOWN);
        SetServo(RIGHT_FRONT_LIFT,LIFT_DOWN);
        SetServo(LEFT_FRONT_ELBOW,LIFT_DOWN);
        SetServo(LEFT_BACK_ELBOW,LIFT_DOWN);
        SetServo(RIGHT_BACK_ELBOW,LIFT_DOWN);
        SetServo(RIGHT_FRONT_ELBOW,LIFT_DOWN);
        UpdateServos();
        Delay(HALF_SEC); /* delay */

        SetServo(LEFT_FRONT_HIP,HIP_BACKWARD);
        SetServo(RIGHT_BACK_HIP,HIP_FORWARD);
        SetServo(LEFT_BACK_HIP,HIP_FORWARD);
        SetServo(RIGHT_FRONT_HIP,HIP_FORWARD);
        UpdateServos();
        Delay(HALF_SEC); /* delay */

        SetServo(LEFT_FRONT_LIFT,LIFT_UP);
        SetServo(LEFT_BACK_LIFT,LIFT_UP);
        SetServo(RIGHT_BACK_LIFT,LIFT_UP);
        SetServo(RIGHT_FRONT_LIFT,LIFT_UP);
        SetServo(LEFT_FRONT_ELBOW,LIFT_UP);
        SetServo(LEFT_BACK_ELBOW,LIFT_UP);
```

```
        SetServo(RIGHT_BACK_ELBOW,LIFT_UP);
        SetServo(RIGHT_FRONT_ELBOW,LIFT_UP);

        UpdateServos();
        Delay(HALF_SEC); /* delay */


        SetServo(LEFT_FRONT_HIP,HIP_FORWARD);
        SetServo(RIGHT_BACK_HIP,HIP_BACKWARD);
        SetServo(LEFT_BACK_HIP,HIP_BACKWARD);
        SetServo(RIGHT_FRONT_HIP,HIP_FORWARD);
        UpdateServos();
        Delay(HALF_SEC); /* delay */
        break;

    case TURN_RIGHT:
        SetServo(LEFT_FRONT_LIFT,LIFT_DOWN);
        SetServo(LEFT_BACK_LIFT,LIFT_DOWN);
        SetServo(RIGHT_BACK_LIFT,LIFT_DOWN);
        SetServo(RIGHT_FRONT_LIFT,LIFT_DOWN);
        SetServo(LEFT_FRONT_ELBOW,LIFT_DOWN);
        SetServo(LEFT_BACK_ELBOW,LIFT_DOWN);
        SetServo(RIGHT_BACK_ELBOW,LIFT_DOWN);
        SetServo(RIGHT_FRONT_ELBOW,LIFT_DOWN);
        UpdateServos();
        Delay(HALF_SEC); /* delay */

        SetServo(LEFT_FRONT_HIP,HIP_BACKWARD);
        SetServo(RIGHT_BACK_HIP,HIP_FORWARD);
        SetServo(LEFT_BACK_HIP,HIP_BACKWARD);
        SetServo(RIGHT_FRONT_HIP,HIP_BACKWARD);
        UpdateServos();
        Delay(HALF_SEC); /* delay */

        SetServo(LEFT_FRONT_LIFT,LIFT_UP);
        SetServo(LEFT_BACK_LIFT,LIFT_UP);
        SetServo(RIGHT_BACK_LIFT,LIFT_UP);
        SetServo(RIGHT_FRONT_LIFT,LIFT_UP);
        SetServo(LEFT_FRONT_ELBOW,LIFT_UP);
        SetServo(LEFT_BACK_ELBOW,LIFT_UP);
        SetServo(RIGHT_BACK_ELBOW,LIFT_UP);
        SetServo(RIGHT_FRONT_ELBOW,LIFT_UP);

        UpdateServos();
        Delay(HALF_SEC); /* delay */


        SetServo(LEFT_FRONT_HIP,HIP_FORWARD);
        SetServo(RIGHT_BACK_HIP,HIP_BACKWARD-200);
        SetServo(LEFT_BACK_HIP,HIP_BACKWARD);
        SetServo(RIGHT_FRONT_HIP,HIP_FORWARD-200);
        UpdateServos();
        Delay(HALF_SEC); /* delay */
        break;

    case FORWARD:
        SetServo(LEFT_FRONT_LIFT,LIFT_DOWN);
```

```
            SetServo(LEFT_BACK_LIFT,LIFT_DOWN);
            SetServo(RIGHT_BACK_LIFT,LIFT_DOWN);
            SetServo(RIGHT_FRONT_LIFT,LIFT_DOWN);
            SetServo(LEFT_FRONT_ELBOW,LIFT_DOWN);
            SetServo(LEFT_BACK_ELBOW,LIFT_DOWN);
            SetServo(RIGHT_BACK_ELBOW,LIFT_DOWN);
            SetServo(RIGHT_FRONT_ELBOW,LIFT_DOWN);
            UpdateServos();
            Delay(HALF_SEC); /* delay */

            SetServo(LEFT_FRONT_HIP,HIP_BACKWARD);
            SetServo(RIGHT_BACK_HIP,HIP_FORWARD);
            SetServo(LEFT_BACK_HIP,HIP_FORWARD);
            SetServo(RIGHT_FRONT_HIP,HIP_BACKWARD);
            UpdateServos();
            Delay(HALF_SEC); /* delay */

            SetServo(LEFT_FRONT_LIFT,LIFT_UP);
            SetServo(LEFT_BACK_LIFT,LIFT_UP);
            SetServo(RIGHT_BACK_LIFT,LIFT_UP);
            SetServo(RIGHT_FRONT_LIFT,LIFT_UP);
            SetServo(LEFT_FRONT_ELBOW,LIFT_UP);
            SetServo(LEFT_BACK_ELBOW,LIFT_UP);
            SetServo(RIGHT_BACK_ELBOW,LIFT_UP);
            SetServo(RIGHT_FRONT_ELBOW,LIFT_UP);

            UpdateServos();
            Delay(HALF_SEC); /* delay */


            SetServo(LEFT_FRONT_HIP,HIP_FORWARD);
            SetServo(RIGHT_BACK_HIP,HIP_BACKWARD);
            SetServo(LEFT_BACK_HIP,HIP_BACKWARD);
            SetServo(RIGHT_FRONT_HIP,HIP_FORWARD);
            UpdateServos();
            Delay(HALF_SEC); /* delay */
            break;
      }
}

void SetServo(char which, int value)
/* Sets the value of a servo in its table */
{
      if(value > MAX_PWM)
      {
            servo_new_position[which]=MAX_PWM;
            return;
      }
      if(value < MIN_PWM)
      {
            servo_new_position[which]=MIN_PWM;
            return;
      }
      servo_new_position[which]=value;
}

void UpdateServos()
```

```
/* places the new values into the current table */
{
      char i;

      for(i=0;i<12;i++)
      {
            servo_current_position[i] = servo_new_position[i];
      }
}
```