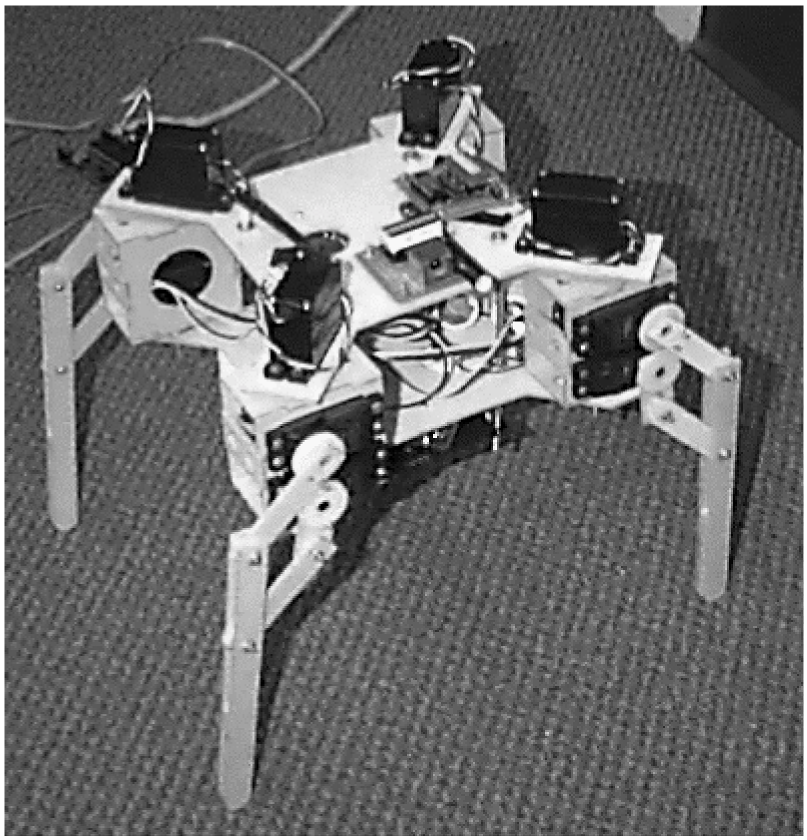
Bob,
An Autonomous Quadrupedal Robot

Final Report



"...like a dog's walking on his hind legs. It is not done well, but you are surprised to find it done at all." - Samuel Johnson

Steve Stancliff
University of Florida
Intelligent Machines Design Laboratory
Submitted: April 23, 1998
Edited: March 25, 2000

Acknowledgments

This project would not have been completed without the assistance of all of the IMDL TAs, and especially Jenny Laine, who wrote the code that resides on the single-chip board and directly controls the servos.

Bob owes a great deal to the robot "Thing" by Willard S. MacDonald. It was not my intention to end up with a robot so close to Thing in design, but as I experimented and encountered various problems, I found that the solutions implemented in Thing were usually the best solutions I could find.

Thanks to Rohm Electronics for donating a LCD panel for this project, which, unfortunately, is not yet implemented.

Abstract

The objective of this project was to design, construct, and test hardware and software to create a small autonomous quadrupedal robot. Four legs were chosen for this project as a reasonable compromise between the hardware complexity when many legs are used and the software complexity required for stability when fewer legs are used. A design giving three degrees of freedom (DOF) per leg was chosen because fewer DOFs restricts the robot's mobility, and more DOFs are redundant. At the time of the writing of this paper, the robot is capable of walking forward and backward over level surfaces, turning to the left, and raising and lowering it's body. It implements minimal obstacle detection using a single forward looking infrared emitter/detector pair. It also uses a photocell for detection of ambient light level.

Contents

List of Tables

List of Figures

1. Executive Summary

Most of the effort to date on Bob has been spent on mechanical and electrical hardware. Once the basic configuration was chosen, a bottom-up design methodology was followed.

The first stage in the design was to establish a leg design. Several balsa prototypes were constructed and tested for range of motion and sturdiness. In the end, the leg design of "Thing" was adopted.

The next design issue was how to mount the lift and extension servos to the base and to the swing servo. A "shoulder box" was quickly settled on to contain these servos in a rigid arrangement. The problem of how to mount the shoulder box to the base while allowing free rotation was one of the more difficult ones. Eventually, a ball-bearing wheel was located which could be used between the base and the shoulder box.

In an attempt to differentiate Bob from Thing, the base of the prototype was made square. Once the prototype was assembled, it was obvious that this shape severely limited the range of motion of the swing servo. A design similar to Thing, with the swing servos extending diagonally out from a square center, was adopted for the final robot.

At the time of the prototype's construction, the primary processor board was to be a Motorola EVBU in combination with a Mekatronix ME11. When the prototype was assembled, it was obvious that this combination was a limiting factor in the size and weight of the robot. The pair of boards was replaced by the Mekatronix MRC11, which not only weighs half as much, but also allows the platform to be 30% smaller, resulting in additional weight savings.

In parallel with the hardware development, three sensor systems were developed. A sound sensor was designed around the LM741 op-amp, to amplify and rectify microphone signals. A photocell sensor was developed to detect ambient light-level, and a MIL-standard infrared emitter/detector pair was used for obstacle detection.

Software development to this point has been trivial, since pre-existing servo-controller code was used. The higher level code is less than a hundred lines in length at this point. Future developments will concentrate in this area.

2. Introduction

2.1 Legged Robots

Most existing mobile robots rely on wheels for locomotion. This is the best solution for traversing relatively even surfaces, but wheeled platforms have difficulties dealing with obstacles which legged platforms can simply step over. On the other hand, wheeled platforms are inherently stable, since the wheels are intended to remain on the ground. A legged robot must shift its center of gravity as it walks in order to maintain stability. In addition, the driving mechanism for a wheeled vehicle can be relatively simple and inexpensive to construct, while a leg mechanism requires complicated linkages and multiple motors, as well as sensors to determine the location of the leg relative to the ground.

2.2 Number of Legs

Choosing the number of legs for the robot amounts to a tradeoff between mechanical complexity (which implies cost and weight) and software complexity. Existing robots, as well as most walking creatures, are mostly of three varieties: two-, four-, or six-legged. Successful implementation of a two-legged robot is an extremely complicated task, and beyond the scope of a one-semester, one-student project. A six-legged robot, on the other hand, requires greater mechanical complexity, and offers more opportunity for unit failure. In addition, the problem of a six-legged robot seems to have been reasonably accomplished, as there are several examples available. For these reasons, a four-legged design was chosen.

2.3 Objectives

The primary design criteria for this robot were as follows:

- minimal cost
    - incorporate already-owned components
    - use cheap and easily-worked materials for platform
    - minimize weight to make use of cheaper servos

- robustness
    - solid design and construction

- able to be implemented in stages
    - modular design
    - room for expansion

The objectives for this semester were:

- design and construct the platform

- implement a simple, hard-coded walking routine

- implement minimal sensors for obstacle avoidance

- if time allows, implement a more intelligent walking algorithm

2.4 Organization of Paper

Section 3 describes the robot system as a whole, and how it meets the design criteria.

Section 4 describes the platform structure.

Section 5 describes the servos and servo-control system.

Section 6 describes the sensors.

Section 7 describes the sensor-based behaviors.

Section 8 describes the experimental testing of the design.

Section 9 gives a summary of the project and plans for future work.

3. Integrated System

Bob is an autonomous mobile robot which uses four legs for locomotion. Each of these legs has three servos controlling it's rotation, lift, and extension, giving each leg three degrees of freedom (DOF). With three DOF, the legs have a wide range of motion, and can move directly between any two points in that range.

The platform and legs are constructed from plywood which is designed for use in model airplanes, and thus has a high strength-to-weight ratio. The leg actuators are "standard" model aircraft servos. The legs and platform were designed to give overlapping workspaces for the legs, which increases the maneuverability of the robot, but requires the software to keep the legs from getting tangled.

The servos are controlled by a Motorola 68HC711E9 microprocessor running in single-chip mode on a Mekatronix MSCC11 board. This board generates the PWM signals required for the servos. The code on this board resides in the EEPROM and is listed in Appendix D.

The "intelligence" of the robot comes from a second 68HC711E9 running in expanded mode on a Mekatronix MRC11 board with 32k of SRAM (Figure 1.1). This board sends servo positions to the single-chip board through the serial interface. The sensors are attached to this board. The code for this board is listed in Appendix E.

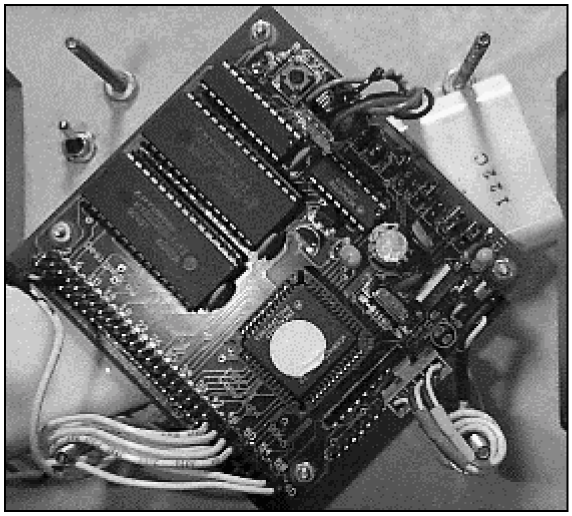


Figure 1.1 - MRC11

The robot currently walks in a statically-stable mode, which is the definition of walking, as opposed to running. To be statically stable at all times, the center of gravity must be constantly shifted to keep it within the base of support provided by the three legs which are on the ground. It should be possible to implement a “running” behavior where two legs are off the ground at once, but this will require some type of tilt-sensors and more complex software.

A list of the parts used in this project is given in Appendix A.

4. Mobile Platform

4.1 Requirements

The platform design was driven primarily by three things:

- the need to minimize weight,
- the need to maximize leg workspace
- the size of affordable and obtainable components

For the type of robot envisioned, minimizing weight was very important because when the robot is standing, the lift servos have to support the weight of the robot. Greater weight would require more powerful servos, at much greater cost. Weight reduction techniques included choosing a thin (3/32") plywood and cutting holes in the parts. A significant amount of additional material could be removed from the upper and lower bases without harming the structural integrity, but this was not done in order to leave space for mounting future additions. A list of component weights is given in Appendix B.

The need to provide maximum maneuverability and symmetry of motion for the legs dictated the shape of the platform. The "shoulder boxes" are extended diagonally away from the body at the corners. This design mimics that of "Thing", and was chosen after more simple designs proved to limit the workspace severely.

The size of the platform was limited by the size of the processor boards and the servos. The resulting design has a large amount of unused surface area and internal volume.

4.2 Construction

The platform was constructed of 3/32" model-aircraft plywood. The components were drawn in AutoCAD and then cut out by the T-Tech routing machine. This machine made it possible to cut shapes which would have been difficult or impossible to cut by hand, and cut so accurately that the finished product is very finished-looking.

The upper and lower bases are attached to one another by four bolts, which extend below the lower base and provide a support for the robot when the servos are off. The swing servos are mounted on the upper base, and the rotation and extension servos are mounted in "shoulder boxes" which are attached at the top to the swing servos, and at the bottom to a wheel with ball bearings. This wheel supports the weight of the shoulder, reducing stress on the swing servos.

The mounting of these bearings was the most complicated part of the fabrication. In order to provide free rotation of the shoulder, it was desired that the inner race of the bearing touch only the shoulder box and not the bottom base, and that the outer race touch only the base and not the box. This was accomplished by routing out one and two layers of the five-ply plywood to provide recesses for the bearing. The bearing is attached to the shoulder by a bolt, and is held in place on the base by pressure. Again, this fabrication would have been difficult or impossible without the T-Tech router.

The shoulder boxes were designed with interlocking joints. This proved to be difficult, since in the act of offsetting the AutoCAD drawing to allow for the width of the router bit, the tabs become wider and the slots narrower. The prototype shoulder boxes, after significant filing, held together well with no glue. The boxes on the final robot are held together more by glue than by the joints themselves.

The leg design follows that of “Thing”. A good discussion of leg designs is found in (MacDonald 94).

AutoCAD drawings of the wooden parts are located in Appendix C.

5. Actuation

5.1 Servos

Bob is motivated by twelve model-airplane servos. Four standard Futaba servos are used as the swing servos, and eight standard Hitec servos for the extension and lift servos. The Hitec servos are rated at 42 oz-in of torque at 4.8 V, and all of the servos weigh approximately 1.7 oz. The Futaba servos were removed from a model airplane, and the Hitec servos were chosen entirely for low cost. It will probably be prudent to upgrade to ball-bearing servos, at least for the lift servos, which have the greatest load under normal circumstances.

5.2 Controllers

All twelve servos are controlled by the MSCC11 board, which receives commands via its serial port, and outputs PWM signals to the servos. The code for this board was provided by Jennifer Laine. The serial commands to this board consist of three characters in succession. The first character indicates the start of a message, the second selects the servo to be positioned, and the third selects the position for the servo. The possible range of positions accommodated by the software is 0x00 to 0xA5, but in practice it was found that the range of motion of the servos sometimes fell outside this range. For the extension and lift servos, only a small portion of the servo's range is needed, but for the swing servos, it would be preferred to have the full range of motion. The servo control code is located in Appendix D.

As currently implemented, the code on the MRC11 board contains servo-position data for various motion regimes, such as walking forward or turning right. The amount of data needed for complex behavior is very large. The amount of data could be reduced by implementing the kinematic equations of motion for the legs in the MRC11, but this would require significant work on the part of the programmer and the processor. The current data uses only a small fraction of the 64k of memory available on the MRC11 board, so for now it seems better to leave the processor free to perform higher-level tasks.

6. Sensors

6.1 Photosensor

Bob is equipped with a CdS photocell, which is used to detect the ambient light level. The photocell acts as a variable resistor, with resistance inversely proportional to light level. The sensor circuit consists of the photocell in series with a resistor. The resistor value was chosen to give a wide range of output voltages. The output is the voltage across the resistor. Total darkness gives an analog port reading of about 50, while bright light gives a reading of about 225.

6.2 Infrared Emitter/Detector

Bob also possesses an infrared emitter/detector pair of the standard MIL type. The detector is a Sharp Digital IR detector which has been modified to provide an analog output. The emitter is housed in a cylindrical container in order to control the spread of the IR beam somewhat. The analog port reading idles at about 90, and rises to about 130 for an object which is very close to the sensor. A reading of about 120 indicates an object within about a foot of the robot.

6.3 Tilt Sensor

A mercury switch is mounted on Bob as a poor-man's level-sensor. The switch is wired and provides valid input to the MRC11, but it has not been calibrated to a specific angle, and no behaviors are currently implemented using this sensor. To receive useful information about the position of the platform would require at least four of these sensors, two on each axis.

6.4 Sound Sensor

In response to the requirement to develop a "new" sensor for the IMDL project, I decided to build a sensor which would let me create an audiotropic behavior in my robot. The initial design for the audio sensor used a LM386 audio amplifier IC to amplify the signal from the microphone. The LM386 caused distortion in the microphone, so the sensor was developed on the LM741 general-purpose op-amp instead. The sensor works will in this configuration, however, the LM741 is a dual-power chip, and is therefore unsuitable for a mobile platform. Attempts were made to port the circuit to the LM324, and problems similar to the LM386 were encountered. Due to time constraints, further development was not carried out, and the sensor is not currently implemented on Bob. The development and testing of the sound sensor are described in more detail in Section 8.

7. Behaviors

Since Bob looks and moves like a small animal, it made sense to implement animal-like behaviors. Two behaviors are currently implemented.

7.1 Hibernation

When the photocell detects a low light level, Bob crouches down and turns off his servos. In nature, a small animal might react this way to the shadow of a passing bird. Bob reactivates and resumes moving about when he detects a bright light source. For an intermediate light level, Bob keeps doing what he was already doing - moving or hibernating.

7.2 Obstacle Detection/Avoidance

When the forward IR detects an object nearby, Bob raises up to his full height, in order to try and scare it away. If it doesn't move away in a short time, he decides that it must not be scared of him, so he retreats until he doesn't see it anymore, and then turns away and goes back to wandering. Similar behavior can be observed in many animals.

8. Experimental Results

The only formal experiments conducted were on the sound sensor. These are summarized here.

The amplifier/rectifier circuit based on the LM741 is shown in Figure 8.1. The two primary factors in choosing the components were the need for a large output voltage change for a small input signal, and the need to maintain a large difference between the idle output voltage and the saturation output voltage. Increasing the gain to accomplish the first goal also increased the DC offset at the output, defeating the second goal, so a compromise had to be made.

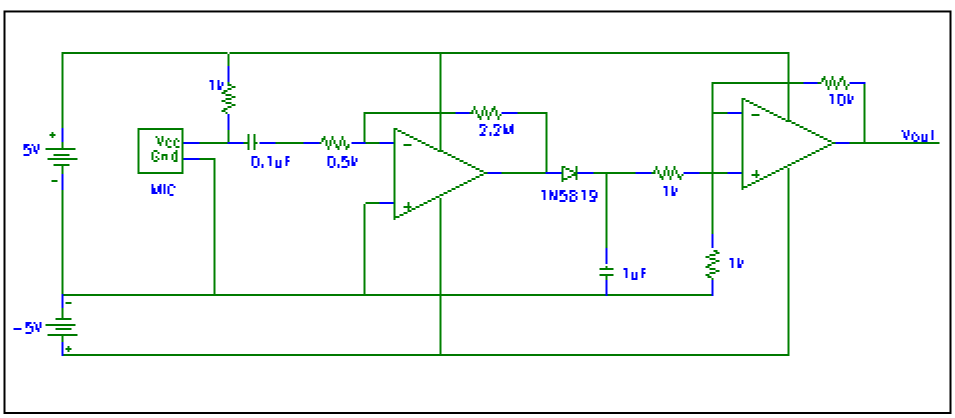


Figure 8.1 - Sound Sensor Schematic

The frequency response of the microphone was measured by using a signal of 10V amplitude output through a small speaker, at a constant position relative to the microphone. The results of this testing are shown in Table 8.1 and Figure 8.2. This test actually measures the combined response of the speaker and the microphone, but a better test could not be done without a sound pressure level meter.

Table 8.1 - Microphone Frequency Response

Vin 22 V p-p

f (kHz)	Vout (mV p-p)	Gain	Gain (dB)
0.18	11	0.0005	-66
0.35	28	0.0013	-58
0.76	18	0.0008	-62
1.1	27	0.0012	-58
2.0	65	0.0030	-51
3.0	220	0.0100	-40
3.9	113	0.0051	-46
6.1	110	0.0050	-46
8.2	21	0.0010	-60
9.7	12	0.0005	-65

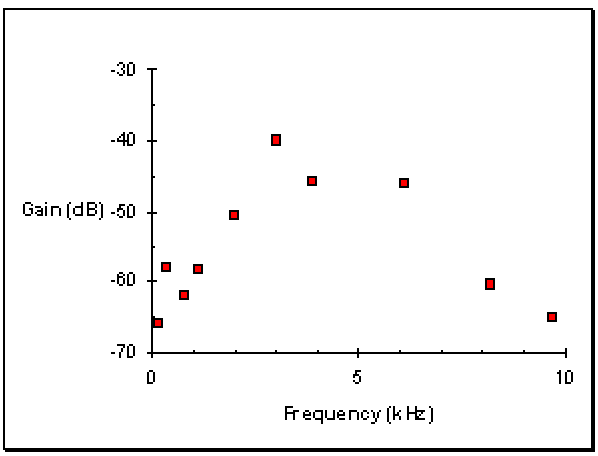


Figure 8.2 - Microphone Frequency Response

The frequency responses of the amplifiers were measured individually by inputting a signal from a function generator. The results are shown in Table 8.2 and Figure 8.3 for the first amplifier and Table 8.3 and Figure 8.4 for the second amplifier.

Table 8.2 - Frequency Response of First Amplifier

Vin 22 mV

f (kHz)	Vout (V)	Gain	Gain (dB)
0.096	12.5	568	55
0.30	9.9	450	53
0.50	8.6	391	52
0.72	8	364	51
0.93	7.6	345	51
1.0	7.5	341	51
1.1	7.3	332	50
1.6	6.9	314	50
2.0	5.6	255	48
3.0	3.9	177	45
4.0	3	136	43
5.0	2.6	118	41
6.1	2.1	95	40
7.1	1.8	82	38
8.2	1.5	68	37
9.3	1.3	59	35
10.3	1.1	50	34

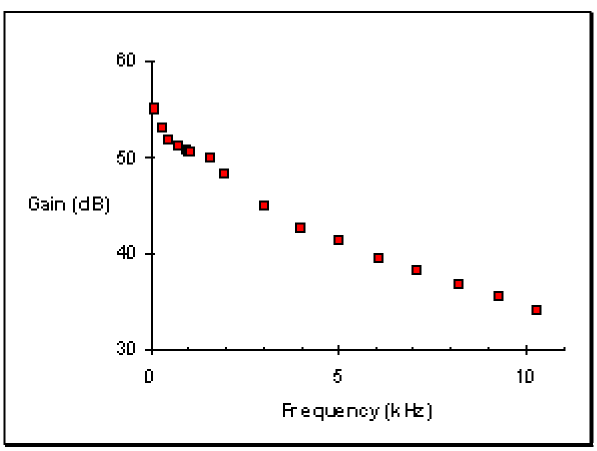


Figure 8.3 - Frequency Response of First Amplifier

Table 8.3 - Frequency Response of Second Amplifier

Vin 21 mV p-p

f (kHz)	Vout (mV)	Gain	Gain (dB)
0.97	140	6.7	16
0.20	180	8.6	19
0.30	182	8.7	19
0.40	182	8.7	19
0.50	180	8.6	19
0.61	177	8.4	19
0.71	177	8.4	19
0.82	176	8.4	18
0.92	174	8.3	18
1.0	171	8.1	18
1.1	169	8.0	18
1.2	166	7.9	18
2.1	151	7.2	17
3.0	132	6.3	16
4.1	115	5.5	15
5.1	102	4.9	14
6.1	93	4.4	13
7.1	83	4.0	12
8.2	77	3.7	11
9.3	69	3.3	10
10.3	64	3.0	10

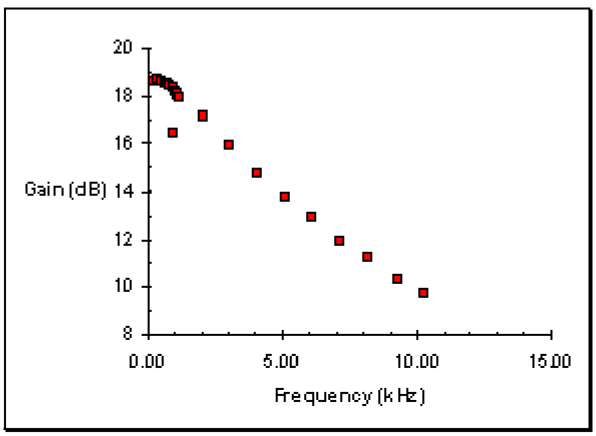


Figure 8.4 - Frequency Response of Second Amplifier

9. Conclusions

At this point, Bob has met the semester objectives of creating a walking platform with obstacle avoidance and hard-coded flat-floor walking. So far, no new ground has been covered by this project, but Bob is a good platform which can be used for more advanced studies at a later time. In addition, Bob is an improvement in many ways over his ancestor, Thing. Bob is cheaper, smaller, lighter, uses fewer processors and is of simpler construction than Thing.

Some future extensions which I envision for Bob are:

- add more movement types: walking sideways, turning right, waving a paw, etc.
- add the ability to climb objects. this probably involves the development of contact sensors for the feet.
- improve obstacle avoidance with more IR sensors and perhaps whisker-type sensors.
- add a charging circuit
- improve and implement the sound sensor

References

Alexander, R. McN., “The Gaits of Bipedal and Quadrupedal Animals.”, *International Journal of Robotics Research*, Summer, 1984.

Coggin, D., "Bot'arina", University of Florida Intelligent Machines Design Laboratory, 1995.

Fryman, J., "Antaean", University of Florida Intelligent Machines Design Laboratory, 1996.

Hirose, Shigeo, “A Study of Design and Control of a Quadrupedal Walking Vehicle.” *International Journal of Robotics Research*, Summer, 1984.

Huber, M., MacDonald, W., Grupen, R., “A Control Basis For Multilegged Walking.” *IEEE Conference on Robotics and Automation, April 1996, Volume 4*, pp. 2988-2993.

_____, “Building Walking Gaits for Irregular Terrain from Basis Controllers.” *IEEE Conference on Robotics and Automation, 1997*.

Klein, Charles, et. al., “Use of Force and Attitude Sensors for Locomotion of a Legged Vehicle over Irregular Terrain.”, *International Journal of Robotics Research,* Summer, 1983.

MacDonald, W.S., "Design and Implementation of a Multilegged Walking Robot", University of Massachusetts - Amherst Laboratory for Perceptual Robotics, 1994.

Osorio, R.J., "Spot: An autonomous mobile platform", University of Florida Intelligent Machines Design Laboratory, 1997.

Reddish, A., "Dogbot", University of Florida Intelligent Machines Design Laboratory, 1997.

Reibert, Marc, *Legged Robots That Balance*, MIT Press, 1986.

Van Anda, J., "Quadro", University of Florida Intelligent Machines Design Laboratory, 1997.

APPENDICES

Appendix A - Parts List

Table A.1 - Component Listing with Costs

Part	Quan	Manufacturer	Source	Price
MRC11 Board for 68HC11	1	Mekatronix	MIL	$70 est
MSCC11 Board for 68HC11	1	Mekatronix	MIL	$30
MB2325 Serial Board	1	Mekatronix	MIL	$11
XC68HC711E9 MPU	2	Motorola	N/A	N/A
HS-300 Servo	8	Hitec	Major Hobby	$80
FP-S28 Servos	4	Futaba	N/A	$60 est
3/32”x12”x24” plywood	2			$6
Bearings	4		Home Depot	$6
GP1U58X IR Sensor	1	Sharp	MIL	$3
IR LED	1		MIL	$1
Misc Electronics		Various	Various	$20 est
Misc Hardware		Various	Various	$20 est
TOTAL				~$300

Appendix B - Weights

Table B.1 - Component Weights

Component	Weight (oz)
MSCC11 Board	
MRC11 Board	
6V Battery for Servos	8.5
9.6V Battery for Electronics	7.1
Servo	1.7
Entire Assembly	56.7

Appendix C - Mechanical Drawings[1]

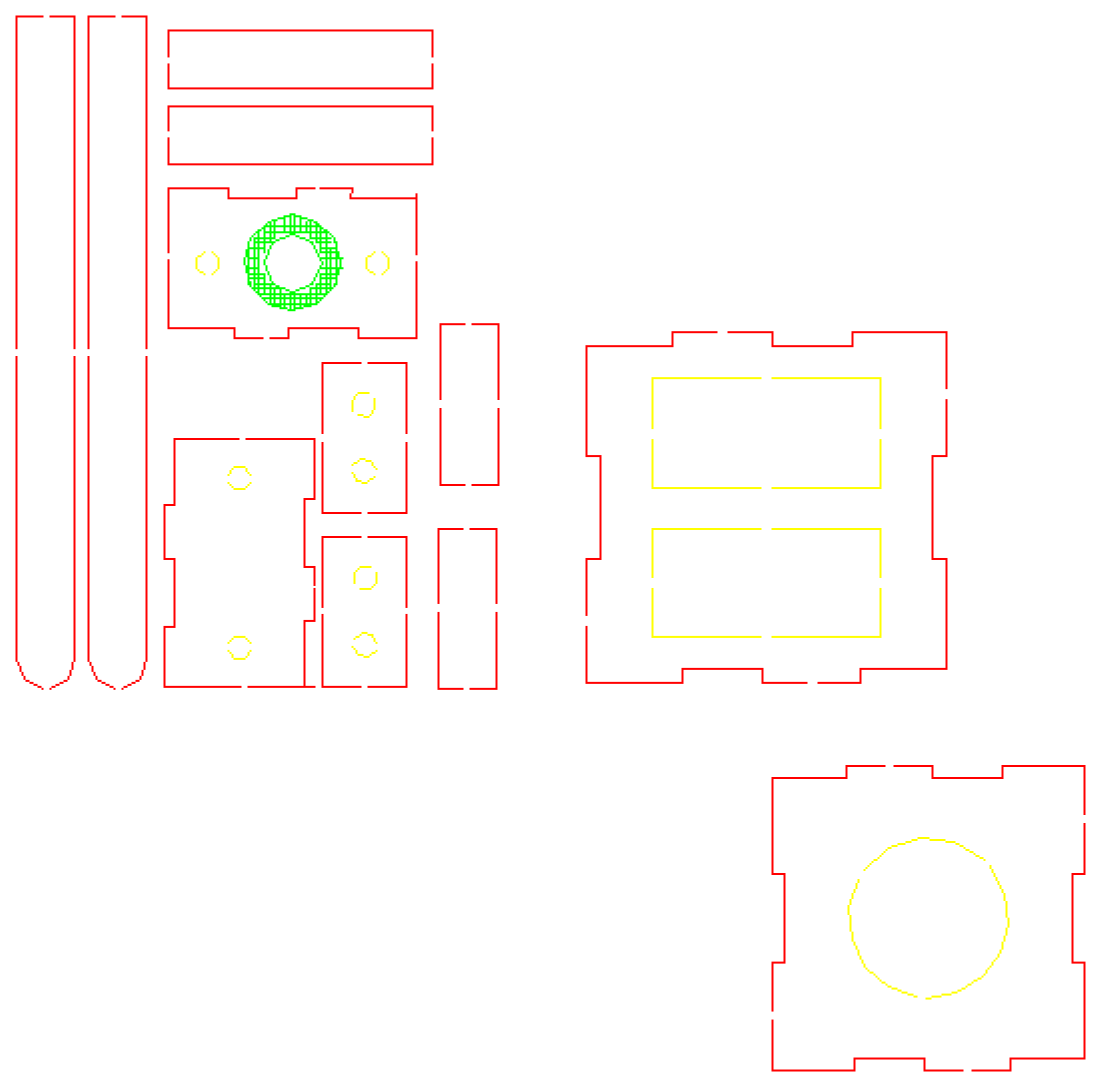


Figure C.1 - Leg And Shoulder Box

[1] Drawings are not at a constant scale.

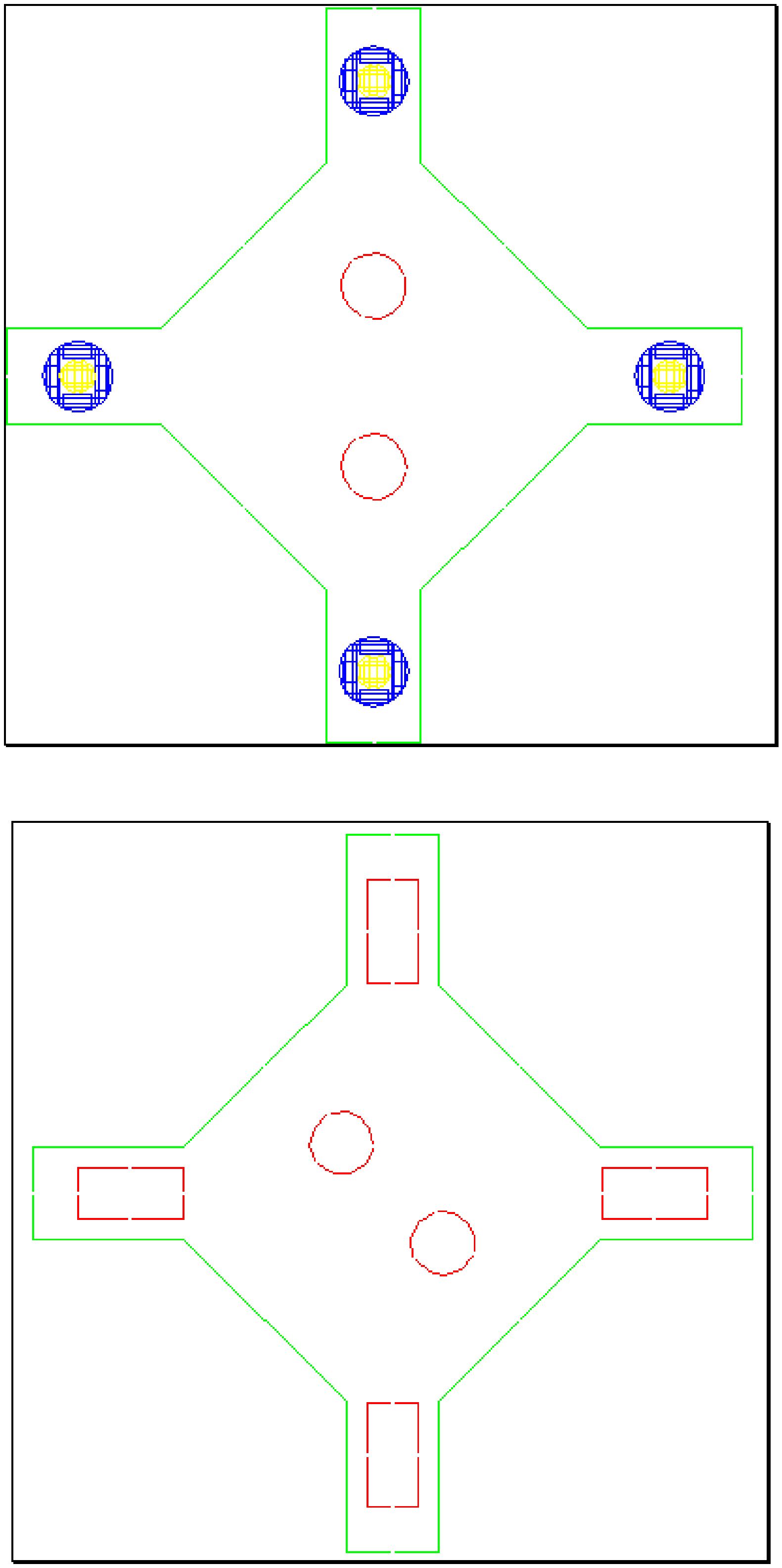


Figure C.2 - Upper and Lower Platforms

Appendix D - Servo Code

```
*********************************************************************
* THIS IS THE PROGRAM WHICH ALLOWS A USER TO CONTROL UP TO 16 SERVOS *
* FROM A TERMINAL OR FROM A HIGHER LEVEL PROCESSOR.  BASICALLY, IT   *
* INCLUDES THE INITIALIZATION CODE AND THE OC2ISR INTERRUPT ROUTINE  *
* WRITTEN BY JENNY AND PARTIALLY BY ERIK.                            *
*********************************************************************
*********************************************************************
*       ZERO PAGE EQUATES                                           *
*********************************************************************
*       A 152 ENTRY TABLE OF 16-BIT VALUES IS RESERVED AT LOCATIONS  *
*       $0000 THROUGH $012F (INDEXED AS 0 THROUGH 151). ENTRY X IS   *
*       FOUND AT LOCATION 2*X IN THE MEMORY MAP. EACH ENTRY IS A     *
*       TURN-OFF BITMAP WITH EACH BIT (BIT 15 TO BIT 0) REPRESENTING *
*       SERVO NUMBER 0 TO 15 RESPECTIVELY. AT A TIME GIVEN BY 164.5uS*
*       PLUS THE QUANTITY [13.5uS * (150-X)], THE CONTROL PULSES TO  *
*       THE SERVOS SPECIFIED BY THE BITMAP AT ENTRY X ARE TURNED OFF.*
*       TABLE_OFFSET IS A POINTER TO THE LAST TABLE VALUE. THE TABLE *
*       IS PROCESSED FROM HIGHEST TO LOWEST INDEX FROM ENTRY $A7 TO  *
*       $00 WITH THE ENTRY A7 BEING A NULL ENTRY USED TO PROPERLY    *
*       ALIGN THE CONTROL PULSES. ATTEMPTS TO WRITE TO THE $97 INDEX *
*       WILL BE REDIRECTED TO THE $96 INDEX.                         *
*********************************************************************
TRENA           EQU     $0C     ; Transmit, Receive ENAble
RDRF            EQU     $20     ; Receive Data Register Full
TDRE            EQU     $80     ; Transmit Data Register Empty
TABLE_OFFSET    EQU     $14A    ; Last record in turn-off table
CHAN1           EQU     $10     ; CHANNELS 1-4
CHAN2           EQU     $14     ; CHANNELS 5-8
ADON            EQU     $80     ; AD POWER UP
ADOFF           EQU     $7F     ; AD POWER DOWN
SPION           EQU     $4C     ; ENABLE SPI
SPIOFF          EQU     $0C     ; DISABLE SPI
SPIDONE         EQU     $80     ; SPI TRANSFER COMPLETE
WAIT_SHORT      EQU     #9      ; 67 us on loop
WAIT_MID        EQU     #248    ; 1501 us in loop
TURN_DELAY      EQU     #1000   ; TURN DELAY
*********************************************************************
*       STANDARD EQUATES                                            *
*********************************************************************
PORTA   EQU     $1000   ; PORT A data register
PORTC   EQU     $1003   ; PORT C data register
PORTB   EQU     $1004   ; PORT B data register
PORTD   EQU     $1008   ; PORT D data register
DDRC    EQU     $1007   ; PORT C direction register
PORTE   EQU     $100A   ; PORT E data register
CFORC   EQU     $100B   ; Timer Compare Force
TOC2    EQU     $1018   ; Timer Output Compare register 2
TOC1    EQU     $1016   ; Timer Output Compare register 1
TOC5    EQU     $101E   ; Timer Output Compare register 5
TCTL1   EQU     $1020   ; Timer ConTroL register 1
TCTL2   EQU     $1021   ; Timer ConTroL register 2
TMSK1   EQU     $1022   ; Timer interrupt MaSK register 1
TFLG1   EQU     $1023   ; Timer interrupt FLaG register 1
RTFLG1  EQU     $23     ; Relative to $1000 Timer interrupt FLaG register 1
TMSK2   EQU     $1024   ; Timer interrupt MaSK register 2
TFLG2   EQU     $1025   ; Timer interrupt FLaG register 2
PACTL   EQU     $1026   ; PULSE ACCUMULATOR CONTROL
OC1D    EQU     $100D   ; OUTPUT COMPARE 1 DATA REGISTER
SPCR    EQU     $1028   ; SPI Control Register
SPSR    EQU     $1029   ; SPI Status Register
SPDR    EQU     $102A   ; SPI Data Register
BAUD    EQU     $102B   ; SCI Baud Rate Control Register
```

```
SCCR1   EQU     $102C   ; SCI Control Register 1
SCCR2   EQU     $102D   ; SCI Control Register 2
SCSR    EQU     $102E   ; SCI Status Register
SCDR    EQU     $102F   ; SCI Data Register
ADCTL   EQU     $1030   ; A/D CONTROL Register
ADR1    EQU     $1031   ; RESULT 1
ADR2    EQU     $1032   ; RESULT 2
ADR3    EQU     $1033   ; RESULT 3
ADR4    EQU     $1034   ; RESULT 4
OPTION  EQU     $1039   ; OPTION Register
TCNT    EQU     $100E   ; TCNT Register
REVWALK1 EQU    $E170   ; Reverse Walk number 1
REVWALK2 EQU    $E2F0   ; Reverse Walk number 2
TURNLEFT EQU    $E470   ; Turning left walk
STARTAD   EQU   $E000   ; Beginning address of walk tables
*********************************************************************
*       REGISTER STRUCTURES                                         *
*********************************************************************
               ORG     $014C
CURRENT_OFF     RMB     16              ; The current turn-off values
ONMASK          RMB     2               ; The mask of selected servos
STORAGE         RMB     2               ; Temporary Storage
STORAGE2        RMB     2               ; Temporary Storage 2
ONWAIT          RMB     2               ; Signal Alignment Variable
WAIT_COUNT      RMB     2               ; Signal Alignment Variable
COUNTER         RMB     2               ; A 16-bit Counter for WAIT function
FLAGS           RMB     1               ; HEADER FLAG
COMPLETE        RMB     1               ; TABLE COMPLETE FLAG
HOLD           RMB    2               ; Temp register
*********************************************************************
*       SETUP AND INITIALIZATION CODE                               *
*********************************************************************
       ORG     $D000           ; $D000 is the beginning of EPROM
START
       LDS     #$01FF          ; Set stack at the top of ram
       LDAA    #$30            ; Set baud to 9600
       STAA    BAUD            ; Set the port baud
       CLR     SCCR1           ; Set mode if indetermined to N81
       LDAA    #TRENA          ; Load mask for Tx, Rx
       STAA    SCCR2           ; Enable the serial subsystem
       LDAA    #$FF            ; Set for output
       STAA    DDRC            ; All port C pins now output
       CLRA                    ; Initialize outputs to
       STAA    PORTC           ; zero to prevent jerking
       STAA    PORTB           ; zero to prevent jerking
       LDAA    #ADON           ; Power on
       STAA    OPTION          ; A/D system
       LDX     #ONMASK+1       ; Start at end of timing tables
CLEARLOOP:
       CLR     0,X             ; Clear the location
       DEX                     ; Move to previous cell
       BNE     CLEARLOOP       ; Keep clearing
       CLR     0,X             ; Clear the $00 location too
*********************************************************************
*       SET UP INTERRUPT FOR OC2                                    *
*********************************************************************
       LDAA    #$40            ; Set up OC2 bitmap
       STAA    TFLG1           ; Clear the interrupt flag register
       STAA    TMSK1           ; Request hardware interrupt sequence
*********************************************************************
       LDX     #$1000          ; Set X to beginning of control registers
INIT_PACTL
       BSET    $26,X  $88      ; Set PA7 for output
PWM_INIT
       BSET    $0C,X  $B0      ; Set OC1 to control OC1, OC3, OC4
       CLR     OC1D            ; All OCx pins go low on a compare of OC1
       LDAA    #$AA            ; Set OC3-4 to go low and OC5, OC2 go low.
       STAA    TCTL1           ; "
* VARIABLE ONWAIT HOLDS THE ITERATION OF THE SERVO TURN-ON CYCLE
* AT WHICH THE ONMASK IS FIRST ASSERTED ONTO THE SERVO PORTS

       LDX     #CURRENT_OFF+14 ; 15 loops desired before turn on
TIMEST
       STX     ONWAIT          ; Store the wait value
       CLR     COUNTER         ; Clear interrupt counter
       CLR     COUNTER+1       ; Counter is 16 bits
```

```
        LDD     #$0000          ; All servos are initially off
        STD     ONMASK          ; This is crucial to make servos not go.
        CLR    FLAGS           ;
        CLI                     ; Turn on interrupts
WOW
         CLR    FLAGS           ;
        LDX     #$8000          ; onmask flag
         STX     HOLD
GETB
        JSR     GETCHAR         ; Get the character from serial port
        CMPA    #$BB            ; Looking for header character (terminal header)
        BNE     GETB           ; Terminal is communicating with processor
READY
        LDX     #CURRENT_OFF    ; X contains the positions table pointer
LOOP
        JSR     GETCHAR         ; Get servo number ($C0-$CF)
        CMPA    #$BF            ;
        BLS     WOW             ;
        ANDA    #$0F            ;
         TAB                    ;
         FCB     $3A             ;
        CLRA                    ;
        XGDY                    ;
        INY                     ;
         TST    FLAGS             ;
        BNE     WOW             ;
STLOO
        DEY                     ;
        BEQ     ENDLOO          ;
         LDD     HOLD             ;
        LSRD                    ;
         STD     HOLD             ;
        BRA     STLOO           ;
ENDLOO
        JSR     GETTIME         ;
        CMPA     #$AA            ; Off directive?
        BEQ     OFF             ; Yes
        TST     FLAGS           ; Test flags
        BNE     WOW             ; Flag was set
        STAA    0,X             ; Put servo position in table
         LDD     HOLD             ; Put onmask flag in D
        ORAA    ONMASK          ; Turn respective servo on
        ORAB    ONMASK+1        ;
        STD     ONMASK          ;
        BRA     WOW             ;
OFF
         TST    FLAGS           ;
        BNE     WOW             ;
         LDD     HOLD             ;
        COMA                    ; Complement
        COMB                    ; Complement
        ANDA    ONMASK          ; Turn off respective servo
        ANDB    ONMASK+1        ;
        STD     ONMASK          ;
        BRA     WOW             ;
* THE FOLLOWING CODE ENABLES SERIAL TRANSMITION BY FORCING OC2
* LOW. IT IS ASSUMED THAT THE OC2 PIN IS WIRED TO AN ENABLE LINE
* ON THE TRANSMITING DEVICE OR THE CTS, DSR, AND DCD PINS OF THE
* CONSOLE RS-232 CABLE. (PINS 5,6 AND 8 RESPECTIVELY) THE ENABLE
* SIGNAL BLOCKS SERIAL COMMUNICATION DURING THE INTERRUPT ROUTINE
* AND IS ACTIVE LOW COMPLYING WITH RS-232 STANDARDS. NOTE THAT THE
* OC2 PIN DOES NOT PRODUCE RS-232 LEVEL OUTPUT AND THE SIGNAL SHOULD
* BE FED INTO A MAX232 OR MC145407P BEFORE BEING SENT OVER A SERIAL
* LINE.
        LDAA    #$80            ; prepare to force interrupt
        STAA    TCTL1           ; line back to low
        LDAA    #$40            ; load bitmap for OC2
        STAA    CFORC           ; force line high
        LDAA    #$C0            ; restore the go-high
        STAA    TCTL1           ; request code
        CLI                     ; Turn on interrupts
**********************************************************************
*       I/O AND TRANSLATION FUNCTIONS                                *
**********************************************************************
**********************************************************************
* ROUTINE CLIP: TAKES A BYTE AND SETS AT $69 IF BETWEEN $69 AND $E0  *
```

```
*              VALUES BETWEEN $E1 AND $FF ARE SET TO $00              *
***********************************************************************
CLIP
        CMPA   #$AA            ; This is the "off" command
       BEQ     GETDONE         ; Done
       CMPA    #$A5             ; $A5 is the maximum position
       BHI     PROS_NEG         ; This number is larger
GETDONE
       RTS                      ; Return value in A
PROS_NEG:
       CMPA    #$E0             ; limit value
       BHI     PUTZERO          ; Greater than $e0
        LDAA    #$A5             ; Limit value to $69
       BRA     GETDONE          ; Get out of routine
PUTZERO
       LDAA    #$00             ; Underflow from subtract
       BRA     GETDONE          ; Done
***********************************************************************
* ROUTINE GETTIME: RETURNS A VALID TIMING VALUE FROM CONSOLE          *
***********************************************************************
GETTIME
       PSHB                     ; SAVE B REGISTER
       JSR     GETCHAR          ; GET A CHARACTER
       BSR     CLIP             ; ASSURE RANGE IS APPROPRIATE
       PULB                     ; RESTORE B REGISTER
       RTS
***********************************************************************
* ROUTINE GETBYTE: CONSTRUCTS A BYTE VALUE FROM TWO ASCII INPUTS      *
***********************************************************************
GETBYTE
       PSHB                     ; SAVE B REGISTER
       BSR     GETCHAR          ; GET A CHARACTER
       BSR     XLATE            ; TRANSLATE TO NIBBLE
       LSLA                     ; TRANSFER TO HIGH NIBBLE
       LSLA                     ; USING FOUR
       LSLA                     ; SUCCESSIVE SHIFTS
       LSLA                     ; TO THE LEFT
       TAB                      ; STORE IN B REGISTER
       BSR     GETCHAR          ; GET THE SECOND HALF
       BSR     XLATE            ; TRANSLATE TO NIBBLE
       FCB     $1B              ; CREATE FULL BYTE
       PULB                     ; RESTORE B REGISTER
       RTS                      ; RETURN BYTE IN A
***********************************************************************
* ROUTINE XLATE: TRANSLATES ASCII CHARACTER INTO NIBBLE               *
***********************************************************************
XLATE   CMPA    #$39            ; IS IT A NUMBER?
       BGT     LETTER           ; TREAT AS LETTER
       ANDA    #$0F             ; GET ABSOLUTE VALUE
       BRA     XDONE            ; FINISHED WITH NUMBER
LETTER  ANDA    #$5F            ; MAKE UPPERCASE
       SUBA    #55              ; ADJUST TO HEX NUMBER
XDONE   RTS                     ; FAIRLY EASY
***********************************************************************
* ROUTINE GETCHAR: GETS BYTE FROM SERIAL PORT AND ECHOS TO CONSOLE   *
***********************************************************************
GETCHAR
       LDAA    SCSR             ; CHECK RECEIVE REGISTER
       ANDA    #RDRF            ; FOR INCOMING CHARACTER
       BEQ     GETCHAR          ; NOT THERE, KEEP TRYING
GETC
       LDAA    SCDR             ; GET THE CHARACTER IN A

       RTS                      ; RETURN CHARACTER
************************************************************************
* SUBROUTINE GETCHARNP: GETS BYTE FROM SERIAL PORT AND DOES NOT ECHO  *
* IT TO CONSOLE. RESULT IS IN ACCUMULATOR A                            *
************************************************************************
GETCHARNP
       LDAA    SCSR             ; CHECK RECEIVE REGISTER
       ANDA    #RDRF            ; FOR INCOMING CHARACTER
       BEQ     GETCHARNP        ; NOT THERE, KEEP TRYING
       LDAA    SCDR             ; GET THE CHARACTER IN A
       RTS                      ; RETURN CHARACTER
************************************************************************
*   SUBROUTINE XDECI: TRANSFORMS 1 BYTE OF HEX IN ACCUMULATOR A       *
```

```
*   INTO DECIMAL NUMBER IN ACCUMULATOR A:                              *
**********************************************************************
XDECI
       PSHB                     ; SAVE B ON STACK
       PSHA                     ; SAVE A ON STACK
       ANDA    #$F0             ; ISOLATE HIGH NIBBLE ON A
       LSRA                     ; MOVE HIGH NIBBLE TO LOW NIBBLE
       LSRA                     ; IN ORDER TO MULTIPLY IT
       LSRA                     ; "
       LSRA                     ; "
       LDAB    #10              ; MULTIPLY CONTENTS OF A WITH 10 IN B
       MUL                      ; "
       PULA                     ; RESTORE A INTO ACCUMULATOR A
       ANDA    #$0F             ; ISOLATE LOW NIBBLE
       FCB     $1B              ; ADD THE TWO AND PUT RESULT IN A

       PULB                     ; RESTORE B
       RTS                      ; DONE
****************************************************************************
*       THE INTERRUPT ROUTINE                                              *
****************************************************************************
* First reset for the next interrupt
OC2ISR
       LDD     #40000           ; 40,000 E's is 20ms
       ADDD    TOC2             ; Add directly to
       STD     TOC2             ; preserve timing accuracy
       LDAA    #$40             ; prepare to clear the
       STAA    TFLG1            ; interrupt flag
       LDD     COUNTER          ; get the current count
       ADDD    #1               ; increment 16-bit value
       STD     COUNTER          ; store into the 20ms counter
* Look at complete table flag
* Take positions from secondary table and put them in primary table
* Clear FLAGS and COMPLETE flags
        LDAA   #1
       STAA    FLAGS
        CLR     COMPLETE
* Process the current servo list
* Now set the new turn-off values
*************************************************************************
* THE LOCATION STORAGE HOLDS A BITMAP WITH A 16-BIT VALUE CORRESPONDING  *
* TO THE CURRENT SERVO BEING PROCESSED. FOR EXAMPLE, THE BITMAP $40000   *
* IS USED TO PROCESS SERVO 1. (0100... RECALL THAT INDEXING STARTS AT    *
* ZERO) ONMASK HOLDS THE 16-BIT BITMAP OF THE SERVOS CURRENTLY DSIRED    *
* TO BE ON. ALL SERVOS THAT ARE ON WILL HAVE THE BITMAP IN STORAGE       *
* APPLIED TO THE ($97-X) ENTRY IN THE TIMING TABLE WHERE X IS THE TIMING *
* VALUE IN THE CORRESPONDING CURRENT_OFF REGISTER. SINCE MORE THAN ONE   *
* SERVO MIGHT HAVE THE SAME TURN OFF TIMING VALUE, THE BITMAP IN STORAGE *
* IS "OR-ED" WITH THE PREVIOUS VALUE TO PREVENT OVERWRITING ANY OTHER    *
* SERVO'S INFORMATION.                                                   *
*************************************************************************
       LDD     #$8000           ; Set active bitmap to servo one
       STD     STORAGE          ; Save the bitmap
       LDD     ONMASK           ; Find out what servos are on
       STD     STORAGE2         ; Working bitmap
       LDY     #CURRENT_OFF-1   ; Get the first servo address
SLOOP
       LDD     0,Y              ; Save the address
       CLRA                     ; Zero for address usage
       LSLD                     ; Offset for 16-bit value
       XGDX                     ; Get address of the servo time register
       LDD     STORAGE2         ; Get active bitmap
       LSLD                     ; Get status in carry flag
       BCS     SERVO_ACTIVE     ; Servo active
* IN THE ONMASK INDICATES THAT THE SPECIFIED SERVO IS ON, IT IS
* PROCESSED AT SERVO_ACTIVE ROUTINE. OTHERWISE, THE SERVO_OFF ROUTINE
* WILL UPDATE THE SELECTION MASK IN STORAGE AND WASTE ENOUGH TIME SO
* AS TO BALANCE THE SERVO_ACTIVE ROUTINE
SERVO_OFF:
       STD     STORAGE2         ; Match delay of the other routine
       LDD     STORAGE          ; Get the active bitmap
       LSRD                     ; Shift for next channel
       STD     STORAGE          ; Store the active bitmap
       TST     0,X              ; Burn 6 cycles
       TST     0,X              ; Burn 6 cycles
       TST     0,X              ; Burn 6 cycles
```

```
        BRA     T_ON_CHECK      ; Now ready to resume routine
SERVO_ACTIVE:
        STD     STORAGE2        ; Store active bitmap
        LDD     STORAGE         ; Restore the active bitmap
        ORAA    0,X             ; Inclusive Or to prevent
        ORAB    1,X             ;   overwriting another servo's
        STAA    0,X             ;   turn-off request.
        STAB    1,X             ;
        LDD     STORAGE         ; Reload bitmap
        LSRD                    ; Set bitmap for next servo channel
        STD     STORAGE         ; Save bitmap
* AT THIS POINT, CHECK IF THE TURN-ON POINT OF THE CHANNELS HAS BEEN
* REACHED. THE CONTROL OF THE TURN-ON POINT IS ACHIEVED BY CHECKING
* FOR A CERTAIN NUMBER OF LOOPS THROUGH THE UPDATE ROUTINE. THE NUMBER
* OF LOOPS BEFORE TURN ON IS GIVEN IN ONWAIT.  AS IT IS NOW, THE TIME
* DELAY BETWEEN TURN ON AND THE BEGINNING OF TURN OFF IS 0.484 MS
T_ON_CHECK:
        INY                     ; Go to next table entry
        CPY     ONWAIT          ; See if X loops done
        BNE     NEXT_SERVO      ; Bypass servo turn on
        LDD     ONMASK          ; Find which servos are active
        STD     PORTC           ; and turn them on
NEXT_SERVO:
        CPY     #CURRENT_OFF+15 ; Done if at this address
        BNE     SLOOP           ; Keep transfering table values
* NOW UPDATE THE LAST_OFF TABLE. SINCE THE TABLES ARE OFFSET BY
* EXACTLY 16 BYTES, THERE IS NO NEED FOR TWO INDEXES. TWO VALUES
* ARE UPDATED AT A TIME, SO ONLY 8 LOOPS ARE REQUIRED.
* THE TURN-OFF LOOP GIVING 13.5uS PER LOOP AT 8MHz
* THIS IS THE TIGHTEST POSSIBLE WAY TO EXECUTE THE TURN OFFS.
* THE TABLE MUST BE PROCESSED BACKWARDS BECAUSE COMPARING THE INDEX
* TO A FINAL VALUE AND BRANCHING CONDITIONALLY TAKES MORE TIME THAN
* DECREMENTING AND BRANCHING ON ZERO. THE ADDITIONAL TURNOFF CYCLE
* IS NECESSARY AFTER THE OFFLOOP BECAUSE THE BRANCH ON ZERO DOESN'T
* PROCESS THE ENTRY AT INDEX ZERO.
TURNOFF
        LDX     #TABLE_OFFSET   ; GET TIMING TABLE ADDRESS
* TIMED LOOP STARTS HERE: 27E'S = 13.5 uS PER LOOP
OFFLOOP
        LDD     0,X             ; GET THE TIMING VALUE
        EORA    PORTC           ; XOR TO TAKE HIGH LINE
        EORB    PORTB           ; LOW AND THEN
        STD     PORTC           ; UPDATE TO SERVO CHANNELS
        DEX                     ; GO TO NEXT 16 BIT
        DEX                     ; TABLE VALUE
        BNE     OFFLOOP         ; IF NOT DONE CONTINUE TABLE
* TIMED LOOP ENDS HERE
        LDD     0,X             ; MUST TO FINAL TABLE VALUE
        EORA    PORTC           ; SINCE A COMPARISON TO ZERO
        EORB    PORTB           ; WAS THE TIGHTEST LOOP
        STD     PORTC           ; POSSIBLE
* CLEAR TURN OFF TABLE
        LDX     #CURRENT_OFF-1  ; Prepare to get servo turn off times
CLEAR_LOOP
        LDD     0,X             ; Prepare to clear value
        CLRA                    ; Zero high byte of address
        LSLD                    ; Adjust for 16 bit value
        XGDY                    ; Get timing address in Y
        CLR     0,Y             ; Now clear location
        CLR     1,Y             ; and the other half
        INX                     ; Go to next location
        CPX     #CURRENT_OFF+15 ; Have all locations been initialized
        BNE     CLEAR_LOOP      ; Keep clearing
* NOW FORCE THE OC2 PIN BACK LOW TO ALLOW SERIAL COMMUNCATION AND
* RESET THE TCTL1 CODE SO THAT THE NEXT INTERRUPT FORCES IT BACK
* HIGH.
        LDAA    #$80            ; prepare to force interrupt
        STAA    TCTL1           ; line back to low
        LDAA    #$40            ; load bitmap for OC2
        STAA    CFORC           ; force line high
        LDAA    #$C0            ; restore the go-high
        STAA    TCTL1           ; request code
        RTI                     ; INTERRUPT DONE
CLEAR   FCB     $1B, $5B, $32, $4A
        FCB     $00
CR      FCB     $0D, $0A, $00
```

```
        ORG     $FFBF
* ANY UNIMPLEMENTED INTERRUPTS ARE RETURNED IMMEDIATELY
BADINT  RTI                     ; ALL UNUSED VECTORS HERE
**********************************************************************
*       INTERRUPT TABLE                                              *
**********************************************************************
        ORG     $FFC0   ; E9 VECTORS START AT $FFC0
        FDB     BADINT  ; RESERVED
        FDB     BADINT  ; RESERVED
        FDB     BADINT  ; RESERVED
        FDB     BADINT  ; RESERVED
        FDB     BADINT  ; RESERVED
        FDB     BADINT  ; RESERVED
        FDB     BADINT  ; RESERVED
        FDB     BADINT  ; RESERVED
        FDB     BADINT  ; RESERVED
        FDB     BADINT  ; RESERVED
        FDB     BADINT  ; RESERVED
        FDB     BADINT  ; SCI Serial System
        FDB     BADINT  ; SPI Serial Transfer Complete
        FDB     BADINT  ; Pulse Accumulator Input Edge
        FDB     BADINT  ; Pulse Accumulator Overflow
        FDB     BADINT  ; Timer Overflow
        FDB     BADINT  ; In Capture 4/Output Compare 5 (TI4O5)
        FDB     BADINT  ; Timer Output Compare 4 (TOC4)
        FDB     BADINT  ; Timer Output Compare 3 (TOC3)
        FDB     OC2ISR  ; Timer Output Compare 2 (TOC2)
        FDB     BADINT  ; Timer Output Compare 1 (TOC1)
        FDB     BADINT  ; Timer Input Capture 3 (TIC3)
        FDB     BADINT  ; Timer Input Capture 2 (TIC2)
        FDB     BADINT  ; Timer Input Capture 1 (TIC1)
        FDB     BADINT  ; Real Time Interrupt (RTI)
        FDB     BADINT  ; External Pin or Parallel I/O (IRQ)
        FDB     BADINT  ; Pseudo Non-Maskable Interrupt (XIRQ)
        FDB     BADINT  ; Software Interrupt (SWI)
        FDB     BADINT  ; Illegal Opcode Trap ()
        FDB     BADINT  ; COP Failure (Reset) ()
        FDB     BADINT  ; COP Clock Monitor Fail (Reset) ()
        FDB     START   ; /RESET
        END
```

Appendix E - High-Level Code

```
/* bobdefs.h */

#define  NUM_SERVOS  12
#define  REPEAT      4

#define  RR_LIFT  0xC0
#define  RR_EXT   0xC2
#define  RR_SWING 0xC1
#define  LR_LIFT  0xC6
#define  LR_EXT   0xC5
#define  LR_SWING 0xC7
#define  RF_LIFT  0xC9
#define  RF_EXT   0xCB
#define  RF_SWING 0xCA
#define  LF_LIFT  0xCE
#define  LF_EXT   0xCF
#define  LF_SWING 0xCD
```

```
/* bob.c - main program */

/*#define      BORLAND*/

#include       "icc2bc.h"
#include       "bobdefs.h"
#include       "posns.c"

void setservo(int servo, int posn)
{
        int   j;

        for(j=0;j<REPEAT;j++){
                putchr(0xBB);
                putchr(code[servo]);
                putchr(posn);
        }
        return;
}

void  UnRise(void)
{
        int   i,k;
        for(k=0;k<MAXRISEPOS;k++){
                for(i=0;i<NUM_SERVOS;i++){
                        setservo(i,risepos[k][i]);
                }
        }
        return;
}

void Rise(void)
{
        int   i,k;
        for(k=MAXRISEPOS-1;k>-1;k--){
                for(i=0;i<NUM_SERVOS;i++){
                        setservo(i,risepos[k][i]);
                }
        }
        return;
}

void UnCrouch(void)
{
        int   i,k;
        for(k=0;k<MAXCROUCHPOS;k++){
                for(i=0;i<NUM_SERVOS;i++){
                        setservo(i,crouchpos[k][i]);
                }
        }
        return;
}

void  Off(void)
{
        int   i,k;
        for(k=MAXCROUCHPOS-1;k>-1;k--){
                for(i=0;i<NUM_SERVOS;i++){
                        setservo(i,crouchpos[k][i]);
                }
        }
        for(i=0;i<NUM_SERVOS;i++){
                setservo(i,0xAA);
        }
        return;
}

void Crouch(void)
{
        int   i,k;
        for(k=MAXCROUCHPOS-1;k>-1;k--){
                for(i=0;i<NUM_SERVOS;i++){
                        setservo(i,crouchpos[k][i]);
                }
        }
        return;
```

```
}

void  Left(void)
{
	int   i,k;
	for(k=MAXLEFTPOS-1;k>-1;k--){
		for(i=0;i<NUM_SERVOS;i++){
			setservo(i,turnleftpos[k][i]);
		}
	}
	return;
}

void  Forward(void)
{
	int   i,k;
	for(k=MAXFWDPOS-1;k>-1;k--){
		for(i=0;i<NUM_SERVOS;i++){
			setservo(i,walkfwdpos[k][i]);
		}
	}
	return;
}

void  Back(void)
{
	int   i,k;
	for(k=0;k<MAXBACKPOS;k++){
		for(i=0;i<NUM_SERVOS;i++){
			setservo(i,walkbackpos[k][i]);
		}
	}
	return;
}

void main()
{
	int   light,IR_front;
	int   mode=0,counter=0;

	UnCrouch();
	while(1){
		switch(mode){			/*movement*/
			case  0:
				Forward();
				counter++;
				if(counter==8){
					mode=1;
					counter=0;
				}
				break;
			case  1:
				Left();
				counter++;
				if(counter==10){
					mode=0;
					counter=0;
				}
				break;
			case   4:
				Back();
				break;
			default:
				break;
		}
		light=analog(7);
		IR_front=analog(5);
		printf("\nlight=%d\tFLIR=%d\tmode=%d\n",light,IR_front,mode);
		switch(mode){				/*sensors*/
			case   0:
			case   1:
				if(light<90){
					mode=3;
					Off();
				}else if(IR_front>120){
					mode=4;
```

```
					Rise();
					msleep(3500);
					UnRise();
				}
				break;
			case  3:
				if(light>165){
					UnCrouch();
					counter=0;
					mode=0;
				}
				break;
			case	4:
				if(IR_front<110){
					mode=1;
				}
				break;
			default:
				break;
		}
	}
}
```