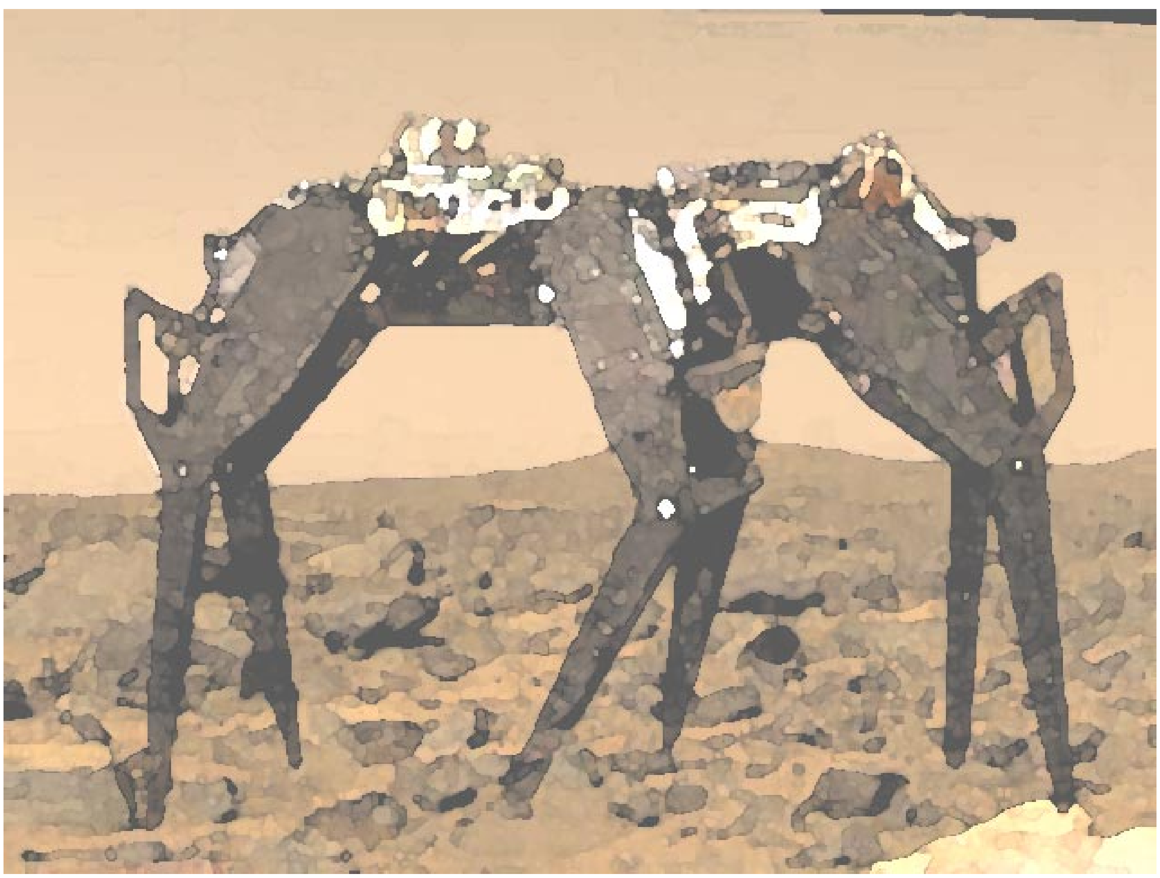
Not So Evil Bug
Six Legged Walker
Intelligent Machines Design Lab
Final Report
Spring 1998



Andor Almasi
April 25, 1998
Prof. Antonio Arroyo

Table of Contents

ABSTRACT

The Not So Evil Bug is a six-legged mobile platform designed to walk and climb/descend high obstacles. The bug is controlled by two HC11s, one for servo control, another for everything else. The current behaviors consist of object-avoidance.

EXECUTIVE SUMMARY

This robot is based on a six-legged platform designed to allow the ascent/descent of tall objects. I designed the platform in AutoCAD and cut it out of plywood on the T-Tech machine. I modeled the legs after the grasshopper's hind legs. They use two servos each, and have two degrees of freedom.

The robot is controlled by two Motorola 68HC11 microprocessors. One of them is housed in a MB2325 board. This processor runs in single-chip mode, and generates the PWM signals controlling the servos. The other HC11 is on the EVBU board, running in expanded mode. This processor runs all the other code, including sensor reading, object avoidance, movement coordination.

I successfully designed and built the platform, and wrote the software that would enable my robot to do everything advertised. Unfortunately, the platform turned out to be way to heavy, and the robot could not support itself while walking. This is a major failure, since the planned behaviors of the robot rely on the specific platform I designed.

INTRODUCTION

Most mobile platforms used for robots employ wheels as a mode of propulsion. Although wheels are a very efficient mode of propulsion, they do have their drawbacks. One of these is their inability to traverse extremely rough terrain. Legged platforms on the other hand do not rely on a constant contact point with the surface, and are thus the better platform to use over rough terrain. I chose to push this idea a bit further and attempted to design a robot that could climb up/down an obstacle about one half the robot's standing height. In order to do this, the robot needed a way to detect an object's height/depth and enter a crawl mode in which it would execute a specialized two-legged walk, instead of the normal three-legged walk. The planned behaviors for the robot are object avoidance and climbing.

HARDWARE DESIGN

The robot hardware includes the body, legs, servos, battery, sensors and microprocessor boards.

Platform design

The robot is based on a six-legged mobile platform. My inspiration for this platform was Robobug, designed/built/programmed by David Novick and Jennifer Laine. I first saw Robobug at a demo in 4744, and thought that it was the coolest thing since (apple pie?) The ideas for my own robot started to develop in the coming weeks, during some oh-so-dull 4712 classes. By the start of this semester, I had a good idea of what I wanted my platform to look like. I did the actual design in AutoCAD version 14. The entire platform is built from wood, cut on the T-tech machine in the Intelligent Machines Design Laboratory lab. The design sheets for the cutouts can be found in appendix A.

Leg design

The most important consideration in designing the legs was to make them long enough to allow the robot to climb over fairly high objects. The robot has six identical legs, two facing forward and four facing toward the rear. The legs have two degrees of freedom, one rotational and one lateral (figure 1). The legs move in a plane parallel to the length of the body. The idea for the general shape and function is most closely

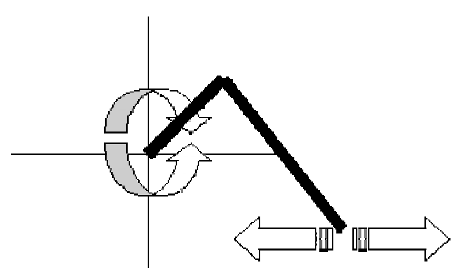


Figure 0 2DOF of the legs

related to the rear leg of a grasshopper (figure 2). The upper leg (thigh) holds the servo actuating the lower leg. It is attached to the body via a servo horn on the inner surface of the thigh. The lower leg is actuated by a linkage system. This linkage is designed to magnify the servo's range of motion. The servo connects to

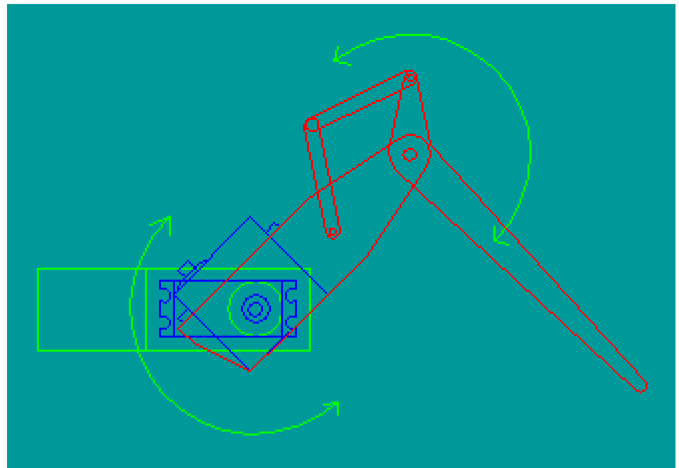


Figure 0 Leg design

the linkage by a length of 1/16th inch diameter piano wire. The leg pieces are designed to easily fit into each other during assembly, and to provide good structural support once built. A tubular joint made from 1/8th inch thick aluminum connects the upper and lower legs. The large diameter of the tubing ensures that the joint will be strong and smooth rolling. I originally considered using ball bearings in the joint, but after testing one of the above joints, I decided the added weight and complexity were not worth the minute gain in smoothness. I had to redesign the legs after the first cutout, because I made the notches that connect the pieces incorrectly. I also made the attachment points for the linkages at the wrong place. The first version of the legs served as a prototype for the redesign, and helped point out some of the pitfalls that I would likely encounter in latter designs. One of these was that the notches that connected the pieces together only fit if I used a new drill bit in the T-tech. Most of the time the bit wear can be neglected, but in this case the effects of bit wear are actually doubled. This is because if the bit is thinner than expected, the notch comes out thicker and the hole it fits into comes out thinner. This leads to a warm relationship with the Dremel tool, which I often used to correct the above problem. A late addition to the legs was the feet (figure 3). They prevent the legs from slipping and serve as touch sensors. Some very tacky rubber (from a lint-removing roller) is glued on the bottom of the feet to ensure a good contact with the ground. The foot surfaces are currently too small and make the feet unstable.

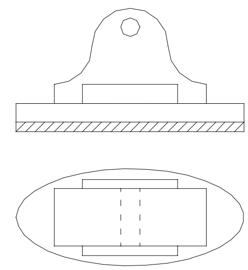


Figure 0 Foot design

Body design

The purpose of the body is to provide an attachment point for the legs and to hold all the other components (microprocessor boards, batteries). The body design is centered around the servos and the battery pack. I built a cardboard prototype of the body to avoid any clearance problems in the final design. Extraneous pieces of wood in the body were cut out to save weight. I added notch attachment points throughout the body to allow for further expansion. The EVBU board rests in the middle of the body, attached by four metal posts. The servo control board rests on the rear of the body, attached by four screws and rubber feet.

Both the legs and the body are painted with flat black spray-paint. This serves as a waterproof barrier and a cosmetic enhancement.

Electronic hardware

The robot uses two microprocessor boards. One is the EVBU board, the other is an MB2325 board. The EVBU board has the 32k memory expansion, two output ports, one input port and a 40kHz signal generator. The EVBU is powered from the MB2325, which has a 5V voltage regulator on it. The MB2325 is directly attached to a 7.2 volt battery pack (6 subC NiCad cells). Leg movement is provided by two servos per leg (one inside the body, one inside thigh). I am using Hitec SuperSport servos purchased from Mayor Hobby (www.majorhobby.com). The servo specifications are listed in table 1. The servos are powered by a 6 volt voltage regulator attached to a heat sink on the bottom of the robot. Two IR emitter-receiver pairs are attached to the front of the robot, as well as sensor switches actuated by whiskers. There are a total of three whiskers, two in a cross configuration pointing forward and one pointing down. The whiskers are made of thin piano wire that bends some on contact. This ensures that nothing breaks when the robot walks into a wall. As an added benefit, the whisker switches activate over a wide range of approach angles. The switch sensors (whiskers and foot) are broken up into groups of three, each driving a voltage-divider network. This network reduces nine digital inputs to three analog inputs.

Torque	49 oz/in @ 6V
Speed	60deg in .17 sec @ 6V
Weight	1.6 oz
Size	1.6 x 0.8 x 1.4”

Table 0 Servo specifications

SOFTWARE DESIGN

Overview

All of the software running on the robot is written in assembly. I chose assembly over IC or ICC because it gives me precise timing control crucial in coordinating leg movements. The software can be broken up into the hierarchical structure

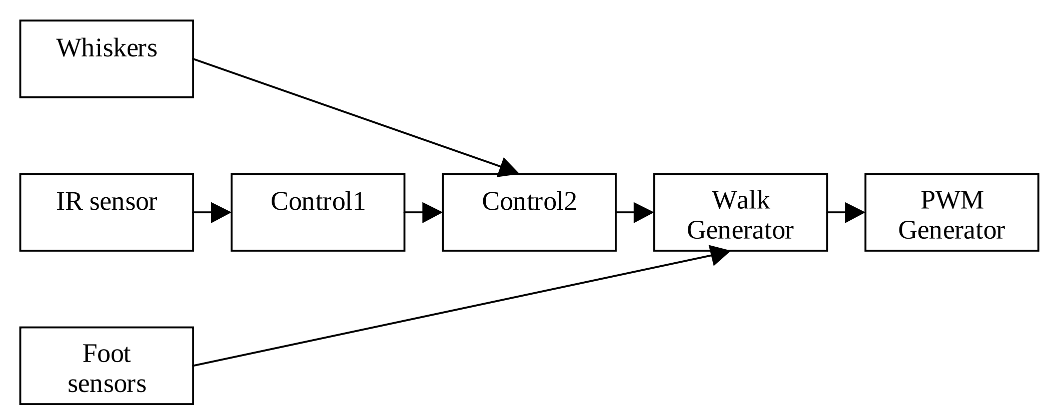


Figure 4 Software Structure

shown in figure 4. The software is broken into three parts. The sensor reading process includes reading all of the sensor inputs and converting them to a more usable form. The control process coordinates movements and behaviors based on sensor outputs. The walk generator process generates the appropriate leg movement sequences for the action requested by the control process. The PWM generator uses those sequences to directly control the servos. All but the PWM generator code is loaded into the SRAM on the EVBU board. The PWM code is running on the MB2325 board, in EPROM. See Appendix B for a complete software listing.

Sensor Reading

This process has the job of reading any and all sensor data and preparing it for use by the control process. For the IR sensors it uses a lookup table to distinguish among five different IR level readings. The final output is in the form of a single 8-bit variable. Four bits are dedicated to each IR sensor. The possible IR readings and their meanings are listed in table 2. This method allows calibration of the IR sensors completely transparently to processes using these readings. The only things that need to be changed in case the sensors need recalibrating are the lookup table values within this procedure. In fact, the IR sensors could be replaced with a completely different type of sensor, without any effect on other methods. T he IR readings are often unreliable, since surface color greatly affects the reflectance of IR. Figure 5 shows the difference in reaction between a white and a dark surface.

0000	
	Bad reading
0001	
	Low reading, nothing ahead
0010	
	High reading, possible obstruction
0100	
	Saturated
1000	

Table 2 IR levels and their meanings

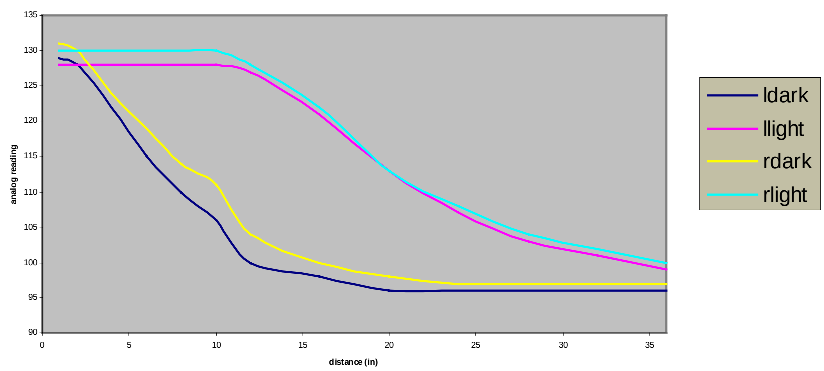


Figure 5 IR readings on different surfaces

The whisker sensors compensate for any errors the IR sensors make. The output format for all the switch sensors is binary. In theory, the analog readings coming from the switch voltage divider network should wiggle one specific bit of the digitized reading for a specific switch. Unfortunately, due to mismatch in resistances this does not work out quite well. Therefore, a lookup table is used to simulate the desired effect. The added benefit of this method is transparency. Once again, any of the foot or whisker sensors could be changed to a different type, without having to change the output.

The foot sensors give feedback about leg position for a dynamically generated walking algorithm. They work the same way as the whiskers above.

My special sensor was going to be an IR range finder. These are used in autofocus cameras to measure the distance between the camera and the subject. They work by emitting a highly localized pulsed IR spot (about six inches diameter at five feet) and measuring the offset of the spot on a specialized IR receiver. This receiver's current output varies according to where on its surface the IR reading is the highest. Lenses focus both the outgoing and incoming IR beams. A decoder chip takes as input the IR emitter frequency and the IR receiver output and converts it to a more usable form. Unfortunately, I have little idea what that form is. I obtained three cameras that had this sensor in them, but could not get any of the manufacturers (Kodak, Canon, and Pentax) to release any useful information. I also tried contacting the decoder manufacturers (Sharp and Hamamatsu Corp) for a datasheet without success. I managed to figure out how the Pentax sensor works. I found that pin 13 on the Sharp IR3S43A decoder emitted a PWM signal whose duty cycle was relative to the distance to the object. The signal varied linearly with distance, duty cycle decreasing until object distance of about 1 foot, increasing for closer and further distances. I was going to use this sensor as an edge-of-the-world detector that would prevent my robot from walking off a table, and allow it to judge if it can walk off a platform or not. Unfortunately while trying to separate the sensor from the rest of the robot I managed to fry some mystery component and could never get the sensor to work again. I did not have any success finding an output on the other two sensors.

Control

The control method evaluates the sensor readings and makes a decision on the next action to send to the legs. The decision process is currently in two phases. In the first phase, IR sensor outputs are evaluated and a recommendation is made to the second phase. The second phase checks the more accurate sensors (whiskers, and depth sensor, if it worked) and makes sure that they do not conflict with the phase 1 recommendation. If so, the recommendation is promoted to an action and is passed on to the walk generator. In case of a conflict, phase 2 reevaluates the possible actions based on its sensor inputs and generates an action. The actions generated by the two control phases emerge into a basic object-avoidance behavior.

Walking

The walk generator takes the action and turns it into a sequence of servo movements. There are two possible walking methods. The first is to use a hardcoded sequence of movements for each action. The second is to dynamically generate movement sequences based on the action and foot switch inputs. The latter allows would allow to self-compensate for uneven terrain, but is much more difficult to implement. I chose to implement the first method.

Actions are commands like walk fwd, turn, climb, etc. An action is made up of sequences. The time delay between two sequences determines the speed of the action. A sequence is 16 servo position values that are fed into the servo controller. Each sequence is sent to the PWM generator five times. A complete action is treated much like a character string. The sequences from the start of the action are sent to the PWM generator until and ENDACT (end-of-action, $FF) character is encountered. Actions are kept as short as possible. This means that a single execution of the walk action, for example, will move the robot forward one step.

Walking is achieved by lifting up three opposing feet, moving them forward and doing the same for the other three feet. Climbing is achieved by moving opposing pairs of feet onto the object to be climbed. I have not found a good turning algorithm for the robot yet. The most likely possible turning method will move one side of the robot less than the other, much like on a tank.

PWM generation

The MB2325 board is running the servo control code. The input format is BBCx$yy, where x is the servo number, yy is the servo position in the range of 00 to A4. The input is passed through the SCI port. The output is a constant

PWM signal on ports B and C. The servo control code was developed and written by Jenny Laine.

CONCLUSION

Accomplishments

I have built and designed the platform, installed the basic object-avoidance sensors, written the object-avoidance code and interfaced to the servo controller.

Failures

The mobile platform turned out to be too heavy. As of now, the robot can stand on all six feet, but falls over as soon as three feet are lifted. This is a major design flaw, and has prevented me from developing all the planned movement actions into the robot. As much as I wish otherwise, I discovered that the laws of physics apply to my robot just as much as everyone else's.

I was not able to get my special sensor working. This is a minor setback, considering that the robot does not walk.

Future work

The next step in the robot's development will be to try a five-legged walk. This involves moving one leg at a time, until a full step is taken. Another alternative is to get some 80 oz/in servos. These would be strong enough to allow the robot to stand stable on three legs.

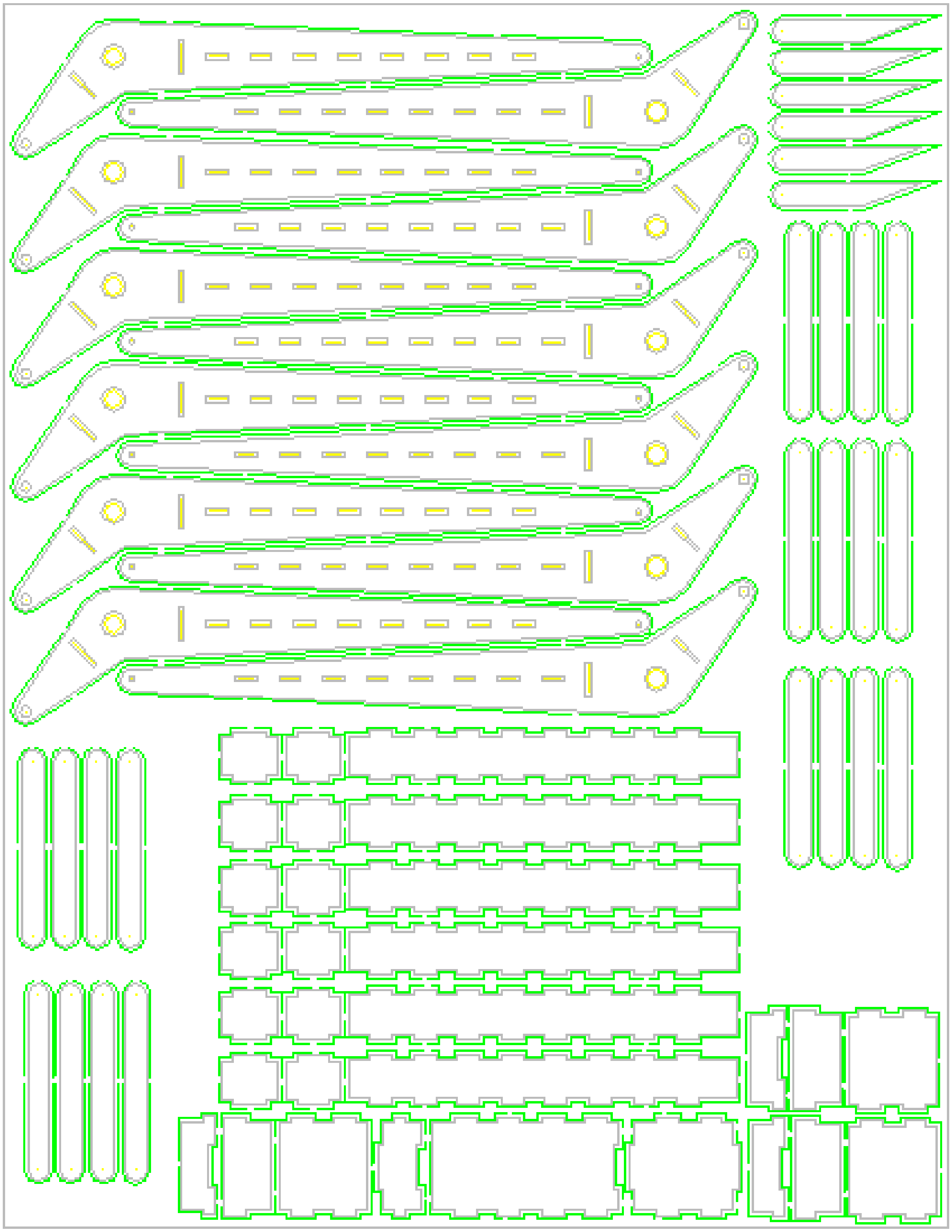
If neither one of these will work, I will be forced to redesign my platform completely.

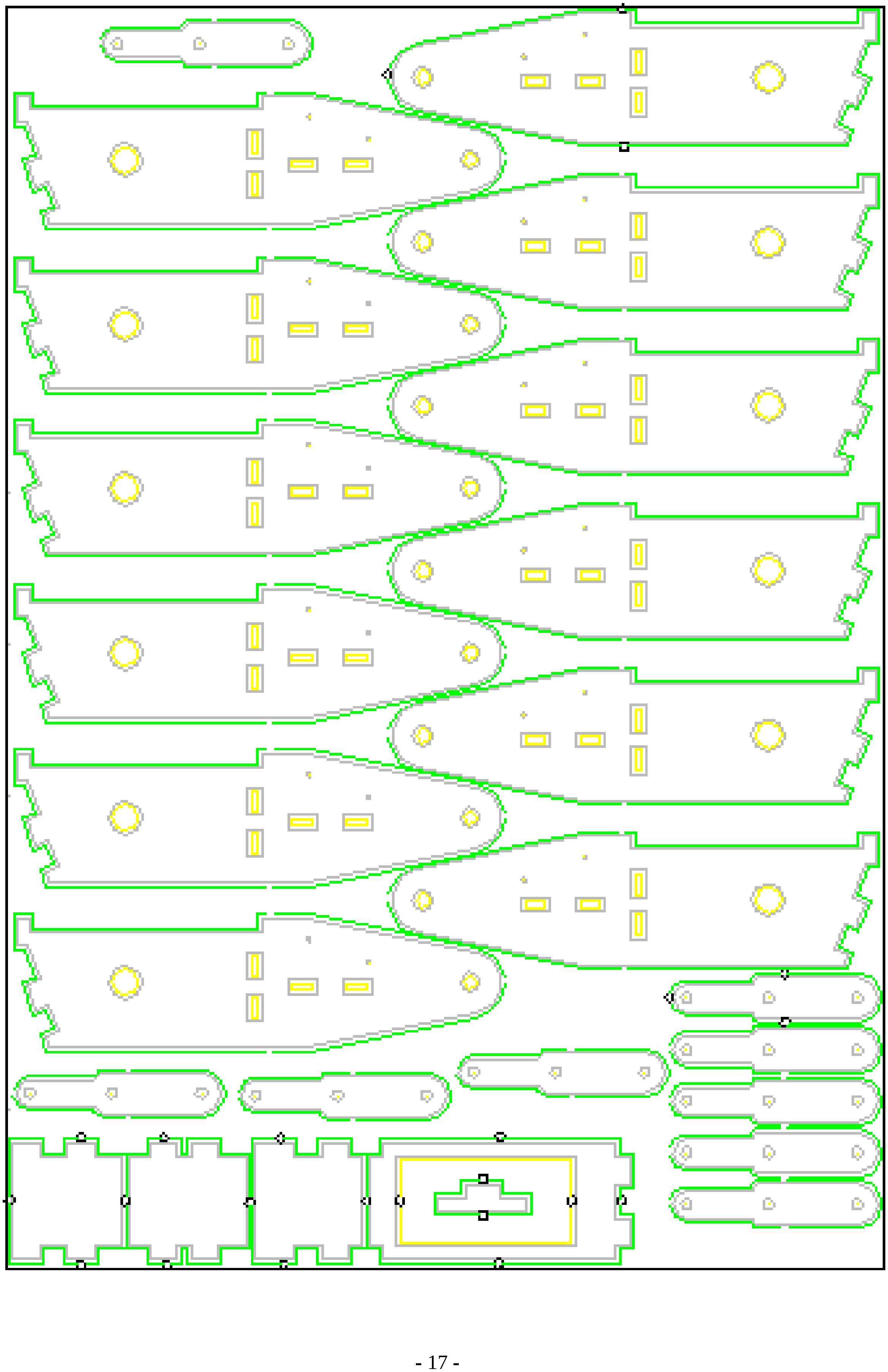
I am also planning to integrate the sensor I developed for my senior project onto the robot platform. This will eliminate the need for my current special sensor.

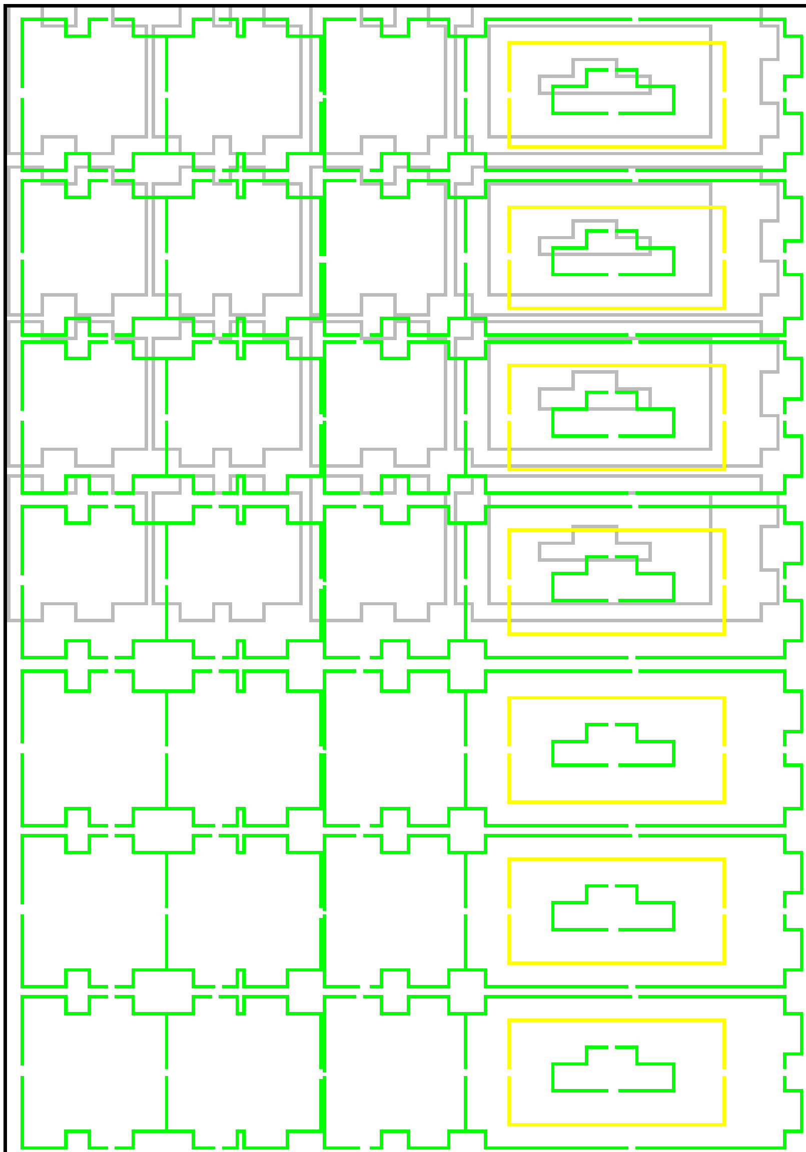
Thanks

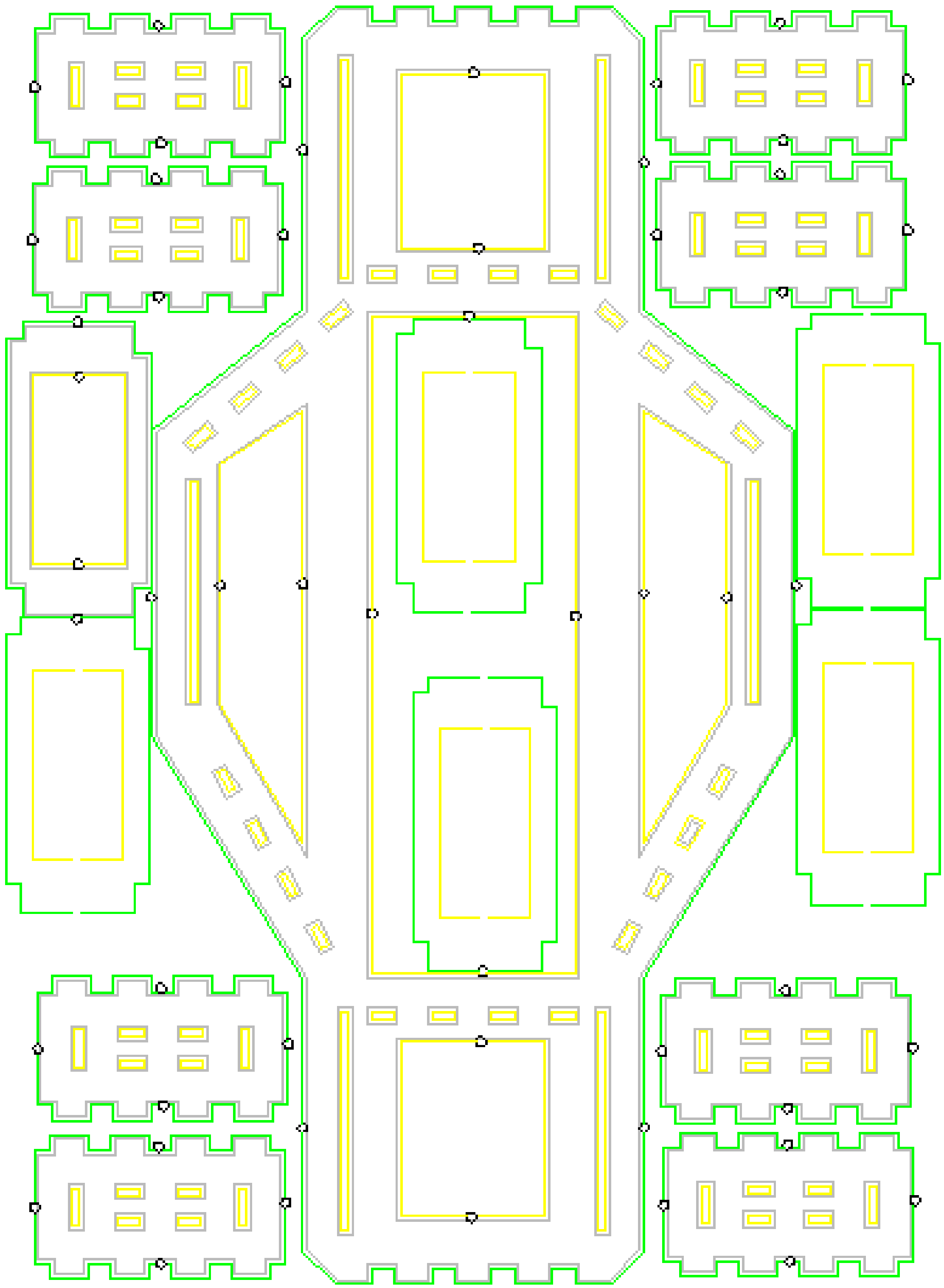
I wish to thank Jenny Laine for developing and letting me use the servo controller code I am running on the MB2325 board.

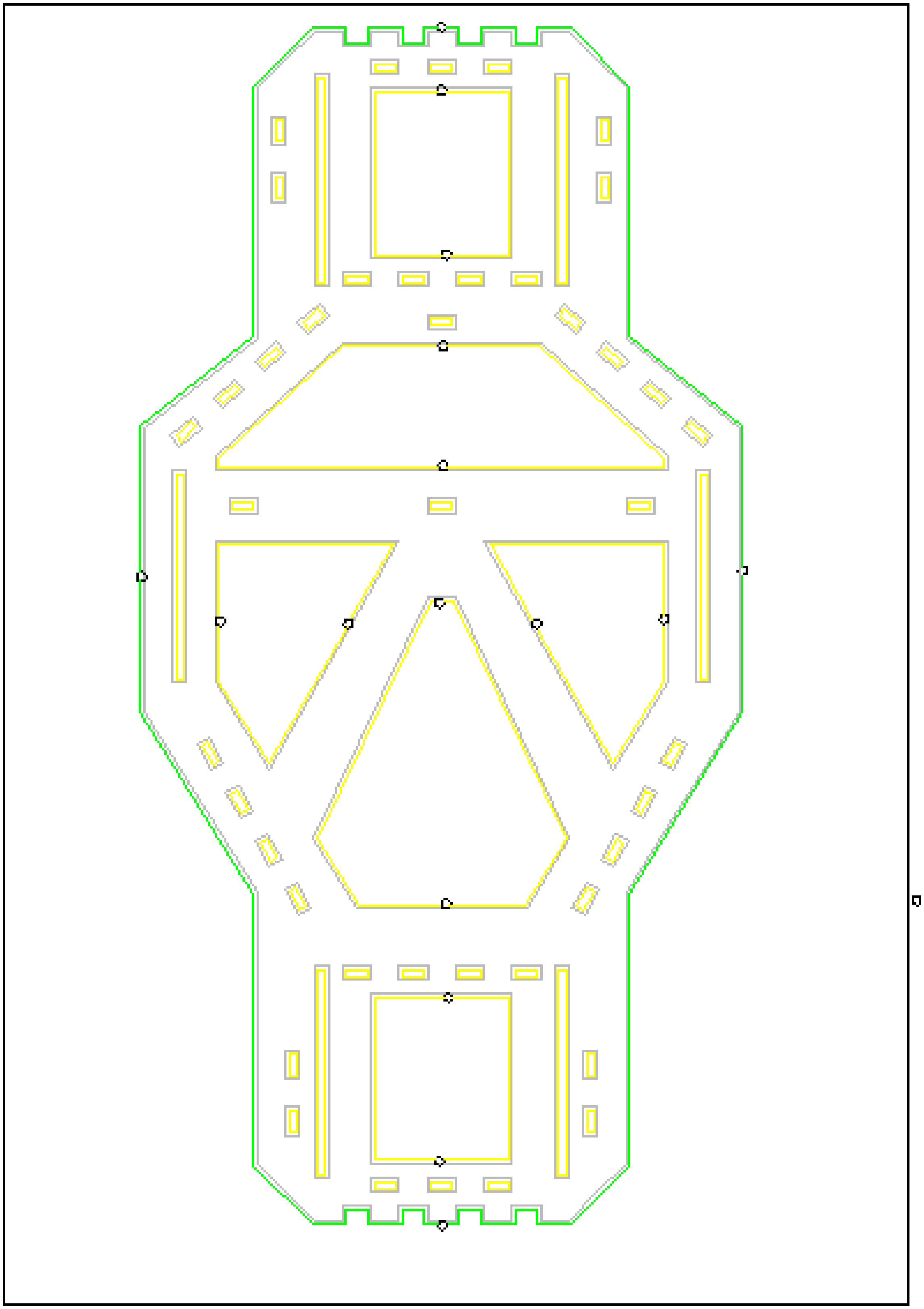
APPENDIX A – Mobile Platform Plans











APPENDIX B – Software source code

Header

```
*< Assembly file header
*  Includes all registers as 8 bit offsets
*  Includes important addresses as 16 bit values
*  Includes single bit masks
*  Place at end of new assembly programs
*>


SRAM    EQU     $2000       ; start of external RAM on EVBU
CRAM    EQU     $0000       ; start of internal RAM
EPROM   EQU     $D000       ; start of eprom
EEPROM  EQU     $B600       ; start of eeprom

BIT7    EQU     %10000000   ; single bit masks
BIT6    EQU     %01000000
BIT5    EQU     %00100000
BIT4    EQU     %00010000
BIT3    EQU     %00001000
BIT2    EQU     %00000100
BIT1    EQU     %00000010
BIT0    EQU     %00000001
INV6    EQU     %10111111   ; inverses
INV5    EQU     %11011111
INV4    EQU     %11101111
INV3    EQU     %11110111
INV2    EQU     %11111011
INV1    EQU     %11111101
INV0    EQU     %11111110

BASE    EQU     $1000       ; register base

ADCTL   EQU     $30         ; A/D Control/Status
ADR1    EQU     $31         ; A/D Result
ADR2    EQU     $32         ; A/D Result
ADR3    EQU     $33         ; A/D Result
ADR4    EQU     $34         ; A/D Result
BAUD    EQU     $2B         ; Baud Rate Control Register
BPROT   EQU     $35         ; Block Protect
CFORC   EQU     $0B         ; Timer Compare Force
CONFIG  EQU     $3F         ; Security disable, COP, ROM Mapping, EEPROM Enables
COPRST  EQU     $3A         ; Arm/Reset COP Timer Circuitry
DDRC    EQU     $07         ; Data Direction Control for Port C
DDRD    EQU     $09         ; Data Direction Control for Port C
EPROG   EQU     $36         ; EPROM Programming Control
HPRIO   EQU     $3C         ; Highest Priority I-Bit Interrupt amd Misc
INIT    EQU     $3D         ; RAM and Register Mapping
OC1D    EQU     $0D         ; Output Compare 1 Data
OC1M    EQU     $0C         ; Output Compare 1 Mask
OPTION  EQU     $39         ; System Configuration Options
PACNT   EQU     $27         ; Pulse Accumulator Control
```

```
PACTL   EQU     $26          ; Pulse Accumulator Control
PIOC    EQU    $02          ; Parallel I/O Control
PORTA   EQU     $00          ; Port A Data
PORTB   EQU     $04          ; Port B Data
PORTC   EQU     $03          ; Port C Data
PORTCL  EQU     $05          ; Port C Latched Data
PORTD   EQU     $08          ; Port D Data
PORTE   EQU     $0A          ; Port E Data
PPROG   EQU     $3B          ; EEPROM Programming Control
SCCR1   EQU     $2C          ; SCI Control 1
SCCR2   EQU     $2D          ; SCI Control 2
SCDR    EQU     $2F          ; SCI Data Register
SCSR    EQU     $2E          ; SCI Status Register
SPCR    EQU     $28          ; Serial Peripheral Control
SPDR    EQU     $2A          ; SPI Data
SPSR    EQU     $29          ; SPI Status Register
TCNT    EQU     $0E          ; Timer Count
TCTL1   EQU     $20          ; Timer Control 1
TCTL2   EQU     $21          ; Timer Control 2
TEST1   EQU     $3E          ; Factory Test
TFLG1   EQU     $23          ; Timer Interrupt Flag 1
TFLG2   EQU     $25          ; Timer Interrupt Flag 2
TIC1    EQU     $10          ; Timer Input Capture 1
TIC2    EQU     $12          ; Timer Input Capture 2
TIC3    EQU     $14          ; Timer Input Capture 3
TIC4    EQU     $1E          ; Timer Input Capture 4
TMSK1   EQU     $22          ; Timer Interrupt Mask 1
TMSK2   EQU     $24          ; Timer Interrupt Mask 2
TOC1    EQU     $16          ; Timer Output Compare 1
TOC2    EQU     $18          ; Timer Output Compare 2
TOC3    EQU     $1A          ; Timer Output Compare 3
TOC4    EQU     $1C          ; Timer Output Compare 4
TOC5    EQU     $1E          ; Timer Output Compare 5
```

Sensor Acquisition

```
*< Title        : Robot sensor data ack prog
* Filename      : sensor.asm
* Programmer    : Andor Almasi
* Date          : Mar 18, 1997
* Version       : 1.0
* Description   : Read in foot, feelers, ir
*>

*<
**************************************************************************
*           Data Section
**************************************************************************
*>
STACK   EQU     $41

                        ORG     CRAM

*< The following are the treshold values for the IR sensors
```

```
* 0000 < BAD < 0001 < LOW <0010 < HI < 0100 <SAT < 1000
*>
BIT76   EQU             %11000000
BIT65   EQU             %01100000
BIT75   EQU             %10100000
BIT765  EQU             %11100000
BIT32   EQU             %00001100
BIT21   EQU             %00000110
BIT31   EQU             %00001010
BIT321  EQU             %00001110

LIR_BAD EQU             $59             ; left reading too low
LIR_LOW EQU             $65             ; left reading far
LIR_HI  EQU             $74             ; left reading close
LIR_SAT EQU             $7E             ; left reading saturated

RIR_BADEQU              $5C             ; right reading too low
RIR_LOW EQU             $68             ; right reading far
RIR_HI  EQU             $74             ; right reading close
RIR_SAT EQU             $7E             ; right reading saturated

IRSENS  RMB             1               ; IR sensor data

*< Whisker sensor data structure
*>
L_C_R   EQU     $90             ; L_C_R > 90            1110xxxx
L_X_R   EQU     $85             ; L_X_R > 85            1010xxxx
X_C_R   EQU     $79             ; X_C_R > 79            1100xxxx
X_X_R   EQU     $65             ; X_X_R > 65            1000xxxx
L_C_X   EQU     $55             ; L_C_X > 55            0110xxxx
L_X_X   EQU     $35             ; L_X_X > 35            0010xxxx
X_C_X   EQU     $20             ; X_C_X > 20            0100xxxx

WHISKER         RMB             1               ; whisker sensor data

*< Foot sensor data structure, same for left & right
*>
R_M_F   EQU     $9C             ; R_M_F > 9C            1110
R_M_X   EQU     $95             ; R_M_X > 95            1100
R_X_F   EQU     $85             ; R_X_F > 85            1010
R_X_X   EQU     $75             ; R_X_X > 75            1000
X_M_F   EQU     $5A             ; X_M_F > 5A            0110
X_M_X   EQU     $40             ; X_M_X > 40            0010
X_X_F   EQU     $20             ; X_X_F > 20            0100

FOOT    RMB             1               ; FOOT sensor data
AD      RMB     8       ; A/D register readings (for debugging)

*<
************************************************************************
*       Define Strings for displaying messages
************************************************************************
*>
Mess1   FCB      LF, CR
                FCC     ' AD: '
```

```
                FCB     EOS

Mess0   FCC     ' Here we go '
                FCB     EOS
*<
****************************************************************************
*               MAIN PROGRAM
* Calls         : InitSCI, OutStr, InChar, OutChar
****************************************************************************
*>
                ORG     SRAM
Main    SEI                     ; turn off interrupt system
                LDS     #STACK          ; Define a stack
                LDX     #BASE
                BSET    OPTION,X BIT7   ; turn on A/D system
                LDAA    #40             ; wait 100 us (200 E)
LGF4    DECA                    ; for A/D to charge
                BNE     LGF4            ;

*< printout code
*>
                JSR     InitSCI         ; init serial Communication
                LDX     #Mess0
                JSR     OutStr
                LDX     #CLS            ;
                JSR     OutStr          ; clear the screen
                LDX     #Mess0
                JSR     OutStr

Again   JSR     ReadSens
                LDX     #Mess1
                JSR     OutStr
                LDX     #FOOT
                JSR     OutByt
                LDX             #WHISKER
                JSR     OutByt
                LDX             #IRSENS
                JSR             OutByt

                LDY     #0
Loop    DEY
                LDAB    #1
Loop2       DECB
                BNE     Loop2
                CPY       #0
                BNE     Loop
                BRA     Again

                SWI             ; return to buffalo

*<
****************************************************
** Subroutine to read sensors
** Raw A/D data is stored in AD
****************************************************
```

```
*>
ReadSens PSHA
                PSHX
                PSHY
                LDX    #BASE
                LDY            #AD                         ; debug

*< These are pins 44-46, IR inputs *
* Here sensor readings are converted into two 4bit
* values stored in IRSENS
*>
                LDAA   #%00010100
                STAA   ADCTL,X
*<    Total delay is 2+((2+2+3)*9) = 65 E's *>
                LDAA   #9        ; 2 E cycles
LOOP1a  NOP            ; 2 E cycles
                DECA            ; 2 E cycles
                BNE    LOOP1a      ; 3 E cycles
                CLR            IRSENS                ; void old sensor reading
                LDAA   ADR1,X              ; the left sensor
                CMPA    #LIR_SAT           ; convert sensor read
                BLO     NEXT1a
                BSET    IRSENS BIT7
                BRA            ENDLa
NEXT1a  CMPA    #LIR_HI
                BLO     NEXT2a
                BSET    IRSENS BIT6
                BRA            ENDLa
NEXT2a  CMPA    #LIR_LOW
                BLO     NEXT3a
                BSET    IRSENS BIT5
                BRA            ENDLa
NEXT3a  CMPA    #LIR_BAD
                BLO     ENDLa
                BSET    IRSENS BIT4
ENDLa   STAA    4,Y                        ; debug
                LDAA   ADR2,X              ; the right sensor
                CMPA    #RIR_SAT           ; convert sensor read
                BLO     NEXT4a
                BSET    IRSENS BIT3
                BRA            ENDRa
NEXT4a  CMPA    #RIR_HI
                BLO     NEXT5a
                BSET    IRSENS BIT2
                BRA            ENDRa
NEXT5a  CMPA    #RIR_LOW
                BLO     NEXT6a
                BSET    IRSENS BIT1
                BRA            ENDRa
NEXT6a  CMPA    #RIR_BAD
                BLO     ENDRa
                BSET    IRSENS BIT0
ENDRa   STAA    5,Y                        ; debug
*< These are pins 45-47-49
*>
```

```
                LDAA    #%00010000
                STAA    ADCTL,X
*<      Total delay is 2+((2+2+3)*18) = 128 E's *>
                LDAA    #18         ; 2 E cycles
LOOP1a  NOP                 ; 2 E cycles
                DECA                ; 2 E cycles
                BNE     LOOP1a      ; 3 E cycles
                CLR             FOOT                ; void old sensor reading
                LDAA    ADR2,X                      ; pin 45, right feet
                CMPA    #R_M_F          ; convert sensor read
                BLO     NEXT1c
                BSET    FOOT BIT321
                BRA             ENDRc
NEXT1c  CMPA    #R_M_X
                BLO     NEXT2c
                BSET    FOOT BIT32
                BRA             ENDRc
NEXT2c  CMPA    #R_X_F
                BLO     NEXT3c
                BSET    FOOT BIT31
                BRA             ENDRc
NEXT3c  CMPA    #R_X_X
                BLO     NEXT4c
                BSET    FOOT BIT3
                BRA             ENDRc
NEXT4c  CMPA    #X_M_F
                BLO     NEXT5c
                BSET    FOOT BIT21
                BRA             ENDRc
NEXT5c  CMPA    #X_M_X
                BLO     NEXT6c
                BSET    FOOT BIT2
                BRA             ENDRc
NEXT6c  CMPA    #X_X_F
                BLO     ENDRc
                BSET    FOOT BIT1
ENDRc   STAA    1,Y                             ; debug
                LDAA    ADR3,X
                CMPA    #R_M_F          ; convert sensor read
                BLO     NEXT1d
                BSET    FOOT BIT765
                BRA             ENDRd
NEXT1d  CMPA    #R_M_X
                BLO     NEXT2d
                BSET    FOOT BIT76
                BRA             ENDRd
NEXT2d  CMPA    #R_X_F
                BLO     NEXT3d
                BSET    FOOT BIT75
                BRA             ENDRd
NEXT3d  CMPA    #R_X_X
                BLO     NEXT4d
                BSET    FOOT BIT7
                BRA             ENDRd
NEXT4d  CMPA    #X_M_F
```

```
                BLO     NEXT5d
                BSET    FOOT BIT65
                BRA             ENDRd
NEXT5d  CMPA    #X_M_X
                BLO     NEXT6d
                BSET    FOOT BIT6
                BRA             ENDRd
NEXT6d  CMPA    #X_X_F
                BLO     ENDRd
                BSET    FOOT BIT5
ENDRd   STAA    2,Y                             ; debug
                LDAA    ADR4,X                  ; pin 43, whiskers
                CLR             WHISKER                 ; void old sensor reading
                CMPA    #L_C_R                  ; convert sensor read
                BLO     NEXT1b
                BSET    WHISKER BIT765
                BRA             ENDWb
NEXT1b  CMPA    #L_X_R
                BLO     NEXT2b
                BSET    WHISKER BIT75
                BRA             ENDWb
NEXT2b  CMPA    #X_C_R
                BLO     NEXT3b
                BSET    WHISKER BIT76
                BRA             ENDWb
NEXT3b  CMPA    #X_X_R
                BLO     NEXT4b
                BSET    WHISKER BIT7
                BRA             ENDWb
NEXT4b  CMPA    #L_C_X
                BLO     NEXT5b
                BSET    WHISKER BIT65
                BRA             ENDWb
NEXT5b  CMPA    #L_X_X
                BLO     NEXT6b
                BSET    WHISKER BIT5
                BRA             ENDWb
NEXT6b  CMPA    #X_C_X
                BLO     ENDWb
                BSET    WHISKER BIT6
ENDWb   STAA    3,Y                             ; debug
                PULY
                PULX
                PULA
                RTS
```

Controller

```
*< Title        : Robot controller prog
* Filename      : control.asm
* Programmer    : Andor Almasi
* Date          : Mar 18, 1997
* Version       : 1.0
* Description   : guide robot based on sensor data
*>
```

```
*<
**********************************************************************
*           Data Section
**********************************************************************
*>
STACK   EQU     $1ff

                        ORG     SRAM
                        JMP                     Main

*< The following are the treshold values for the IR sensors
* 0000 < BAD < 0001 < LOW <0010 < HI < 0100 <SAT < 1000
*>
BIT76       EQU                 %11000000
BIT65       EQU                 %01100000
BIT75       EQU                 %10100000
BIT765      EQU                 %11100000
BIT32       EQU                 %00001100
BIT21       EQU                 %00000110
BIT31       EQU                 %00001010
BIT321      EQU                 %00001110

LIR_BAD EQU                     $59                 ; left reading too low
LIR_LOW EQU                     $65                 ; left reading far
LIR_HI      EQU                 $74                 ; left reading close
LIR_SAT EQU                     $7E                 ; left reading saturated

RIR_BADEQU                      $5C                 ; right reading too low
RIR_LOW EQU                     $68                 ; right reading far
RIR_HI      EQU                 $77                 ; right reading close
RIR_SAT EQU                     $81                 ; right reading saturated

IRSENS  RMB                     1                   ; IR sensor data

*< Whisker sensor data structure
*>
L_C_R   EQU     $90                 ; L_C_R > 90                1110xxxx
L_X_R   EQU     $85                 ; L_X_R > 85                1010xxxx
X_C_R   EQU     $79                 ; X_C_R > 79                1100xxxx
X_X_R   EQU     $65                 ; X_X_R > 65                1000xxxx
L_C_X   EQU     $55                 ; L_C_X > 55                0110xxxx
L_X_X   EQU     $35                 ; L_X_X > 35                0010xxxx
X_C_X   EQU     $20                 ; X_C_X > 20                0100xxxx

WHISKER         RMB             1                   ; whisker sensor data

*< Foot sensor data structure, same for left & right
*>
R_M_F   EQU     $9C                 ; R_M_F > 9C                1110
R_M_X   EQU     $95                 ; R_M_X > 95                1100
R_X_F   EQU     $85                 ; R_X_F > 85                1010
R_X_X   EQU     $75                 ; R_X_X > 75                1000
X_M_F   EQU     $5A                 ; X_M_F > 5A                0110
```

```
X_M_X   EQU     $40             ; X_M_X > 40            0010
X_X_F   EQU     $20             ; X_X_F > 20            0100

FOOT    RMB             1               ; FOOT sensor data
AD      RMB   1     ; A/D register readings (for debugging)
AD1             RMB             1
AD2             RMB             1
AD3             RMB             1
AD4             RMB             1
AD5             RMB             1
AD6             RMB             1
AD7             RMB             1

ACTION  RMB             2               ; address of action stored here
SPEED   RMB             2               ; address of speed of action

FAST    EQU             $8000
SLOW    EQU             $0000


Hello   FCC             'hello.... '
                FCB             EOS
STAND   FCC             '< STAND IN PLACE >'
                FCB             EOS
FWD             FCC             '< WALK FORWARD >'
                FCB             EOS
BWD             FCC             '< WALK BACKWARD >'
                FCB             EOS
TURNL   FCC             '< TURNING LEFT >'
                FCB             EOS
TURNR   FCC             '< TURNING RIGHT >'
                FCB             EOS
SHARPL FCC              '< SHARP LEFT >'
                FCB             EOS
SHARPR FCC              '< SHARP RIGHT >'
                FCB             EOS
CLIMB   FCC             '< CLIMB >'
                FCB             EOS

*<
****************************************************************************
*               MAIN PROGRAM
****************************************************************************
*>
Main   SEI              ; turn off interrupt system
                LDS    #STACK       ; Define a stack
                LDX    #BASE
                BSET   OPTION,X BIT7   ; turn on A/D system
                LDAA   #40          ; wait 100 us (200 E)
LGF4   DECA             ; for A/D to charge
                BNE    LGF4         ;


*< printout code
*>
```

```
                JSR    InitSCI       ; init serial Communication
                LDX    #CLS           ;
                JSR    OutStr         ; clear the screen
                LDX    #Hello
                JSR    OutStr

Again   JSR    ReadSens
                JSR                  WhatToDo
                BRA    Again


*<
*****************************************************
** Subroutine to decide action based upon sensor
** readings
*****************************************************
*>

WhatToDo PSHA
                PSHB
                PSHX
                PSHY

                LDX                  SLOW
                STX                  SPEED                              ; default speed is slow

                BRCLR   0,X %00001111 STAND_R   ; bad reading, stop
                BRCLR   0,X %11110000 STAND_R   ; bad reading, stop
                BRSET   0,X %10001000 BACK_R    ; back up
                BRSET   0,X %10000000 SHR_R             ; sharp right
                BRSET   0,X %00001000 SHL_R             ; sharp left
                BRSET   0,X %01000000 TR_R              ; turn right
                BRSET   0,X %00000100 TL_R              ; turn left
                BRSET   0,X %00100000 SFWD_R    ; forward slow
                BRSET   0,X %00000100 SFWD_R    ; forward slow
                BRSET   0,X %00010000 FFWD_R    ; forward fast
                BRSET   0,X %00000001 FFWD_R    ; forward fast

STAND_R         LDX                  #STAND                     ; store recommendation
                STX                  ACTION
                BRA                  CHK_WH
BACK_R LDX              #BWD                            ; store recommendation
                STX                  ACTION
                BRA                  CHK_WH
SHR_R   LDX             #SHARPR                         ; store recommendation
                STX                  ACTION
                BRA                  CHK_WH
SHL_R   LDX             #SHARPL                         ; store recommendation
                STX                  ACTION
                BRA                  CHK_WH
TR_R    LDX             #TURNR                          ; store recommendation
                STX                  ACTION
                BRA                  CHK_WH
TL_R    LDX             #TURNL                          ; store recommendation
                STX                  ACTION
```

```
                BRA             CHK_WH
FFWD_R LDX              #FWD                            ; store recommendation
                STX             ACTION
                LDX             FAST                            ; store speed
                STX             SPEED
                BRA             CHK_WH
SFWD_R LDX              #FWD                            ; store recommendation
                STX             ACTION
CHK_WHLDX               #WHISKER
*< these are the final word in the action that will be taken *>
                BRCLR   0,X %11100000 EXEC      ; execute IR recommendation
                BRSET   0,X %10100000 BACK      ; go back
                BRSET   0,X     %10000000 SHR   ; sharp right
                BRSET   0,X %00100000 SHL       ; sharp left
                BRSET   0,X     %01000000 CLMB ; climb
BACK    LDX             BWD                             ; store action
                STX             ACTION
                BRA             EXEC
SHL             LDX             SHARPL                          ; store action
                STX             ACTION
                BRA             EXEC
SHR             LDX             SHARPR                          ; store action
                STX             ACTION
                BRA             EXEC
CLMB    LDX             CLIMB                           ; store action
                STX             ACTION

EXEC    JSR             Move

                PULY
                PULX
                PULB
                PULA
                RTS



*<
************************************************************************
*                       leg control subroutine (temporary one)
* input:    ACTION - start of sequence to execute
*                       SPEED - delay between sequence steps
************************************************************************
*>

Move    PSHA
                PSHB
                PSHX
                PSHY

                LDX             ACTION
                JSR             OutStr
                LDY             SPEED   ; delay before going to next seq
Loop    DEY                             ;
                LDAB    #$15    ;
```

```
Loop2   DECB                    ;
        BNE     Loop2           ;
        CPY     #0              ;
        BNE     Loop            ;

        PULY                    ; outta here
        PULX
        PULB
        PULA
        RTS

*<
***************************************************
** Subroutine to read sensors
** Raw A/D data is stored in AD
***************************************************
*>
ReadSens PSHA
        PSHX
        PSHY
        LDX     #BASE
        LDY     #AD                     ; debug

*< These are pins 44-46, IR inputs *
* Here sensor readings are converted into two 4bit
* values stored in IRSENS
*>
        LDAA    #%00010100
        STAA    ADCTL,X
*<      Total delay is 2+((2+2+3)*9) = 65 E's *>
        LDAA    #9              ; 2 E cycles
LOOP1a  NOP                     ; 2 E cycles
        DECA                    ; 2 E cycles
        BNE     LOOP1a          ; 3 E cycles
        CLR     IRSENS                  ; void old sensor reading
        LDAA    ADR1,X                  ; the left sensor
        CMPA    #LIR_SAT                ; convert sensor read
        BLO     NEXT1a
        BSET    IRSENS BIT7
        BRA     ENDLa
NEXT1a  CMPA    #LIR_HI
        BLO     NEXT2a
        BSET    IRSENS BIT6
        BRA     ENDLa
NEXT2a  CMPA    #LIR_LOW
        BLO     NEXT3a
        BSET    IRSENS BIT5
        BRA     ENDLa
NEXT3a  CMPA    #LIR_BAD
        BLO     ENDLa
        BSET    IRSENS BIT4
ENDLa   STAA    4,Y                     ; debug
        LDAA    ADR2,X                  ; the right sensor
        CMPA    #RIR_SAT                ; convert sensor read
        BLO     NEXT4a
```

```
                BSET    IRSENS BIT3
                BRA             ENDRa
NEXT4a  CMPA    #RIR_HI
                BLO     NEXT5a
                BSET    IRSENS BIT2
                BRA             ENDRa
NEXT5a  CMPA    #RIR_LOW
                BLO     NEXT6a
                BSET    IRSENS BIT1
                BRA             ENDRa
NEXT6a  CMPA    #RIR_BAD
                BLO     ENDRa
                BSET    IRSENS BIT0
ENDRa   STAA    5,Y                             ; debug
*< These are pins 45-47-49
*>
                LDAA    #%00010000
                STAA    ADCTL,X
*<    Total delay is 2+((2+2+3)*18) = 128 E's *>
                LDAA    #18         ; 2 E cycles
LOOP1a  NOP             ; 2 E cycles
                DECA            ; 2 E cycles
                BNE     LOOP1a      ; 3 E cycles
                CLR             FOOT            ; void old sensor reading
                LDAA    ADR2,X                  ; pin 45, right feet
                CMPA    #R_M_F          ; convert sensor read
                BLO     NEXT1c
                BSET    FOOT BIT321
                BRA             ENDRc
NEXT1c  CMPA    #R_M_X
                BLO     NEXT2c
                BSET    FOOT BIT32
                BRA             ENDRc
NEXT2c  CMPA    #R_X_F
                BLO     NEXT3c
                BSET    FOOT BIT31
                BRA             ENDRc
NEXT3c  CMPA    #R_X_X
                BLO     NEXT4c
                BSET    FOOT BIT3
                BRA             ENDRc
NEXT4c  CMPA    #X_M_F
                BLO     NEXT5c
                BSET    FOOT BIT21
                BRA             ENDRc
NEXT5c  CMPA    #X_M_X
                BLO     NEXT6c
                BSET    FOOT BIT2
                BRA             ENDRc
NEXT6c  CMPA    #X_X_F
                BLO     ENDRc
                BSET    FOOT BIT1
ENDRc   STAA    1,Y                             ; debug
                LDAA    ADR3,X
                CMPA    #R_M_F          ; convert sensor read
```

```
                BLO     NEXT1d
                BSET    FOOT BIT765
                BRA             ENDRd
NEXT1d  CMPA    #R_M_X
                BLO     NEXT2d
                BSET    FOOT BIT76
                BRA             ENDRd
NEXT2d  CMPA    #R_X_F
                BLO     NEXT3d
                BSET    FOOT BIT75
                BRA             ENDRd
NEXT3d  CMPA    #R_X_X
                BLO     NEXT4d
                BSET    FOOT BIT7
                BRA             ENDRd
NEXT4d  CMPA    #X_M_F
                BLO     NEXT5d
                BSET    FOOT BIT65
                BRA             ENDRd
NEXT5d  CMPA    #X_M_X
                BLO     NEXT6d
                BSET    FOOT BIT6
                BRA             ENDRd
NEXT6d  CMPA    #X_X_F
                BLO     ENDRd
                BSET    FOOT BIT5
ENDRd   STAA    2,Y                             ; debug
                LDAA    ADR4,X                  ; pin 43, whiskers
                CLR             WHISKER                         ; void old sensor reading
                CMPA    #L_C_R                  ; convert sensor read
                BLO     NEXT1b
                BSET    WHISKER BIT765
                BRA             ENDWb
NEXT1b  CMPA    #L_X_R
                BLO     NEXT2b
                BSET    WHISKER BIT75
                BRA             ENDWb
NEXT2b  CMPA    #X_C_R
                BLO     NEXT3b
                BSET    WHISKER BIT76
                BRA             ENDWb
NEXT3b  CMPA    #X_X_R
                BLO     NEXT4b
                BSET    WHISKER BIT7
                BRA             ENDWb
NEXT4b  CMPA    #L_C_X
                BLO     NEXT5b
                BSET    WHISKER BIT65
                BRA             ENDWb
NEXT5b  CMPA    #L_X_X
                BLO     NEXT6b
                BSET    WHISKER BIT5
                BRA             ENDWb
NEXT6b  CMPA    #X_C_X
                BLO     ENDWb
```

```
                BSET    WHISKER BIT6
ENDWb   STAA    3,Y                             ; debug
                PULY
                PULX
                PULA
                RTS
```

Movement generator

```
*< Title       : Robot movement coordination
* Filename     : move.asm
* Programmer   : Andor Almasi
* Date         : Apr 22, 1997
* Version      : 1.0
* Description  :
*>



*<
************************************************************************
*           Data Section
************************************************************************
*>
STACK   EQU     $1ff

                ORG     SRAM
                JMP       Main

*<
************************************************************************
*           Define Strings for displaying messages
************************************************************************
*>
TEMP1   RMB             2
Spd             FCC             ' Enter servo speed: '
                FCB             EOS

SERVO   RMB             1               ; current servo count
SPEED   RMB             2               ; time between movements
ACTION  RMB             2               ; which action to perform
ENDACT EQU              $FF             ; end action delimiter
ROT_ADJ         EQU             $5A             ; servo rotation adjustment factor

*< posi+tion tables, hold sequences of positions for 16 servos
* a set of sequences form an action
* body part notation [LEFT/RIGHT][FRONT/CENTER/BACK][UPPER/LOWER]
*
* servo number  c0  c1  c2  c3  c4  c5  c6  c7  c8  c9  ca  cb  cc  cd  ce  cf
*<inverse:               NO  NO  NO  NO  NO  NO              YES YES YES YES YES YES *>
*<body part:        LFL LFU RCL RCU RRL RRU N/C N/C N/C N/C RFL RFU LCL LCU LRL LRU *>
STAND   FCB             $30,$35,$40,$30,$30,$35,$00,$00,$00,$00,$30,$35,$40,$30,$30,$35
                FCB             ENDACT
```

```
*<body part:	LFL LFU RCL RCU RRL RRU N/C N/C N/C N/C RFL RFU LCL LCU LRL LRU *>
FWD	FCB	$40,$50,$50,$20,$30,$35,$00,$00,$00,$00,$30,$35,$40,$30,$50,$20
	FCB	$40,$50,$50,$20,$20,$25,$00,$00,$00,$00,$40,$20,$30,$30,$50,$20
	FCB	$50,$40,$40,$40,$20,$25,$00,$00,$00,$00,$40,$20,$30,$30,$20,$40

	FCB	ENDACT

*<body part:	LFL LFU RCL RCU RRL RRU N/C N/C N/C N/C RFL RFU LCL LCU LRL LRU *>
BWD	FCC	'< WALK BACKWARD >'
	FCB	ENDACT

*<body part:	LFL LFU RCL RCU RRL RRU N/C N/C N/C N/C RFL RFU LCL LCU LRL LRU *>
TURNL	FCC	'< TURNING LEFT >'
	FCB	ENDACT

*<body part:	LFL LFU RCL RCU RRL RRU N/C N/C N/C N/C RFL RFU LCL LCU LRL LRU *>
TURNR	FCC	'< TURNING RIGHT >'
	FCB	ENDACT

*<body part:	LFL LFU RCL RCU RRL RRU N/C N/C N/C N/C RFL RFU LCL LCU LRL LRU *>
SHARPL FCC	'< SHARP LEFT >'
	FCB	ENDACT

*<body part:	LFL LFU RCL RCU RRL RRU N/C N/C N/C N/C RFL RFU LCL LCU LRL LRU *>
SHARPR FCC	'< SHARP RIGHT >'
	FCB	ENDACT

*<body part:	LFL LFU RCL RCU RRL RRU N/C N/C N/C N/C RFL RFU LCL LCU LRL LRU *>
CLIMB	FCC	'< CLIMB >'
	FCB	ENDACT


*<
*************************************************************************
*		MAIN PROGRAM
*************************************************************************
*>
Main	SEI		; turn off interrupt system
	LDS	#STACK	; Define a stack
	LDX	#BASE

	LDAA	#$2000
	STAA	$b900	; notify user to switch

*< printout code
*>
	JSR	InitSCI	; init serial Communication

*	LDX	#Spd
*	JSR	OutStr
*	JSR	InByt
*	JSR	InByt
*	LDD	TEMP1
*	STD	SPEED
```

```
*               LDX             #TEMP1
*               JSR             OutByt
*               JSR             OutByt

                LDAA    #0
                STAA    $b900   ; notify user to switch
                LDY             #0              ; delay gives time to switch
Loope   DEY                             ;
                LDAB    #$15    ;
Loop2e  DECB                    ;
                BNE     Loop2e  ;
                CPY     #0                      ;
                BNE     Loope   ;

                LDX             #$3000
                STX             SPEED

                LDAA    #5
Again   LDX             #STAND
                STX             ACTION
                JSR     Move
                DECA
                BNE             Again

KeepOn  LDX             #FWD
                STX             ACTION
                JSR     Move
                BRA     KeepOn


*<
*************************************************************************
*                       leg control subroutine
* input:    ACTION - start of sequence to execute
*                       SPEED - delay between sequence steps
*************************************************************************
*>

Move    PSHA
                PSHB
                PSHX
                PSHY

                LDX             ACTION

NEXT_SQ         LDAA    #$c0
                STAA    SERVO   ; current servo



NEXT_SV         LDAA    SERVO   ;
                CMPA    #$d0    ; is it beyond last servo
                BEQ             END_S   ;

                LDAB    #5              ; send same thing five times
```

```
SendOvr LDAA    #$BB    ; header
                JSR             OutChar ; header sent
                LDAA    SERVO   ; servo number
                JSR             OutChar ; servo num sent
                LDAA    0,X             ; servo position
                CMPA    #ENDACT         ; is it end of sequence?
                BEQ             END_A   ; outta here

                LDY             #SERVO
                BRSET   0,Y BIT3 NoAdj  ; rotation adjustment check
                LDAA    #ROT_ADJ                ; load adjustment factor
                SUBA    0,X             ; adjusted position
NoAdj   JSR             OutChar ; position sent
                DECB                    ;
                BNE             SendOvr ;

                INC             SERVO   ; next servo
                INX                             ; next position
                BRA             NEXT_SV         ; do same for next servo

END_S   LDY             SPEED   ; delay before going to next seq
Loop    DEY                             ;
                LDAB    #$15    ;
Loop2   DECB                    ;
                BNE     Loop2   ;
                CPY      #0             ;
                BNE     Loop    ;
                BRA             NEXT_SQ         ; do next sequence

END_A   PULY                    ; outta here
                PULX
                PULB
                PULA
                RTS
```

SCI system

```
*< Title        : SCI system functions
* Filename      : sci.asm
* Programmer    : Andor Almasi
* Date          : Feb 15, 1997
* Version       : 1.0
* Description   : InitSCI, OutChar, OutStr, InChar, OutXY, MakeBCD
*>

*<************************************************************************
*       Common definitions, assumes that header.asm is already included
*************************************************************************>

*<OutChar, OutStr, OutXY*>
CLS     FCB     ESC,$5B,$32,$4A         ; ANSI sequence to clear screen
                FCB     ESC,$5B,$3B,$48         ; and move cursor to home
                FCB     EOS             ; EOS character
```

```
*<OutChar, OutStr, OutXY, InChar*>
EOS     EQU     $04            ; User-defined End Of String (EOS) character
CR      EQU     $0D            ; Carriage Return Character
LF      EQU     $0A            ; Line Feed Character
ESC     EQU     $1B            ; Escape Character
SP      EQU     $20            ; Space Character

*<************************************************************************
*                   SUBROUTINE - InitSCI
* Description: This subroutine initializes the BAUD rate to 9600 and
*         sets up the SCI port for 1 start bit, 8 data bits and
*         1 stop bit.  It also enables the transmitter and receiver.
*         Effected registers are BAUD, SCCR1, and SCCR2.
* Input        : None.
* Output       : Initializes SCI.
* Destroys     : None.
* Calls        : None.
*************************************************************************
*<Baud rate defs:     $30    9600
*                 $31    4800
*                 $32    2400
*                 $33    1200
*                 $34    0600
*                 $35    0300
*                 $36    0150
*                 $37    0075 *>
RATE    EQU     $30
InitSCI PSHA                ; Save contents of A register
        LDY     #BASE
        LDAA    #RATE       ; Set BAUD rate
        STAA    BAUD,Y
        CLR     SCCR1,Y     ; Set SCI Mode to 1/8/1
        LDAA    #$0C        ; Enable SCI Transmitter
        STAA    SCCR2,Y     ;     and Receiver
        PULA                ;Restore A register
        RTS                 ; Return from subtoutine
*<
*************************************************************************
*                   SUBROUTINE -  OutChar
* Description: Outputs the character in register A to the screen after
*           checking if the Transmitter Data Register is Empty.
* Input        : Data to be transmitted in register A.
* Output       : Transmit the data.
* Destroys     : None.
* Calls        : None.
*************************************************************************
*>
OutChar PSHB                ; Save contents of B register
        LDY     #BASE
Loop1   LDAB    SCSR,Y      ; Check status reg (load it into B reg)
        ANDB    #$80        ; Check if transmit buffer is empty
        BEQ     Loop1       ; Wait until empty
        STAA    SCDR,Y      ; Register A ==> SCI data
        PULB                ; Restore B register
        RTS                 ; Return from subtoutine
```

```
*<
************************************************************************
*                    SUBROUTINE -  OutStr
* Description: Outputs the string terminated by EOS. The starting
*         location of the string is pointed by X register. Calls
*         the OutChar subroutine to display a character on the screen
*         and exit once EOS has been reached.  In order to print the
*         string properly with RTI, it automatically disables and
*         enables interrupts.
* Input         : Starting location of the string to be transmitted
*             : (passed in X register)
* Output        : Prints the string.
* Destroys      : Contents of X register.
* Calls         : OutChar.
************************************************************************
*>
OutStr  PSHA                 ; Save contents of A register
                        LDY    #BASE
                        SEI                  ; Disable interrupts
Loop2   LDAA    0,X           ; Get a character (put in A register)
                        CMPA    #EOS         ; Check if it's EOS
                        BEQ     Done         ; Branch to Done if it's EOS
                        JSR     OutChar      ; Print the character by calling OutChar
                        INX                  ; Increment index
                        BRA     Loop2        ; Branch to Loop2 for the next char.
Done    CLI                  ; Enable interrupts
                        PULA                 ; Restore A register
                        RTS                  ; Return from subtoutine
*<
************************************************************************
*              SUBROUTINE  -  InChar
* Description: Receives the typed character into register A.
* Input         : None
* Output        : Register A = input from SCI
* Destroys      : Contents of Register A
* Calls         : None.
************************************************************************
*>
InChar  LDX           #BASE
                        LDAA    SCSR,X    ; Check status reg.
                        ANDA    #$20      ; Check if receive buffer full
                        BEQ     InChar    ; Wait until data present
                        LDAA    SCDR,X    ; SCI data ==> A register
                        RTS             ; Return from subroutine

*<***************************************************************************
* OutByt - convert the byte at X to two
* ASCII characters and output. Return X pointing
* to next byte.
* This is from the buffalo source code
****************************************************************************
*>


OutByt  PSHA
```

```
                LDAA 0,X            ;get data in a
                PSHA                ;save copy
                BSR  OUTLHLF        ;output left half
                PULA                ;retrieve copy
                BSR  OUTRHLF        ;output right half
                PULA
                INX
                RTS

OUTLHLF LSRA                 ;shift data to right
                LSRA
                LSRA
                LSRA
OUTRHLF ANDA #$0F            ;mask top half
                ADDA #$30           ;convert to ascii
                CMPA #$39
                BLE  OUTA           ;jump if 0-9
                ADDA #$07           ;convert to hex A-F
OUTA    JSR  OutChar         ;output character
                RTS


*<
**********************************************************************
* InByt - reads two ascii numbers and converts them to hex,
* returns them in TEMP + 1, shifting TEMP+1 to TEMP
* Uses buffalo function HEXBIN (modified)
*
**********************************************************************
*>
InByt   JSR         InChar
                JSR         HEXBIN
                JSR         InChar
                JSR     HEXBIN
                RTS
*<
*****************
*   HEXBIN(a) - Convert the ASCII character in a
* to binary and shift into TEMP1.  Assumes correct hex input
*****************
*>

HEXBIN  PSHA
                PSHB
                PSHX
                JSR     UPCASE          ; convert to upper case
                CMPA    #'0'
                BLT     HEXRTS          ; jump if a < $30
                CMPA    #'9'
                BLE     HEXNMB          ; jump if 0-9
                CMPA    #'A'
                BLT     HEXRTS          ; jump if $39> a <$41
                CMPA    #'F'
                BGT     HEXRTS          ; jump if a > $46
                ADDA    #$9                     ; convert $A-$F
```

```
HEXNMBANDA      #$0F            ; convert to binary
                LDX     #TEMP1
                LDAB    #4
HEXSHFT ASL     1,X                     ; 2 byte shift through
                ROL     0,X             ; carry bit
                DECB
                BGT     HEXSHFT                 ; shift 4 times
                ORAA    1,X
                STAA    1,X
HEXRTS PULX
                PULB
                PULA
                RTS

*<
*****************
*  UPCASE(a) - If the contents of A is alpha,
* returns a converted to uppercase.
*****************
*>
UPCASE  CMPA #'a'
        BLT  UPCASE1     jump if < a
        CMPA #'z'
        BGT  UPCASE1     jump if > z
        SUBA #$20        convert
UPCASE1  RTS
```