ARGO: Automated Transport

John Ferrara
Intelligent Machine Design Laboratory
EEL 5666
3 August 1998

Table of Contents

Abstract

The following is a discussion of ARGO, and an automated chauffeur. Accomplishment of this task involved the use of infrared sensors and emitters, light/voltage converters, and bump sensors. The robot uses a multitasking environment in which to run several processes simultaneously. These processes are overseen by a command arbiter that determines the current behavior. The most developed behaviors that ARGO possesses are hi differential obstacle avoidance, and lane following. Both use "fuzzy" logic to accomplish smooth driving.

Executive Summary

ARGO was developed to be an autonomous chauffeur of the future. The robot is constructed from a modified RC-car: bumpers, and a sensor hood have been added. The robot uses the Motorola 68HC11 microprocessor to multitask 14 separate processes, as well as the entire interrupt system—taken advantage of through the assembly language. These processes constitute the behavioral patterns of the robot. Some of which are lane following, obstacle avoidance, and light finding.

These algorithms were designed using a fuzzy logic system that ensures smooth turning and smooth transitions between different speeds. The obstacle avoidance has been refined to allow the robot to avoid obstacles at a relatively high rate of speed.
To accomplish these tasks, the robot uses five infrared emitter/detector pairs, two bump sensors, and two light/voltage converters, a high current(100+amps) motor, and a steering servo. When used in conjunction the robot displays very good obstacle avoidance behavior, and decent lane following behavior.

ARGO uses a self-designed h-bridge motor driver that allows the car to operate at nearly a continuous range of speeds between stop and full speed. In addition, ARGO, has a software rest that enables the user all the benefits of the BUFFALO operating system plus the added benefits of special bootstrap mode.

Introduction

The world is becoming increasingly congested. An increasing number of people are taking to the highways to travel, resulting in a higher fatality rate due to automotive relative accidents. What the world needs is something safer, something that can react almost immediately to changing roadway conditions. Herein lies ARGO: the automated chauffeur.

Based upon the Motorola 68HC11 EVBU board, complimented with the ME11 board, ARGO uses an assembly written program to multitask his way around obstacles, and to stay between lane markers. His superior digital motor driver and fuzzy logic steering enables him to act and react just like an automobile.

Herein lies the story of ARGO.

Integrated Systems

The foundation upon which ARGO derives his behavior is a multitasking system. Allotment of a small time slice to each of the processes running on the microprocessor yields the appearance that each process is running independent of the rest. This infrastructure allows for the easy addition of multiple behaviors, and for the seamless interaction among all of the processes. This interaction is facilitated by the addition of a command arbiter: an overseeing process that reviews each processes' recommendation to change the physical state of the robot—its direction and speed—and effectively prioritizes each of the processes. This hierarchy of competition to change ARGO's motion dictates the external behavior.

Before ARGO may effectively interact in his environment, all of his systems must be calibrated. The calibration begins after, and every time, the "soft" boot is initiated. Once calibrated the multitasking system is initialized with all of the process present on ARGO. This system will continually cycle through the two motor control processes, the two servo control processes, the sensor reading, reset control (software reset), lane following, obstacle avoidance, human following (human requires an ir-emitter), light following, and horn control. Once this occurs the command arbiter invokes, according to the environment, different processes to produce varying behaviors.

A schematic of this hierarchy of processes is shown in Figure 1.

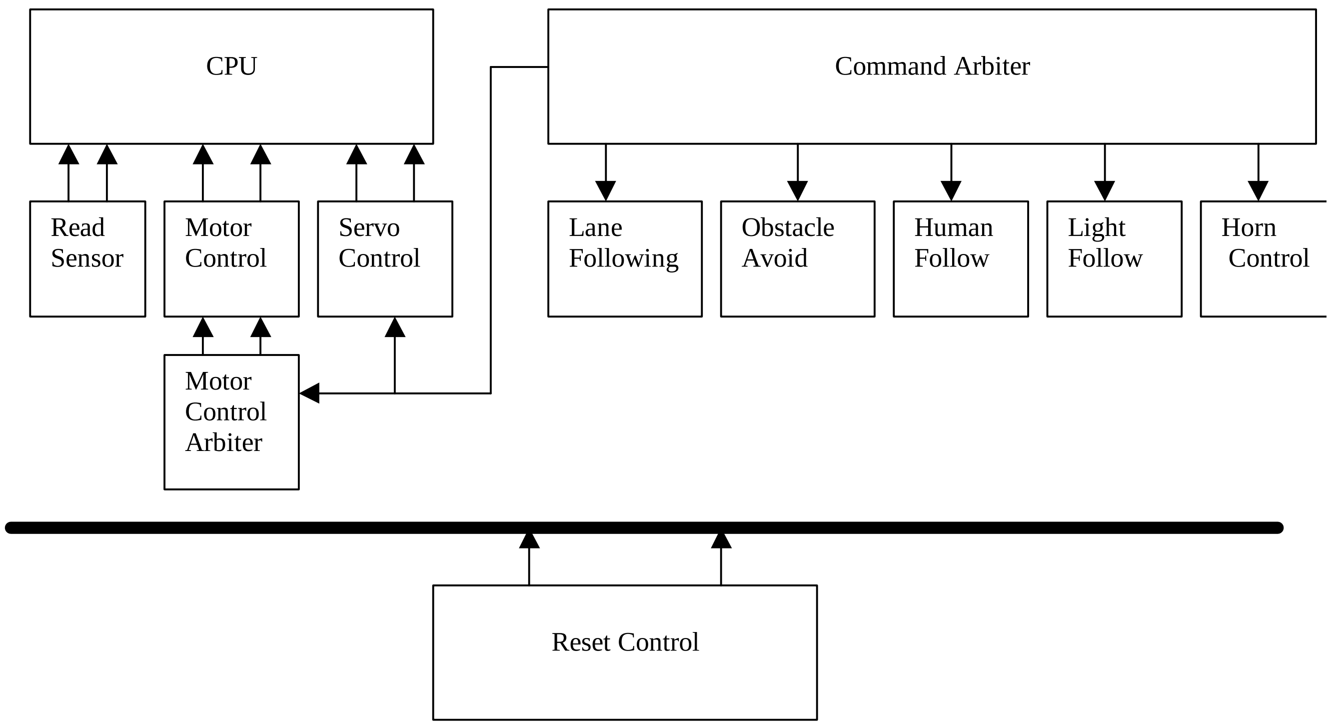


Figure 1

Mobile Platform

The platform upon which ARGO is based is a childhood RC-car. Since my proposal required a sturdy construction and I, lacking mechanical skills and time in which to complete all objectives in addition to the carpentry, resolved to use this car as a foundation for ARGO's construction. Although much of the platform would remain unchanged, room was required in which to mount the sensors, the microprocessor, the batteries, and any other switches and accessories (horn) required by my design.

After removing all transmitters used for remote control, I proceeded to construct a wooden sensor "hood." Upon this hood I secured the Motorola 68HC11 EVBU board, two battery packs, switches used to control reset and power, horn, and infrared transmitters and receivers. In addition, the platform lacked any real front bumper. Since, I would require a bumper to mount most of my sensors—IR, light, and bump—I decided to construct a sturdy bumper from PVC pipe, securing it to my car with screws and a piece of 2-by-4. Into this pipe I was easily able to mount three IR-receivers, emitters, bump sensors, and lane following sensors.

See Figure 2 for a diagram of the overall platform design.

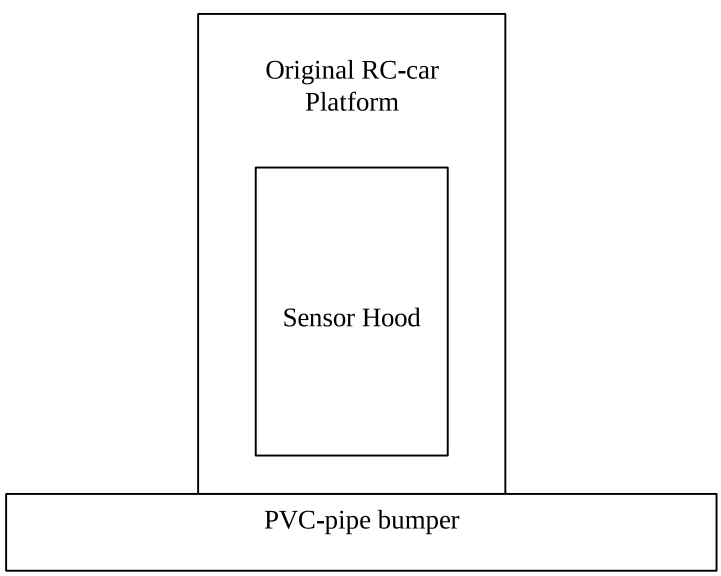


Figure 2

The front bumper construction provided a much needed "hub" upon which I was able to mount most of the sensors necessary for lane following and object avoidance. After numerous testing, the bumper proved to be as strong as it was useful, able to weather a plethora of high-speed crashes.

Due to the age of the RC-platform—approximately 10 years—and overuse, the suspension had become extremely flexible. The robot would sit .25-inches above the ground, producing many problems with the ir-system—reflection from the floor—and practically eliminating the possibility of mounting light sensors on the bumper. Thus, I locked the bumper in statum, by filling the shock's oil-chamber with hot glue. This worked quite well, providing me with more than 2-inches of clearance from the ground.

Once I performed these modifications to the platform, I was able to develop, nearly unfettered, most of the electrical systems needed to control all of ARGO's behaviors.

Actuation

Bringing all of ARGO's behaviors to fruition would require the use of two actuators: a steering servo, and a motor to control the speed. Control of the servo proved to be extremely easy, nearly problem-free. The motor, however, became one of the greatest hurdles to overcome.

Steering Servo

To steer the robot I used a standard servomechanism. The position of this servo depends upon the pulse width applied to the input. A period of 20ms and a pulse width varying from 1ms to 2ms move the servo full left and full right, respectively. Any degree of movement may be achieved by varying the pulse-width between these two boundaries.

I connected the servo directly to the microprocessor via the output compare port. Writing code to pulse-width modulate this output compare pin—see appendix for assembly listing---I was able to accurately describe the motion of the servo in any position necessary.

Motor Control

The task of implementing obstacle avoidance and lane following on ARGO would require extremely precise control of the motor speed on the robot. The analog speed controller used by the RC-car provided only 3 speeds (fast, faster, and 50 mph). Thus, a complete redesign of that circuit was necessary, to enable me to have digital control of the motor was necessary.

To achieve this goal, I used a high-power MOSFET h-bridge construction. In addition, since this circuit would require two output compares and two digital outputs to control direction, I designed additional control circuitry enabling me to use one output compare and one direction signal. I coupled this control logic to the gates of the MOSFETs using opto-isolators. A complete diagram of this circuit is shown in the appendix.

Accurate control of the motor proved to be one of the most difficult tasks I overcame. That I could not accurately rate the stall current of the motor—the stall current far exceeded any measuring tools at my disposal, I was forced to redesign many motor control circuits that, in retrospect, would have worked had I implemented it with the appropriate MOSFETs. After realizing that the stall current of the high-speed motor I was working with exceeded 100-Amps, I was able to design a circuit that accurately controlled the speed of the motor over a nearly continuous range between full-on and completely-off.

Lessons (Funny?)

One of the numerous lessons I learned while designing this circuit, is that the gate voltage must be equal to or greater than the source voltage. Failure to heed this characteristic yields fiery results—burns evidence this property. My original design had biased the gate with only 5V, while biasing the source at 7.2V. Once I established that this would not work, I redesigned the circuit to use an opto-isolator between the control logic and the gate. This would step-up the logic level of 5V to the battery voltage of 7.2V.

Although this seems relatively simple, I was forced to use the opto-isolator in two configurations: one for the n-type, and p-type transistors. I pulled the gate voltage from the emitter of the BJT when biasing the N-type MOSFETs so that I could get 0.00V to turn them fully off. Otherwise, a logic level—from the inverter—of .02 would translate, through the opto-isolator, to about .6V. This produced a short to ground, burning up the transistor.

Conversely, to correctly bias the P-type MOSFETs, I needed to pull the gate voltage from the collector—inverting the signal to the diode of the OI-- to get a voltage equal to exactly the battery voltage to turn them completely off. Otherwise, buy the same logic, a short to ground would be produced.

Sensors

To bring ARGO to life he must have the ability to interact with his environment; that is, he needs to have some way of "feeling" his environment. A robot must be able to read certain data—such as temperature, lighting, imminent obstacles, and color—and interpret this data, allowing the robot to make intelligent decisions based on this data. ARGO will be no different, and herein I will endeavor to characterize all of ARGO's current sensor developments, as well as explain any future work in this area.

ARGO uses five infrared transmitters and detectors; mounted on his front bumper and sensor hood, they allow him to detect and to avoid obstacles, and follow an ir-beacon. In addition, I developed a sensor that will allow ARGO to detect yellow lane markers on the road's surface.

IR-Sensors

The infrared sensor package includes the following: infrared emitters, emitting infrared light modulated at 40kHz; a SHARP infrared detector, detecting reflected infrared light.

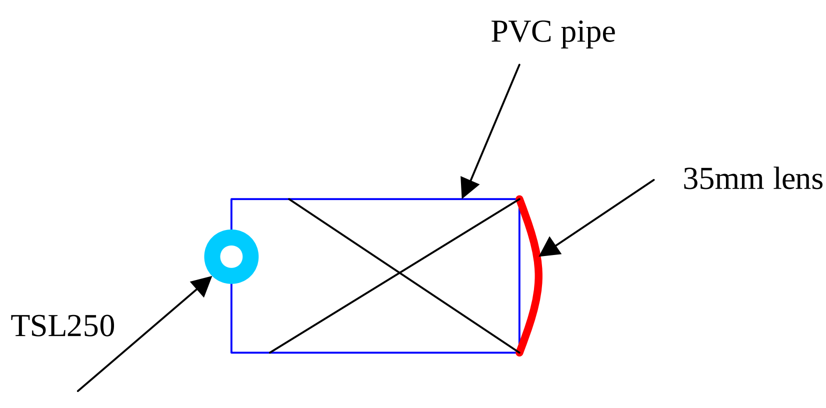
To detect an object the light emitted from the ir-emitters is reflected by a nearby object and detected by the SHARP detectors

Light Sensors

Since ARGO is designed to be a self-driving robot, he must have the capability to detect lane markers and upon detection, follow these lanes. To accomplish this task I received free samples of a Texas Instruments light-to-voltage converter, theTSL250. Before modifying this device, I thoroughly tested the sensor. The results are summarized under Experimental Results.

Realizing that this sensor would not be sufficient for my purpose: lane detection, i.e. black color detection, I sought to implement the sensor in a mechanism that would mimic the eye; that is, focus light that is approximately 6-inches from the robot. To do this, I fitted a 35mm camera lens to a piece of 40mm PVC pipe, placing the TSL250 just beyond the focal length of the lens. This would produce a blurred image of anything toward which the lens was directed: a drawing of the design—which I have dubbed the ARGO EYE--is as follows:

The "ARGO EYE"



After constructing this device, I again tested it in the same manner as before; the data tables and graphs (see experimental results) depict these results. Upon comparison of the two characteristics, I realized that the “eye” produced a response that was dependent upon where the lens was focused. Although the voltage level changed depending upon the color focused upon, changes in the ambient lighting had a dramatic effect on the voltage level, even from one side of the car to the other. Thus, I was forced to eliminate the effect of the ambient light through the implementation of lights strategically positioned beside each of the lane sensors. This produces a response that could be used by my program, since the voltage level now remained relatively constant throughout the room.

Bump Sensors

ARGO uses two bump sensors mounted on the front bumper, interconnected with a thin piece of metal to allow bump sensor triggering from a “hit” between the two sensors. I input these two sensors through an OR-gate to port D input. I did this so that a bump on either of the two sensors would alert the microprocessor that an object had passed through the ir-detection grid.

The circuit used to implement this is trivial: simply a bump sensor connected to power, ground, and port D through a pull-up resistor.

Behaviors

Obstacle Avoidance

Under normal conditions obstacle avoidance would be nearly trivial; however, at the high speeds at which ARGO can travel, obstacle avoidance becomes a daunting task. To accomplish this task, I implemented a type of differential object avoidance in conjunction with “fuzzy” logic code.

The differential object avoidance code looks for a **change** in the level of any of the ir-receivers. When a substantial change is detected the program calculates the difference between the values at the left and right receivers and uses this value to search through a table of predefined ir-differences. When an entry in the table is greater than or equal to that passed into the subroutine, a match is found. The location in this table is used as an index into another table of servo control pulse-widths. The pulse-width is found, and the servo is turned correspondingly. Since the table contains five left and five right turn directions, I have created fuzzy logic with 10-levels.

In addition to turning the servo, the obstacle avoidance code also checks to be sure that the robot has not come too close to an object. If this does occur, however, the program will reverse the motor, putting the car into reverse, and backing away from the object. Then with a slower initial speed, the robot is able to negotiate the object.

Lane-Following

The lane following, due to the primitive nature of the sensor being used, is also slightly rudimentary. Since the sensor cannot see the lane until the car is directly on top of the lane marker, a full turn away from the lane is required to avoid passing over the lane. Thus, the code follows: whenever a lane is detected, the car turns full away from it. While this is rather simple, it works well at slow speeds.

To improve this code I would require another light/voltage converter placed in the middle of the car to sense if the lane had been passed over. In this manner I could greatly speed up the car without risk out straying far from the lanes. However, time restriction proved too great to permit the ordering of another sensor.

Light Following

Having installed light sensors to detect the lanes, and having seen their extreme sensitivity to light, making the robot move toward a light source was a simple task. All that this process does, when invoked by the command arbiter, is compare the two voltages levels at each of the lane sensors, and head toward the higher of the two. Although simple, it accomplishes the task.

Human Following

After I had refined my obstacle avoidance code, and increased the current output to the ir-emitters, I had left two obsolete ir-detectors. I decided rather than take them off, I could use them to implement another behavior: following an ir-beacon held be a human. The code compares the two ir-receiver voltages, and turns the car toward the higher sensor reading. This direction would be toward the person holding the beacon.

Horn Honking

Since I am striving to model reality with my robot in every way that I can, implementing a horn seemed like the obvious decision. This process checks the ir-values to see if they pass beyond a certain threshold. If they do, the program sounds the horn: two short beeps.

Smooth Motor Control

After having written the fuzzy logic for the servo mechanism, reproducing it with some slight modifications was not extremely difficult. The program, one of those being multitasked, looks for the largest ir-value—it uses a modular subroutine to do this—and uses this to search through a table of predefined ir-levels. After finding the matching proximity, the program then uses the entry in this table to index into a table of motor speeds. The motor speed is retrieved and changed by the motor control process. This fuzzy logic system has 5-levels.

Experimental Layout and Results

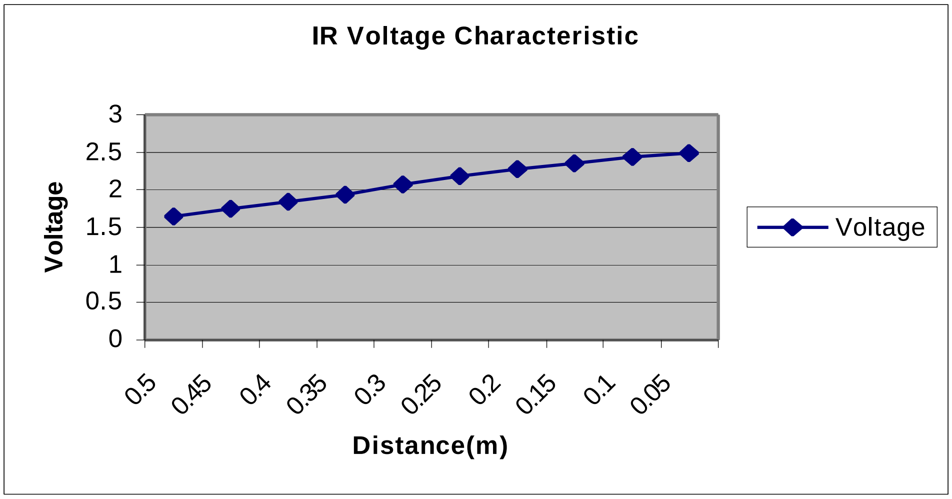
IR-Testing

The voltage dependency on distance, as indicated by table T.A1, varies linearly.

Through software ingenuity and by boosting the output current to the ir-LEDs, I was able to use these sensors for obstacle avoidance, even at relatively high speeds.

IR Characteristic Table

Distance(m)	Voltage	Analog Output
0.5	1.64	84
0.45	1.75	89
0.4	1.84	94
0.35	1.93	98
0.3	2.06	105
0.25	2.18	111
0.2	2.27	116
0.15	2.35	120
0.1	2.43	124
0.05	2.49	127

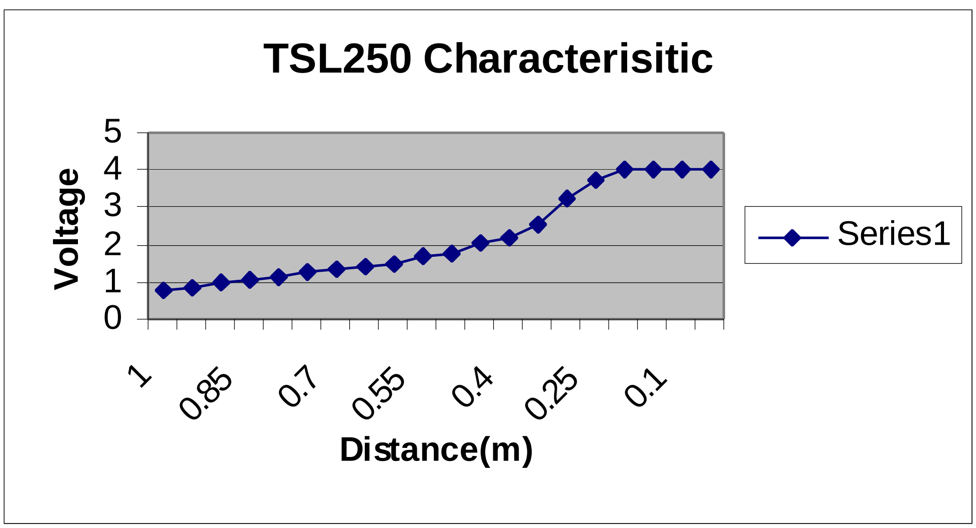


Lane-Sensor

My first test of the TSL250 involved the use of a single candle in a dark room. I then varied the distance of the light source from the sensor linearly and noted the output voltage of the TSL250.

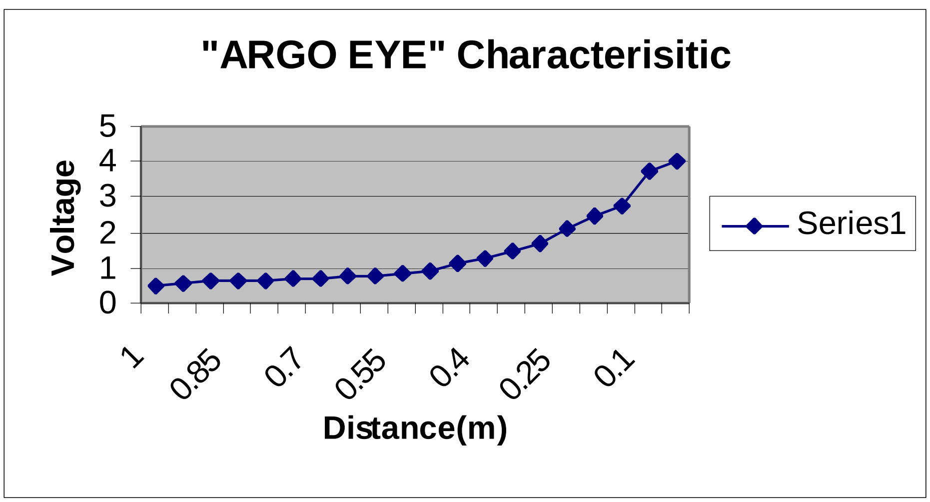
These results are summarized in the following table and graph.

Distance(m)	Voltage
1	0.78
0.95	0.83
0.9	1
0.85	1.09
0.8	1.15
0.75	1.25
0.7	1.31
0.65	1.4
0.6	1.5
0.55	1.68
0.5	1.76
0.45	2.05
0.4	2.2
0.35	2.55
0.3	3.25
0.25	3.75
0.2	4



The first tests of the ARGO "Eye" yield the following results, as summarized in the following table and plot.

Distance(m)	Voltage
1	0.51
0.95	0.55
0.9	0.6
0.85	0.6
0.8	0.63
0.75	0.7
0.7	0.71
0.65	0.75
0.6	0.8
0.55	0.85
0.5	0.91
0.45	1.1
0.4	1.25
0.35	1.5
0.3	1.7
0.25	2.1
0.2	2.45
0.15	2.72
0.1	3.75
0.05	4



In this initial data sampling, the lens was tilted forward so that it focused approximately .5 meters in front of the "eye." Thus, the response of the eye way negligible until the flame had reached that point. After this realization, I conducted another test to corroborate this claim. This time I focused the "eye" on the flame at each distance, and noted the output voltage.

Distance(m)	Voltage
1	1.55
0.95	2.5
0.9	3.85
0.85	4

The "eye" could be made to focus on a light source approximately one meter in front of the car.

Now, that I had tested the light response of the lane-sensor, I needed to determine if the sensor would be able to detect changes in the color, evidencing a lane marker. The following test was made to determine if the "eye" could distinguish color under ambient light.

Color	Distance(m)	Voltage
Blue	0.2	1.53
Blue	0.4	1.6
Blue	0.5	1.7
Blue	0.8	1.65

Color	Distance(m)	Voltage
White	0.2	3.1
White	0.4	2.8
White	0.5	2.75
White	0.8	2.72

Conclusions

Through the construction of ARGO I have learned many things, some of which I would like to forget—the burns from the transistors. Someone once said, ”Patience is a virtue.” After having completed this project—missing a little shy of my initial goals, I have to agree. Through the many hours spent debugging the motor driver, to the many hours looking for that missing pound sign in my assembly code, I have learned its significance. In addition, I have learned the fundamental difference between theory and reality: two worlds juxtaposed, yet not. It takes a lot of work to bring the two together, but once you have done that, everything seems a lot more clear.

Philosophy leads to science, and so too does this paper. When I began the semester I intended to make a robot that would follow a yellow lanes **and** avoid obstacles at the same time. I have a robot that follows black lanes, but cannot, due to the lack of another light/voltage sensor, make it avoid obstacles at the same time. I intended to have the motor control circuit finished by the end of the third week. I finished that circuit at the end of the eighth week, without an analog current meter.

These limitations only served to make me try harder to attain my goals. These goals I have met, and future goals have become even loftier. I intend to further my work on this robot, implementing a camera for its vision and a compass assist in destination finding. Building on a larger scale will enable me to use metal detection—my original, although impractical, lane following design—to follow lanes. This will eliminate the highly problematic ambient lighting differences, differences that only worsen when the robot is brought outside, where it will eventually be able to function. I have numerous ideas about that which I wish to do with this robot. All that I need right now, however, is a break.

Appendix

Code Listing

```
*Author: John Ferrara
*Title:  Demo1.asm

ADCTL EQU     $1030
ADR1    EQU     $1031
ADR2    EQU    $1032
ADR3    EQU    $1033
ADR4    EQU    $1034
TCNT    EQU    $100E  ; TCNT High byte
TFLG2   EQU    $1025  ; Contains RTIF flag
TMSK2   EQU    $1024    ; RTII enable flag
PACTL   EQU    $1026  ; RTI Timer control
BAUD    EQU    $102B  ; BAUD rate control register to set the BAUD rate
SCCR1   EQU    $102C  ; Serial Communication Control Register-1
SCCR2   EQU    $102D  ; Serial Communication Control Register-2
SCSR    EQU    $102E  ; Serial Communication Status Register
SCDR    EQU    $102F  ; Serial Communication Data Register
TOC2    EQU   $1018
TOC3    EQU    $101A
PORTA   EQU    $1000
PORTD   EQU    $1008
DDRD    EQU    $1000
TCTL1      EQU    $1020
TMSK1 EQU     $1022
TFLG1  EQU     $1023
OPTION            EQU     $1039
BASE    EQU     $1000
BIT76   EQU    %11000000
BIT54   EQU     %00110000
BIT7    EQU    %10000000
BIT6    EQU    %01000000
BIT5     EQU     %00100000
BIT4     EQU     %00010000
BIT3    EQU    %00001000
BIT1    EQU   %00000010
BIT2    EQU    %00000100
INV5     EQU     %11011111

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 Charracter
*
*************************************************************************
* Initialize Interrupt Jump Vectors
*************************************************************************
*
*RTI-INTERRUPT: USED FOR MULTITASKING

       ORG     $00EB
       JMP     RTI_ISR
```

```
*OC2-INTERRUPT: USED FOR SERVO CONTROL

            ORG     $00DC
            JMP     OC2_ISR


*OC3-INTERRUPT: USED FOR MOTOR SPEED CONTROL

       ORG     $00D9
       JMP     OC3_ISR

*IRQ-INTERRUPT: USED FOR THE "SOFT" RESET OF THE EVBU

       ORG     $00EE
       JMP     IRQ_ISR

*
*****************************************************************************
* Define Strings and Reserve Variable memory space for system use
*   such as CPT, DSPT, CurPID, etc.
*************************************************************************
       ORG $8000

*MULTITASKING
VARIABLES**************************************************************
CPT         RMB 2
            RMB 2
            RMB 2
            RMB 2
            RMB 2
            RMB 2
            RMB 2
            RMB 2
             RMB 2
             RMB 2

DSPT
              FDB $80FF
              FDB $81FF
              FDB $82FF
             FDB $83FF
             FDB $84FF
             FDB $85FF
             FDB $86FF
            FDB $87FF
            FDB $88FF
            FDB $89FF

MaxProc FCB $09
CurrPID RMB 1
temp    FCB $00
TEMP2  RMB 2
STACK RMB 2
```

```
*MOTOR-CONTROL
VARIABLES************************************************************

OC3Period      FDB     10000   ;20ms period for the servo control pulses
OC3HIGH         RMB     2
OC3LOW          RMB     2

OC3_CHANGE   RMB     2
SPEED_BOOST     FDB     6000
DIRECTION       RMB     1
DIR_PREV        RMB     1

*SERVO AND OBSTACLE AVOIDANCE
VARIABLES************************************************

OC2Period          FDB       40000    ;20ms period for the servo control pulses
OC2HIGH                      RMB      2
OC2LOW                       RMB      2
OC2_CHANGE   RMB     2

IR_TABLE
RF_IR              RMB     1
LF_IR              RMB     1
MF_IR          RMB     1
TRF_IR             RMB     1
TLF_IR             RMB     1

CURR_MAX       RMB     1
MAX_IR                       RMB      1

RF_INIT_IR       RMB     1
LF_INIT_IR     RMB     1
MF_INIT_IR     RMB     1
TRF_INIT_IR    RMB     1
TLF_INIT_IR    RMB     1

RF_OLD         RMB     1
LF_OLD         RMB     1

*FUZZY LOGIC TABLES************************************************

T_INDEX        RMB     1
TURN_FOUND     RMB     2

DIFF_IR        FCB     4
          FCB     7
          FCB     13
          FCB     20

TURN_SMOOTH     FDB     200
          FDB     300
          FDB     500
          FDB     600

DIFF_SPEED     FCB       85
               FCB       95
```

```
                FCB     105
                FCB     115

SMOOTH_SPEED
                FDB     4500
                FDB     3500
                FDB     2700
                FDB     2000

*cOMMAND ARBITRATOR VARIABLES***********************************************

EXEC_HORN    RMB    1
EXEC_AVOID   RMB    1
EXEC_LANE    RMB    1

*LANE FOLLOWING SENSORS***********************************************

RIGHT_LIGHT     RMB     1
LEFT_LIGHT     RMB    1
LEFT_INIT_L    RMB    1
RIGHT_INIT_L    RMB    1

RIGHT         RMB    1
LEFT         RMB    1
RL_OLD                  RMB     1
LL_OLD                  RMB     1

*RESET VARIABLES:"SOFT" REBOOT***********************************************
STARTUP        RMB    1

*************************************************************************
*************************************************************************

     ORG     $9000
SOFT_RESET
     LDS     #$41  ;Initialize Stack Pointer
         LDAA   #0
         STAA   STARTUP

     JSR    INIT_MULTITASK
     TYS
     JSR    InitRTI

     JSR    INIT_DIRECTION
     JSR    INIT_OC

     CLI

         BRA     WAIT_START

STOP_RESET
         LDS      #$41

WAIT_START
         LDAA   STARTUP
         CMPA   #1
```

```
        BNE     WAIT_START

Main    LDS     #$0041  ;Initialize Stack Pointer
        SEI

        JSR     INIT_ANALOG
        JSR     INIT_PORTS

        JSR     INIT_IR
        JSR     INIT_SPEED
        JSR     INIT_DIRECTION
        JSR     INIT_LIGHTS
        JSR     INIT_MULTITASK

        TYS     ;initialize stack pointer

        LDX     #CONTROL_RESTART
        JSR     Spawn

        LDX     #READ_SENSORS
        JSR     Spawn

        LDX     #ARBITRATE_MOTOR
        JSR     Spawn

        LDX     #CONTROL_MOTOR
        JSR     Spawn

        LDX     #CONTROL_SERVO
        JSR     Spawn

        LDX     #AVOID_OBSTACLES
        JSR     Spawn

        LDX     #HEINOUS_NOISE
        JSR     Spawn

        LDX     #FOLLOW_LANE
        JSR     Spawn

        CLI

Command_Arbit
        LDAA    #0
        STAA    EXEC_AVOID
        STAA    EXEC_HORN

        LDAA    #1
        STAA    EXEC_LANE
        LDD     #2000
        STD     OC3_CHANGE

        JSR     Delay

R_1
        LDAA    RIGHT_LIGHT
```

```
        ANDA   #$80
        BNE    R_1

        LDAA   LEFT_LIGHT
        ANDA   #$80
        BNE    R_1

        LDAA   #1
        STAA   EXEC_AVOID
        STAA   EXEC_HORN
        LDAA   #0
        STAA   EXEC_LANE

        LDD    #2200
        STD    OC3_CHANGE

        JSR    Delay

R_2
        LDAA   RIGHT_LIGHT
        ANDA   #$80
        BNE    R_2

        LDAA   LEFT_LIGHT
        ANDA   #$80
        BNE    R_2

        LDAA   #0
        STAA   EXEC_AVOID
        STAA   EXEC_HORN

        LDAA   #1
        STAA   EXEC_LANE

E_DEMO
        LDAA   RIGHT_LIGHT
        ANDA   #$80
        BNE    E_DEMO

        LDAA   LEFT_LIGHT
        ANDA   #$80
        BNE    E_DEMO

        SWI

*******************************************************************
*******************************************************************
*MULTITASKING PROCESSES***************************************
*******************************************************************

*******************************************************************
*SUBROUTINE: CONTROL_RESTART
*PURPOSE: CONTROLS THE SOFTWARE RESTART OF THE PROCESSOR
********************************************************************
CONTROL_RESTART
           LDAA   STARTUP
```

```
        CMPA  #1
        BEQ   END_RESTART

    JSR   INIT_MULTITASK
    JMP          STOP_RESET

END_RESTART
        BRA   CONTROL_RESTART

*******************************************************************
*SUBROUTINE: READ_SENSORS
*PURPOSE: READS THE ANALOG PORTS CONTAINING IR_SENSORS
********************************************************************

READ_SENSORS
        LDAA   #$30    ;FIRST, READ THE FIRST FOUR ANALOG PORTS(ALL IR)
        STAA   ADCTL

WAIT_SENS
        LDAA   ADCTL ;WAIT FOR 4-CONVERSIONS TO COMPLETE
        ANDA   #BIT7
        BEQ    WAIT_SENS

    LDAA   #$30   ;CLEAR THE CONVERSIONS FLAG
        STAA   ADCTL

    LDAA   RF_IR
    STAA   RF_OLD

    LDAA   ADR1   ;GET THE SENSOR READINGS AND PUT INTO GLOBAL
        STAA   RF_IR   ;VARIABLES

    LDAA   LF_IR
    STAA   LF_OLD

    LDAA   ADR2
        STAA   LF_IR

    LDAA   ADR3
    STAA   MF_IR

        LDAA   ADR4
        STAA   TRF_IR

        LDAA   #$34
        STAA   ADCTL

*NOW, GET THE NEXT FOUR ANALOG PORT READINGS

WAIT_SENS2
        LDAA   ADCTL ;WAIT FOR 4-CONVERSIONS TO COMPLETE
        ANDA   #BIT7
        BEQ    WAIT_SENS2

    LDAA   #$34   ;CLEAR THE CONVERSIONS FLAG
        STAA   ADCTL
```

```
    LDAA   ADR1   ;GET THE SENSOR READINGS AND PUT INTO GLOBAL
        STAA   TLF_IR ;VARIABLES

    LDAA   ADR2
        LDAB   RIGHT_LIGHT
        STAB   RL_OLD
        STAA   RIGHT_LIGHT

    LDAA   ADR3
        LDAB   LEFT_LIGHT
        STAB   LL_OLD
    STAA   LEFT_LIGHT

        JMP      READ_SENSORS


*********************************************************************
*SUBROUTINE: FOLLOW_LANE
*PURPOSE: FOLLOWS THE YELLOW LANE
**********************************************************************
FOLLOW_LANE
    LDAA   EXEC_LANE
    CMPA   #1
    BNE    FOLLOW_LANE

    LDAA   RIGHT_LIGHT
    SUBA   LEFT_LIGHT
    CMPA   #7
    BGT    LEFT_LANE

    LDAA   LEFT_LIGHT
    SUBA   RIGHT_LIGHT
    CMPA   #7
    BGT    RIGHT_LANE

    BRA    BETWEEN_LANES

RIGHT_LANE
    LDD    #3600
    STD    OC2_CHANGE
    JMP    FOLLOW_LANE

LEFT_LANE
    LDD    #2400
    STD    OC2_CHANGE
    JMP    FOLLOW_LANE

BETWEEN_LANES
    LDD    #3000
    STD    OC2_CHANGE
    JMP    FOLLOW_LANE

*********************************************************************
*SUBROUTINE: FOLLOW_HUMAN
*PURPOSE: FOLLOWS A HUMAN WITH AN IR EMITTER
```

```
*********************************************************************
FOLLOW_HUMAN
      LDAA    TRF_IR
      CMPA    TLF_IR
      BGT     HUMAN_RIGHT

HUMAN_LEFT
      LDD     #3400
      STD     OC2_CHANGE
      BRA     FOLLOW_HUMAN

HUMAN_RIGHT
      LDD     #2600
      STD     OC2_CHANGE
      BRA     FOLLOW_HUMAN

********************************************************************
*SUBROUTINE: FIND_LIGHT
*PURPOSE: MOVES INTO THE LIGHT, CARROL-ANNE
*********************************************************************
FIND_LIGHT
      LDAA    RIGHT_LIGHT
      LSRA
      LDAB    LEFT_LIGHT
      LSRB
      SBA
      BGT     LIGHT_RIGHT

LIGHT_LEFT
      LDD     #3400
      STD     OC2_CHANGE
      BRA     END_LIGHT

LIGHT_RIGHT
      LDD     #2600
      STD     OC2_CHANGE

END_LIGHT
      JMP     FIND_LIGHT

********************************************************************
*SUBROUTINE: AVOID_OBSTACLES
*PURPOSE: AVOIDS ONCOMING CARS
*********************************************************************
AVOID_OBSTACLES

          LDAA    EXEC_AVOID
      CMPA    #1
      BNE     AVOID_OBSTACLES

     LDAA    PORTD
      ANDA    #BIT3
      BNE     GO_REVERSE_LI

          LDAA   RF_IR
      CMPA   #125
```

```
        BGT     GO_REVERSE

        LDAA    LF_IR
        CMPA    #125
        BGT     GO_REVERSE

        LDAA    MF_IR
        CMPA    #125
        BGT     GO_REVERSE

        LDAA    #0
        STAA    DIRECTION

        LDAA    RF_IR
        CMPA    RF_OLD
        BGT     NEED_TURN

        LDAA    LF_IR
        CMPA    LF_OLD
        BGT     NEED_TURN

        LDAA    RF_IR
        CMPA    RF_INIT_IR + 5
        BGT     NEED_TURN

        LDAA    LF_IR
        CMPA    LF_INIT_IR + 5
        BGT     NEED_TURN


        BRA     STRAIGHT

NEED_TURN
        LDD     #2500
        STD     OC3_CHANGE

        LDAA    RF_IR
        CMPA    LF_IR
        BGT     TURN_LEFT

        LDAA    LF_IR
        CMPA    RF_IR + 3
        BGT     TURN_RIGHT

        JMP     AVOID_OBSTACLES

GO_REVERSE
        LDAA    #1
        STAA    DIRECTION

        LDAA    RF_IR
        CMPA    LF_IR
        BGT     REV_LEFT

REV_RIGHT
        LDD     #3300
```

```
      STD     OC3_CHANGE
      BRA     END_REVERSE

REV_LEFT
      LDD     #2700
      STD     OC2_CHANGE

END_REVERSE
      JSR     HONK_DELAY
      JMP     AVOID_OBSTACLES

STRAIGHT
      LDD     #3500
      STD     OC3_CHANGE

          LDD      #3000
          STD      OC2_CHANGE
      JMP     END_AVOID

TURN_LEFT
      LDAA    RF_IR
      SUBA    LF_IR
      JSR     SEARCH_TABLE_IR

      LDD     #3000
      SUBD    TURN_FOUND

      STD     OC2_CHANGE

      BRA     END_AVOID

TURN_RIGHT
      LDAA    LF_IR
      SUBA    RF_IR
      JSR     SEARCH_TABLE_IR

      LDD     #3000
      ADDD    TURN_FOUND

      STD     OC2_CHANGE
      BRA     END_AVOID

END_AVOID
          JMP      AVOID_OBSTACLES

*********************************************************************
*SUBROUTINE: HEINOUS_NOISE
*PURPOSE: HONKS THE CAR HORN
**********************************************************************
HEINOUS_NOISE

          LDAA    EXEC_HORN
      CMPA    #1
      BNE     HEINOUS_NOISE

      LDAA    RF_IR
```

```
      CMPA   #100
      BGT    HONK

      LDAA   LF_IR
      CMPA   #100
      BGT    HONK

      BRA    HEINOUS_NOISE

HONK
      LDAA   #$F1
      STAA   $7000

      JSR    HONK_DELAY

      LDAA   #$F0
      STAA   $7000

      JSR    HONK_DELAY

      LDAA   #$F1
      STAA   $7000

      JSR    HONK_DELAY

      LDAA   #$F0
      STAA   $7000

      JMP    HEINOUS_NOISE

*********************************************************************
*SUBROUTINE: CONTROL_MOTOR
*PURPOSE: aDJUSTS THE SPEED OF THE MOTOR
*********************************************************************

CONTROL_MOTOR
            LDD     OC3_CHANGE
            STD     OC3HIGH

            LDD     OC3Period
            SUBD    OC3HIGH
            STD     OC3LOW

            LDAA    DIRECTION
            CMPA    #0
            BEQ     FORWARD

REVERSE
            LDAA    PORTD
            ORAA    #BIT2
            STAA    PORTD
      BRA    END_MOTORCONTROL

FORWARD
            LDAA    PORTD
            ANDA    #$FB
```

```
        STAA    PORTD

END_MOTORCONTROL
        JMP     CONTROL_MOTOR

*******************************************************************
*SUBROUTINE: ARBITRATE_MOTOR
*PURPOSE: CONTROLS THE TRANSITION SPEED OF THE MOTOR
********************************************************************
ARBITRATE_MOTOR
        LDAA    EXEC_LANE
        CMPA    #1
        BNE     CHANGE_SPEED

        LDD     #2000
        STD     OC3_CHANGE
        BRA     ARBITRATE_MOTOR

CHANGE_SPEED
        LDAA    DIRECTION
        CMPA    DIR_PREV
        BEQ     SAME_DIR

        LDD     SPEED_BOOST
        STD     OC3_CHANGE
        JSR     HONK_DELAY

SAME_DIR
    LDAA    DIRECTION
        STAA    DIR_PREV

        LDAA    #3
        LDY     #MAX_IR
        LDX     #IR_TABLE
        JSR     FIND_MAX

        LDAA    MAX_IR
        LDX     #DIFF_SPEED
        LDY     #SMOOTH_SPEED
    JSR     SEARCH_TABLE_MOT

        CMPA    #1
        BNE     SLOWEST_SPEED

        STX     OC3_CHANGE
        BRA     END_ARBMOT

SLOWEST_SPEED
        LDX     #1900
        STX     OC3_CHANGE

END_ARBMOT
        JMP     ARBITRATE_MOTOR


*******************************************************************
```

```
*SUBROUTINE: CONTROL_SERVO
*PURPOSE:ADJUSTS THE POSITION OF THE SERVO
*********************************************************************

CONTROL_SERVO
        LDD     OC2_CHANGE
        STD     OC2HIGH

        LDD     OC2Period
        SUBD    OC2HIGH
        STD     OC2LOW

        JMP     CONTROL_SERVO

******************************INITIALIZATION CODES********************************8

********************************************************************
*SUBROUTINE: INIT_MULTITASK
*PURPOSE: INITIALIZES THE CPT AND STACK POINTERS
*********************************************************************

INIT_MULTITASK
     LDAB   #$00
     LDX    #CPT
     LDAA   #$0A

Zero    LDY     #$0     ;this code zeros-out the CPT(table)
        STY     0,X
        INX
     INX
     INCB
        CBA
        BNE     Zero

        LDAA    #$00    ;initialize the starting PID
        STAA    CurrPID

        LDX     #DSPT
        LDY     0,X

        LDX    #CPT     ;setup the initial stack pointer for the first
     STY    0,X  ;...process
     INY

        LDAA    #BIT6   ;clear the RTI-flag so the next process is not
     STAA   TFLG2       ;immediately interruted

        RTS

********************************************************************
*SUBROUTINE: SET_ANALOG
*PURPOSE: CONFIGURES THE A/D SYSTEM FOR LATER USE
*********************************************************************

INIT_ANALOG
        LDAA    #$80    ;POWER-UP A/D-SYSTEM
```

```
        STAA   OPTION

        LDAA   #40
WAIT_AN
     DECA          ;WAIT FOR CHARGE PUMP TO STABILIZE
     BNE    WAIT_AN

        RTS

*******************************************************************
*SUBROUTINE: INIT_IR
*PURPOSE: SETS-UP INITIAL IR CONDITIONS (SELF-CALIBRATION)
*******************************************************************

INIT_IR
        LDAA    #$30
        STAA   ADCTL

WAIT_IR         LDAA   ADCTL ;WAIT FOR 4-CONVERSIONS TO COMPLETE
        ANDA  #BIT7
        BEQ     WAIT_IR

     LDAA    #$30    ;CLEAR THE CONVERSIONS FLAG
        STAA   ADCTL

     LDAA    ADR1    ;GET THE SENSOR READINGS AND PUT INTO GLOBAL
        STAA   RF_INIT_IR       ;VARIABLES

     LDAA    ADR2
        STAA   LF_INIT_IR

     LDAA    ADR3
     STAA    MF_INIT_IR

        LDAA   ADR4
        STAA   TRF_INIT_IR

        LDAA    #$34
        STAA   ADCTL

WAIT_IR2
        LDAA   ADCTL ;WAIT FOR 4-CONVERSIONS TO COMPLETE
        ANDA  #BIT7
        BEQ     WAIT_IR2

     LDAA    #$34    ;CLEAR THE CONVERSIONS FLAG
        STAA   ADCTL

     LDAA    ADR1    ;GET THE SENSOR READINGS AND PUT INTO GLOBAL
        STAA   TLF_INIT_IR     ;VARIABLES

        RTS


INIT_LIGHTS
        LDAA    #$34
```

```
        STAA    ADCTL

WAIT_LTS
        LDAA    ADCTL ;WAIT FOR 4-CONVERSIONS TO COMPLETE
        ANDA    #BIT7
    BEQ     WAIT_LTS

    LDAA    #$34    ;CLEAR THE CONVERSIONS FLAG
        STAA    ADCTL

    LDAA    ADR2    ;GET THE SENSOR READINGS AND PUT INTO GLOBAL
    STAA    RIGHT_INIT_L    ;VARIABLES

    LDAA    ADR3
    STAA    LEFT_INIT_L

        RTS

*******************************************************************
*SUBROUTINE: INIT_PORTS
*PURPOSE: INITIALIZES ALL PORTS TO BE EITHER INPUTS OR OUTPUTS
*******************************************************************

INIT_PORTS
    LDAA    #$F7
    STAA    DDRD

    LDAA    #$F0
    STAA    $7000

    LDAA    #0
    STAA    PORTA

        RTS


*******************************************************************
*SUBROUTINE: INIT_SPEED
*PURPOSE: STARTS THE CAR AT AN INITAL SLOW SPEED
*******************************************************************

INIT_SPEED

    LDD     #3200
    STD     OC3HIGH
    STD     OC3_CHANGE

    LDD     OC3Period
    SUBD    OC3HIGH
    STD     OC3LOW

        RTS

*******************************************************************
*SUBROUTINE: INIT_DIRECTION
*PURPOSE: STARTS THE CAR OUT IN FORWARD GOING STRAIGHT
```

```
*********************************************************************

INIT_DIRECTION
        LDAA   PORTD
        ANDA   #$FB
        STAA   PORTD

        LDAA   #0
        STAA   DIRECTION
    LDAA   #1
    STAA   DIR_PREV

    LDD    #3000   ;START WITH SERVOS STRAIGHT
        STD      OC2HIGH
    STD    OC2_CHANGE

        LDD      OC2Period
        SUBD   OC2HIGH
        STD      OC2LOW

        RTS

********************************************************************
* Subroutine:  InitRTI
* Function:    This routine enables RTIs and sets the RTI rate to
*              32.77ms.
* Input:       None
* Output:      Initializes RTI
********************************************************************
InitRTI LDAA   #$88    ;set the interrupt rate to 32.77ms
    STAA   PACTL
    LDAA         #$40
        STAA   TMSK2
        RTS                  ;return to the main

********************************************************************
*SUBROUTINE: INIT_OC
*PURPOSE: INITIALIZES ALL OUTPUT COMPARES
*********************************************************************

INIT_OC

    LDAA   #%10100000   ;CLEAR OC2 AND 0C3 LINES TO ZERO
        STAA   TCTL1

    LDAA   #%01100000   ;ENABLE OC2 AND OC3 INTERRUPT
        STAA   TMSK1

        RTS

*********************************INTERRUPT SERVICE
ROUTINES***************************8

********************************************************************
*SUBROUTINE: OC2_ISR
*PURPOSE: CONTROLS THE OUTPUT COMPARE 2
```

```
**********************************************************************

OC2_ISR
	LDAA	TFLG1	;CHECK FOR LEGAL INTERRUPT
	ANDA	#BIT6
	BEQ	END_OC2

    LDAA	#BIT6	;CLEAR THE INTERRUPT
    STAA	TFLG1

	LDAA	TCTL1	;CHECK IF LAST PULSE WAS HIGH OR LOW
	ANDA	#BIT6
	BEQ	LASTHIGH_OC2

    LDAA	TCTL1	;SET NEXT PULSE TO BE LOW
    EORA	#$40
    STAA	TCTL1

    LDD	TOC2 ;SET OC2 HIGH TIME
    ADDD	OC2HIGH
	STD	TOC2

    BRA	END_OC2

LASTHIGH_OC2
    LDAA	TCTL1	;SET THE NEXT PULSE TO BE HIGH
    EORA	#$40
	STAA	TCTL1

    LDD	TOC2
    ADDD	OC2LOW
	STD	TOC2

END_OC2
	RTI		;RETURN FROM INTERRUPT

*********************************************************************
*SUBROUTINE: OC3_ISR
*PURPOSE:CONTROLS OUTPUT COMPARE 3
**********************************************************************

OC3_ISR
	LDAA	TFLG1	;CHECK FOR LEGAL INTERRUPT
    ANDA	#BIT5
    BEQ	END_OC3

    LDAA	#BIT5	;CLEAR THE INTERRUPT
    STAA	TFLG1

	LDAA	TCTL1	;CHECK IF LAST PULSE WAS HIGH OR LOW
    ANDA	#BIT4
	BEQ	LASTHIGH_OC3

    LDAA	TCTL1	;SET NEXT PULSE TO BE LOW
    EORA	#$10
    STAA	TCTL1
```

```
      LDD     TOC3 ;SET OC2 HIGH TIME
      ADDD    OC3HIGH
      STD     TOC3

      BRA     END_OC3

LASTHIGH_OC3
      LDAA    TCTL1    ;SET THE NEXT PULSE TO BE HIGH
      EORA    #$10
      STAA    TCTL1

      LDD     TOC3
      ADDD    OC3LOW
      STD     TOC3

END_OC3
          RTI          ;RETURN FROM INTERRUPT
**********************************************************************
*SUBROUTINE: IRQ_ISR
*PURPOSE:USED FOR THE REBOOT PROCESS
***********************************************************************

IRQ_ISR
          LDAA   STARTUP
          CMPA   #0
          BEQ    GO_SYSTEMS

OFF_SYSTEMS
          LDAA   #0
          STAA   STARTUP

      BRA     END_IRQ

GO_SYSTEMS
          LDAA   #1
          STAA   STARTUP

END_IRQ
          RTI



*
**********************************************************************
* Interrupt Service Routine (ISR): RTI_ISR
* Function:  This ISR services the Real-Time Interrupts.
* This ISR should do the followings:
*  - Clear RTI flag.
*  - Update current SP in CPT[Current PID]
*  - Find next PID
*  - Update CurPID
*  - Load New SP from CPT[Next PID]
**********************************************************************
RTI_ISR
          LDX    #BASE
```

```
      LDAA    TFLG2
      ANDA    #BIT6
      BEQ     END_ISR      ;check for valid interrupt

      LDX           #CPT   ;store the current stack pointer to the appropriate
      LDAA    CurrPID      ;entry in the CPT, as dictated by CurrPID
      ASLA                 ;to get the offset into the table must mult.
      STAA    temp         ;CurrPID by 2
      LDAB    temp
      ABX
      TSY
          DEY
      STY     0,X      ;save the current stack pointer of the current process

          LDAA    CurrPID
      INCA            ;...to the appropriate slot in the CPT
*                  ;check to see if the process is the last one
FndNxt  CMPA    #$0A     ;is so the next process to be run is process[0]
          BEQ     REFRESH

      LDX     #CPT         ;determine the next nonzero entry in the CPT
      STAA    temp         ;...this is the stack pointer we need to use
      ASL     temp;for the next process
      LDAB    temp
      ABX
      INCA
      LDY     0,X      ;I must skip zeroed entried here, since my KILL
          BEQ     FndNxt   ;does not account for them
          BRA     SET

REFRESH LDX     #CPT
      LDY     0,X      ;restart the process poll
      LDAA    #$0
      STAA    CurrPID
          INY
          TYS

      BRA     RT_ISR

SET       INY
          TYS
      DECA            ;transfer the correct processes SP[Proc.] to the SP
      STAA    CurrPID

RT_ISR  LDAA    #BIT6  ;clear the RTI-flag so the next process is not
      STAA    TFLG2        ;immediately interruted

END_ISR RTI


******************************************************************
******************************************************************
*SUBROUTINES***************************************
******************************************************************
```

```
************************************************************************
*SUBROUTINE: SEARCH_TABLE
*PURPOSE: FINDS THE APPROPRIATE SERVO DIRECTION FOR HIGH-SPEED
*OBSTACLE AVOIDANCE
*INPUT: THE TABLE TO SEARCH IN THE X-REGISTER
*          THE TABLE WITH THE CORRESPONDING ENTRY SEARCHED FOR
*          IN ACCUMULATOR A IS THE VALUE TO MATCH IN THE TABLES
*************************************************************************
SEARCH_TABLE_IR
      LDAB   #0
      STAB   T_INDEX
      LDX    #DIFF_IR

ST_LOOP
      LDAB   0,X
      CBA
      BLT    FOUND_ENTRY

      INX
      LDAB    T_INDEX
      INCB
      STAB   T_INDEX
      CMPB   #4
      BLT    ST_LOOP

      LDD    #700
      STD    TURN_FOUND
      BRA    END_SEARCH

FOUND_ENTRY
      LDAB   T_INDEX
      ASLB
      LDX    #TURN_SMOOTH
      ABX
      LDD    0,X

END_SEARCH
      STD    TURN_FOUND
      RTS

SEARCH_TABLE_MOT

*
************************************************************************
* Subroutine: Spawn
* Function:   generates a new process
* Input:      X: starting address of the process
* Output:     A: PID of the process just created, or
*             $FF if no slots are available
* Destroys:   Contents of A register
* Side effects: Creates the initial stack for the process.  This
*             stack must have the process PID in A, and %01000000
*             in CCR.
************************************************************************
Spawn
```

```
        PSHY              ;put these on the stack so they are not destroyed
        PSHB
        STX      TEMP2
        LDAA   #$00      ;intialize the CPT position=0
        LDY      #CPT

FndPlc  LDX     0,Y     ;find the next empty spot in the CPT
    INY            ;POINT TO NEXT ENTRY IN THE TABLE
    INY
    INCA

    CMPA    #11     ;if A = 9 then the table has been completely
    BEQ     FULL           ;scanned and there are no zero entries

    CPX     #$0
    BNE     FndPlc

    DECA

    STAA         temp      ;store the displacement into the table in temp
    ASL     temp    ;multiply temp by 2=>index into table(CPT)
    LDAB    temp
    LDX     #DSPT   ;get default stack table
    ABX
    LDY     0,X  ;get appropriate default stack pointer
    LDX     #CPT    ;put into the appropriate CPT slot
        ABX
    STY     0,X

    STAA    temp

    LDY     0,X       ;initialize the stack for the new process
    LDX           TEMP2
        DEY
        STX     0,Y
    DEY
        DEY
        LDX     #$0
    STX     0,Y
    DEY
    DEY
    LDX     #$0
    STX     0,Y
    DEY
    LDAA    temp
    STAA    0,Y
    DEY
    LDAB    #$0
    STAB    0,Y
    DEY
    LDAB    #$40
    STAB    0,Y
        DEY                         ;pulled out DEY here

        ASL     temp
    LDAB    temp
```

```
      LDX    #CPT
      ABX
      STY    0,X

          PULB              ;restore variables used for this subroutine
          PULY
      BRA    RT_RTS

FULL   PULB
      PULY
      LDAA   #$FF

RT_RTS  RTS          ; return from subroutine

*********************************************************************
* Subroutine: Kill
* Function:   removes a currently active process
* Input:      A: process ID to kill
* Output:     A: process ID just killed
* Destroys:   None
*********************************************************************
Kill   PSHB

      STAA   temp
      ASL    temp
      LDAB   temp

          PSHX              ;SAVE THESE REGISTERS SOP THEY ARE NOT CHANGED
          PSHY

          LDY     #$0       ;zero-out the corresponding entry of the table
          LDX     #CPT
      ABX
      STY    0,X

      PULY          ;restor the registers used during the subroutine
          PULX
      PULB

          RTS               ;return from subroutine


*
*********************************************************************
*
* Subroutine:  Delay
* Input:       None
* Output:      Provides a delay by simple looping
* Destroys:    None
* Note:        If you're looking to save memory, this function
*              may be rewritten or subsumed by Process since
*              Process is the only routine to call it.
*********************************************************************
*
Delay   PSHA          ;
      PSHB          ;
```

```
      PSHX          ; Save registers
      PSHY          ;
      LDX    #10    ; Load outer loop counter
Outer   LDY    #10000    ; Load inner loop counter
Inner
      DEY           ; Decrement inner counter
      BNE    Inner     ; Branch if >0 to inner loop
      DEX           ; Decrement outer counter
      BNE    Outer     ; Branch if >0 to outer loop
      PULY          ; Restore registers
      PULX          ;
      PULB          ;
      PULA          ;
      RTS           ; Return from subroutine


HONK_DELAY
      LDX    #20000
H_LOOP
      DEX
      CPX    #0
      BNE    H_LOOP

      RTS


*******************************************************************
*
* Subroutine:  FIND_MAX
* Input:       ARRAY INDEX:X-REGISTER; ARRAY SIZE: A ACCUMULATOR;Y:WHERE TO
STORE THE MAX
* Output:      FINDS MAXIMUM OF A GIVEN ARRAY; PASSED IN THE X-REGISTER
* Destroys:    None
*******************************************************************
*

FIND_MAX
          LDAB   0,X
          STAB   CURR_MAX
          DECA
MAX_LOOP
          LDAB   0,X
          CMPB   CURR_MAX
          BLT    SAME_MAX
          STAB   CURR_MAX
SAME_MAX
          INX
          DECA
          CMPA   #0
          BNE    MAX_LOOP

          LDAB   CURR_MAX
          STAB   0,Y

          RTS
```