University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machine Design Laboratory

Final Report :
MOJO the Robot Arm

by Preston Faiks
April 21, 1999

Instructor: Dr. Antonio Arroyo
TAs: Scott Jantz, Aamir Quaiyumi, Patrick O'Malley

Table of Contents:

Abstract

MOJO is a robot arm mounted on a mobile platform. He searches randomly for objects to pick up while at the same time avoiding obstacles. Obstacle Avoidance is achieved using IR sensors on the fingers and on the base. Upon detecting an object he stops and tries to determine if the object is the right size to pick up by using two IR sensors mounted on the tips of the fingers. After determining that it is the correct size he moves to pick up the object. Once the object is detected between the gripper, the fingers close until there is enough pressure to pick up the object. A force sensor is used to determine when the fingers have gripped the object tight enough. Once an object is acquired MOJO moves the object to the side and moves forward to try and find more objects. If the object is too big or cannot be picked up for some reason, MOJO exhibits a failure behavior by shaking his head.

Executive Summary

MOJO was designed to demonstrate object recognition and manipulation by a robotic arm. The goal was to get the robot to pick up objects with specific properties, including size, shape and color and sort them accordingly. In order to accomplish this the robot needed to be very maneuverable and have several sensors.

MOJO's robot arm has 5 degrees of freedom, base rotation, arm elevation, wrist rotation, wrist elevation, and a gripper. The mobile base has 2 motors mounted in the middle to allow for full speed and direction control. This gives MOJO the maneuverability to perform the tasks that it was intended to do.

MOJO has 5 IR sensors and a force sensing resistor. Two IR sensors for collision avoidance, two for aligning objects, and one for detecting objects between the fingers. The FSR (Force Sensing Resistor) allows the fingers to put the right amount of pressure on the objects being picked up. These allow for very simple object detection, but more sensors would be required to perform all of the tasks that I intended to implement.

Using these sensors and the maneuverability of the arm, MOJO has the following behaviors. Scanning for small objects and avoiding large ones is MOJO's basic behavior. When an object is detected MOJO starts a series of behaviors designed to acquire the object. If the object is not acquired he shakes his head to show that he could not obtain the object.

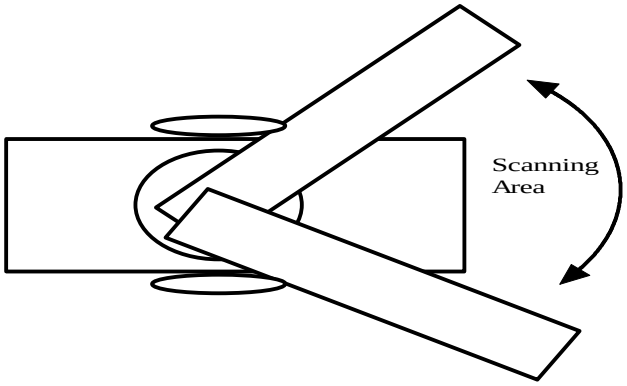
Introduction:

Getting a machine to recognize and manipulate objects has been the subject of many research projects in robotics. It is a challenge for a machine to inteligently dicifer between objects of different sizes and shapes, and even more challenging to pick them up and manipulate them. This project was an experiment to see how well a small robot could recognize and manipulate small objects.

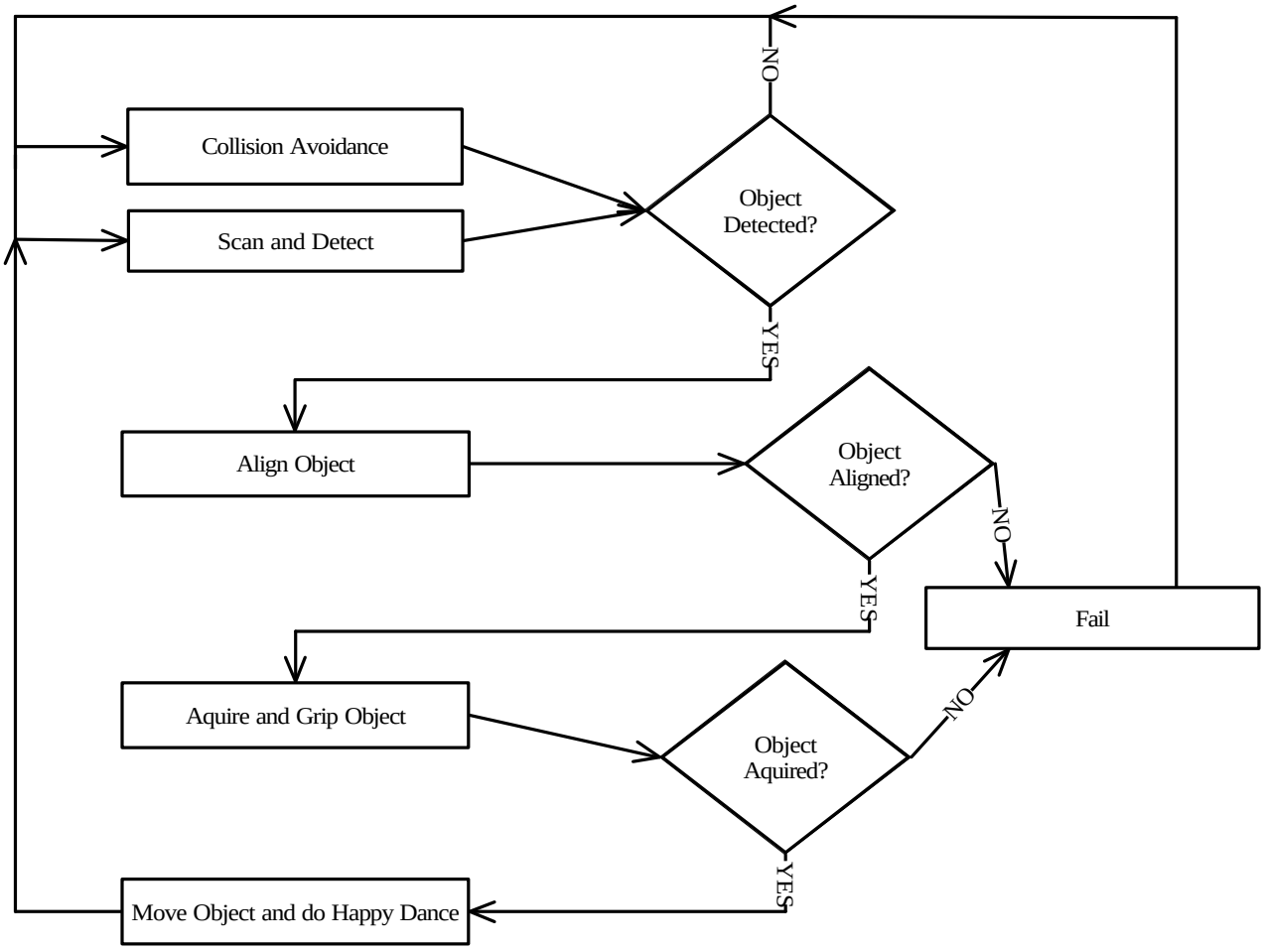
This paper will discuss building a mechanical arm, and the challenge of integrating sensors and software with it to create a robot capable of detectign, recognizing, and handling small objects.

Integrated System

The robotic arm, consisting of 5 servos is mounted onto a simple rectangular platform with a motor on each side. The main arm can swing back and forth across the entire front end of the robot and it is this motion that allows MOJO to scan for objects.



The mobile base just moves around and attempts to avoid obstacles. Since the arm protrudes in front of the base, it must scan at a height above the IR sensors mounted on the base. Although it is possible to scan the ground, collision avoidance would not be possible, because the IR sensors would detect the arm in front of them and not walls and other obstructions. The range of the IR on the base extends just beyond the reach of the arm, meaning it must turn sharply as soon as it detects an obstacle. The sensors on the fingers look for possible objects and serve as backup sensors for collision avoidance. The wrist motions allow the finger sensors to move in many different positions to determine the characteristics of an object. Once a possible object is detected, the mobile base stops and the object acquisition behaviors begin.

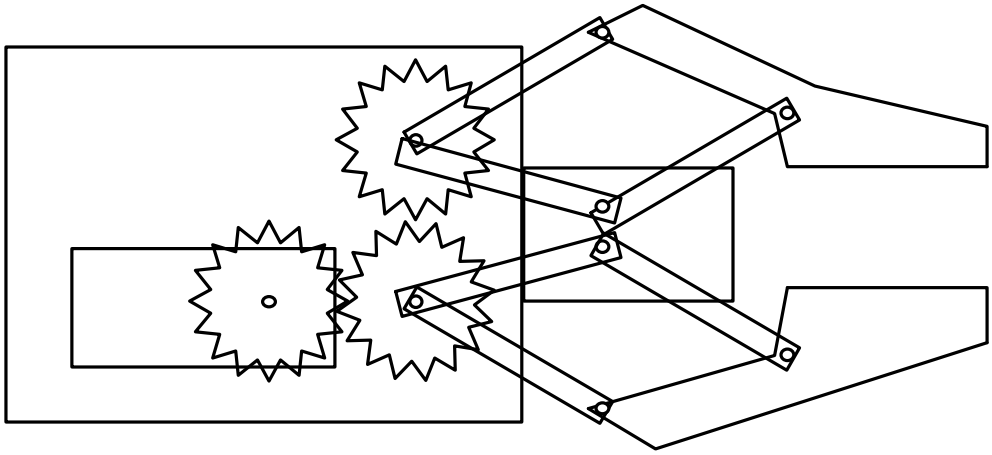


Right now only the simple characteristic of width is checked by the robot. I did not have time to program more complicated object recognition routines. A color sensor was also attempted but never mounted on the robot because it was unreliable.

Mobile Platform

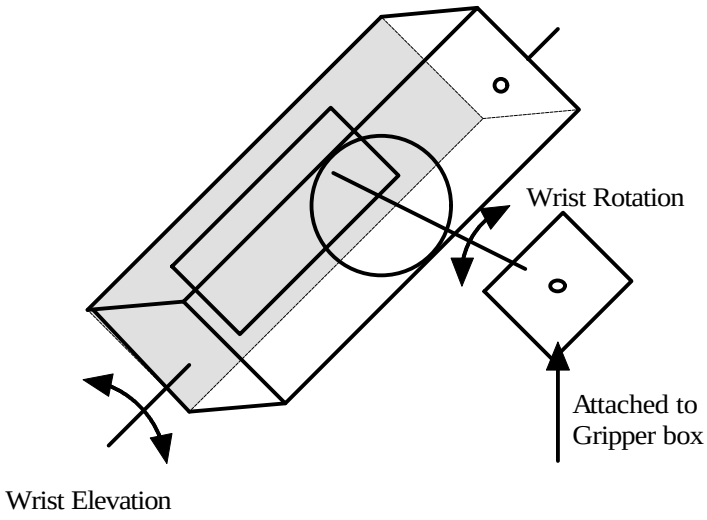
MOJO uses an original design that was drawn in AutoCad and cut out on the t-tech machine. The arm was designed in AutoCad so that simple modifications could be made easily to accommodate adding sensors at the end. As it turns out, the sensors were added too late to make platform modifications to accommodate them, the result was a lot of tape and hot glue holding sensors on. After being cut out on the t-tech, some parts needed the dremel tool's help to make them fit. These parts are indicated in Appendix C. Otherwise the platform was a good design.

The gripper box is probably the most mechanically complicated portion of the robot. It was designed so that the fingers would always be parallel no matter how far apart they were. This is beneficial because the finger sensors always point straight out, and for most objects, the surface area in contact will be maximized. Two gears being turned by one servo allowed for control of the gripper.

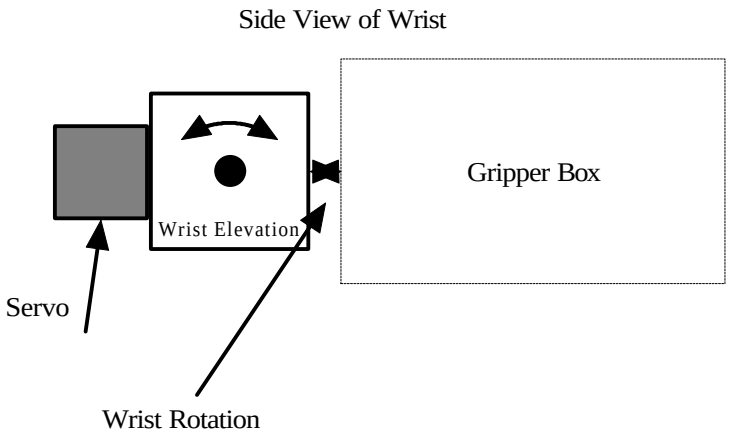


Gipper Design

The gripper box is attached to a rotation box that allows for the gripper box to be rotated a full 180°. The rotation box had attachments on the side so that it itself could be rotated.

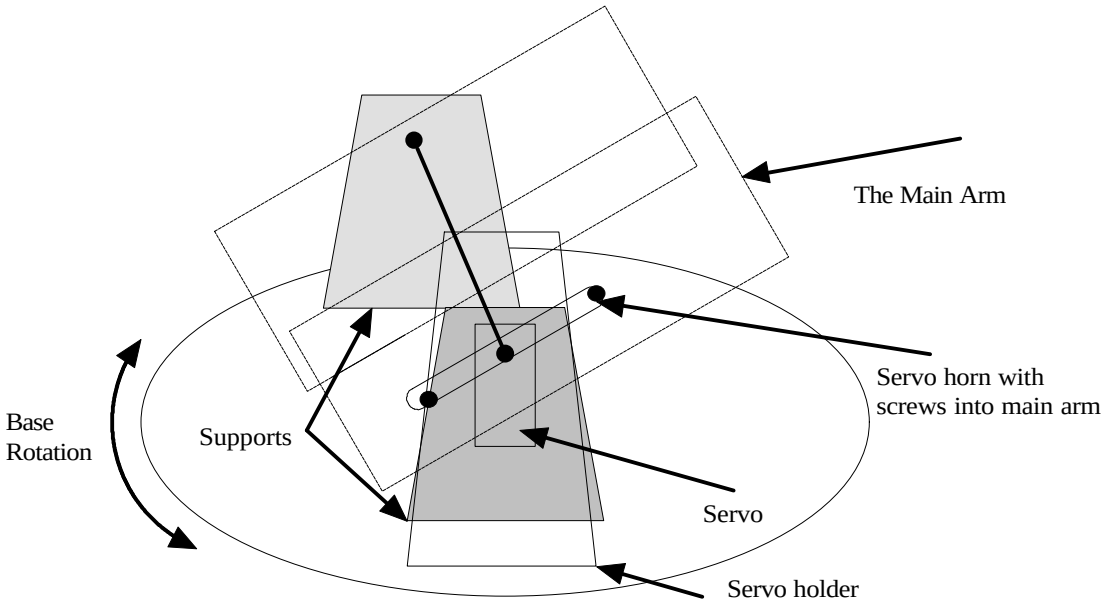


On the end of the arm was a servo attached to the rotation box that could raise and lower the entire hand from straight up to straight down. The axis of rotation was not exactly in the center of mass of the hand, but because the servo of the rotation box stuck out behind the axis, this problem was not too serious, and the hand could be rotated easily with a relatively small servo.



The base of the arm was attached to an axle that rotated freely. A large weight was added to counter balance the large torque caused by the three servos sitting about 10 -12 inches

from the base. Two large lead weights were used and placed so that they exactly balanced the arm when it was not picking anything up. Therefor the servo was responsible for lifting only the weight of the object. This servo was mounted to the side and attached with screws into the main shaft. This was beneficial to attaching the servo directly to the axle, because the weight of the arm was being supported by the supports and not the servo.



This whole structure was then mounted onto a Lazy Suzan platform. This is simply a rotating base with ball bearings that has a hole in the center. A servo mounted under this base sticks through the center and rotates the arm, while the Lazy Suzan supports the weight of the arm and keeps it stable by supporting it on the edges and not in the center.

The mobile platform was a simple rectangular platform with wheels on the center and castors at either end for balance. It has a place to mount all of the electronics and the batteries.

I learned a lot of things while constructing this robot. First of all, I had planed for many pieces to be glued together as the method attachment. This is fine if the design is perfect, but using brackets and screws can reduce problems when disassembling is required for one reason or another. I also learned how much the hardware store has to offer. I could have made the design a lot better if I had known what hardware stores had to offer at the beginning of the semester. You can get almost any piece required for any application. Lastly, small flaws in AutoCad design can be corrected by using the dremel tool or the drill press, holes that are too small can be made bigger, not vice versa so always underestimate if you do not know exact specifications.

Actuation:

MOJO uses 5 servos and 2 gear head motors for actuation. Of the servos, 3 are standard 42 oz-in and 2 are 60 oz-in. The main need for torque is in the two actuators that move the arm and wrist up and down. For these the 60 oz-in were used. The length of the are is about 12 inches, and since the weight of the arm is balanced, this servo only needs to pick up the weight of the object. (60 oz-in)/(12 inches) = 5oz maximum object weight. The wrist is only about 4 inches long, but it is not totally balanced so the 60 oz-in servo is needed to counter the inherent torque and lift the object. (60 oz-in)/(4 in) = 15oz maximum object plus hand weight. The 42 oz-in servos control wrist rotation --which is totally balanced and does not weight much-- and base rotation, which is only under stress when the servo moves, and never when the servo is stationary. The other 42 oz-in was used for the gripper, and was able to deliver enough pressure to hold an object of moderate weight. These calculations seemed fine, but when put to the test, some of the servos where under a lot of stress. The main arm elevator worked, but could be easily moved by bumps and jerks, making it shake when in use. Increasing this servo's power would have solved the problem. The other problem was that the base rested on the ball bearings of the Lazy Suzan piece, giving it very little friction. The base rotation servo could move the arm, but would oscillate when it reached it's desired position. Even when it was perfectly in place, a bump or jerk would make it go out of alignment and start to oscillate. My solution was to add friction with foam between the base and the arm, this provided a dampening to the oscillation and reduced the problem to almost nothing. Again, a more powerful servo would also have worked.

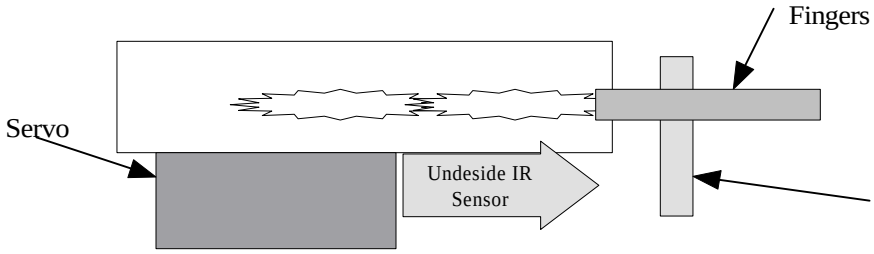
The two gear head motors were attached to 3” wheels on the base and used to move the entire robot. I used powerful motors with low rpm, giving me the torque I needed and sacrificing speed which was not needed. These motors worked well and I had no problems.

To control the servos, I wrote an interrupt service routine that altered an output port. The servos control lines were connected to the port and received a 30Hz pulse width modulated signal. The details of the code are in appendix C. It is important to note that this code can only be used for servos, because it is meant only to send short pulse widths. The duty cycles of all the pins added together cannot exceed 100%. This is fine for servos which normally operate at around 6%, meaning about 16 servos can be controlled using this one ISR. The motors uses the output compare functions of the 68HC11 and the ME11 board.

Sensors:

IR Sensors

MOJO used IR sensors for almost all of his behaviors. Two IR sensor/emitter pairs were positioned on the front sides of the base and angled slightly outward. The calumniation length of these emitters was very short --about ¼ of and inch -- because these sensors needed to see a wide range in front of the robot. Two more sensor/emitter pairs were placed at the ends of the fingers for object detection. The calumniation on these emitters were longer --about ¾ of and inch-- so that the sensors would only detect objects directly in from of them. One more sensor/emitter pair was used underneath the gripper to detect an object between the fingers. This one used about ¾ of an inch calumniation and black electrical tape was placed on the underside, to prevent reflection from the gripper box itself. Objects needed to extend about ¼ of an inch below the bottom of the fingers in order to be detected by this sensor.

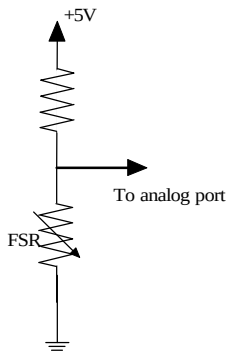


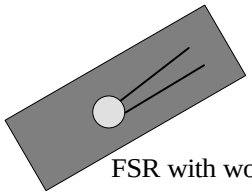
All emitters used the 40kHz signal from the ME11 board and 330Ω resistors. All IR sensors were Sharp GP1U58Y infrared sensors hacked to produce an analog signal.

Force Sensing Resistor (FSR)

This sensor was used on the inside of the fingers to tell if an object was being squeezed hard enough. This sensor gave an almost infinite resistance under no pressure, and

dropped sharply to under 100kΩ once slight pressure was applied. I used a simple voltage divider circuit with a 6.6 kΩ resistor --as shown in the figure—and attached it to an analog port.





FSR with wood on top

The active area of this sensor is a circle about ¼ of an inch in diameter. In order to increase this, I used a small piece of wood with a small spot of hot glue. The spot was placed over the active area and the wood was taped to the finger. Doing this effectively increased the active area because wherever the wood wad being pressed, it would transfer some of the force back to the FSR. This allowed for objects to be picked up anywhere inside of the pinchers.

Color Sensor:

This sensor was intended to give MOJO color recognition, but due to poor performance it was never actually used. The theory was to put color filters over Cds cells and pointing them at objects to determine color. I used some cheap filters from Edmund Scientific, which did not come with transmission vs. wavelength graphs. I feel that much better results could be obtained if I used better filters that can with transmission graphs. The results of this sensor are given in the experiments section.

Sensor Integration

MOJO's sensor's were highly integrated. The two collision sensors worked together to decide where to turn, and the finger sensors provided backup collision avoidance. The Finger sensors also worked together to align objects in front of the arm so that they could be picked up. The underneath sensor triggers the gripping behavior which is controlled by the FSR.

Sensor Experiments:

All of the following tests used display.c (Appendix B) to send sensor values on the serial port.

IR sensors

Testing different materials resulted in different results because materials all have vastly different infrared reflectance's. This table shows what the range and reflective dependence the IR sensors have. The values given are analog port readings.

	Material Used as Object		
Distance (in)	Green Folder	Black Folder	Hand
1	124	120	124
3	124	109	120
5	120	92	110
7	115	88	105
9	104	87	96
11	94	87	90
13	89	87	88
15	87	87	87

FSR

I tested the FSR by pressing on it with my finger and reading the analog port value. I used a 6.6kΩ resistor voltage divider circuit.

No Force	0	
small force	30	My finger resting on then sensor
medium force	120	My finger pushing on the sensor

large force 210 Pushing as hard as I can (within reason)

Color sensor

Although I never used it I did several experiments. As you can see the results were not promising, but the sensor data does show how certain colors could be singled out for detection. I just used different colored folders for these tests.

		Room Light				
	RED		GREEN		BLUE	
no object	104	100.00%	94	100.00%	155	100.00%
blue	49	47.12%	6	6.38%	25	16.13%
green	50	48.08%	12	12.77%	38	24.52%
red	59	56.73%	7	7.45%	35	22.58%
yellow	58	55.77%	15	15.96%	35	22.58%
purple	50	48.08%	5	5.32%	17	10.97%
		Room light plus 60 watt bulb nearby				
	RED		GREEN		BLUE	
no object	93	100.00%	75	100.00%	132	100.00%
blue	72	77.42%	38	50.67%	110	83.33%
green	71	76.34%	45	60.00%	91	68.94%
red	118	126.88%	41	54.67%	146	110.61%
yellow	173	186.02%	78	104.00%	150	113.64%
purple	76	81.72%	39	52.00%	120	90.91%
		Room Light plus 40 watt bulb nearby				
	RED		GREEN		BLUE	
no object	104	100.00%	91	100.00%	142	100.00%
blue	60	57.69%	20	21.98%	60	42.25%
green	60	57.69%	25	27.47%	60	42.25%
red	94	90.38%	20	21.98%	98	69.01%
yellow	97	93.27%	44	48.35%	108	76.06%
purple	64	61.54%	20	21.98%	80	56.34%

Behaviors

MOJO starts out in his basic behavior which is to move around, avoid obstacles and look for objects. As a backup, if the finger mounted sensors detect a large object or wall close by, MOJO will turn sharply until the obstruction is no longer there. Upon detection of an object from the finger mounted IR sensors a whole series of behaviors begin. First is the aligning behavior. MOJO will stop moving and the base will swing from side to side trying to get the object to line up right between the sensors. If MOJO is able to align the object the acquiring behavior begins. During this behavior MOJO moves toward the object until the under mounted IR detects that the object is between the fingers. Once there, the gripper slowly closes down until the pressure reaches a certain critical value, or the gripper is fully closed. If the object is in the grips of the arm, MOJO simply moves the object to the side, and does a happy dance showing off the range of motion that each servo has. After dancing, MOJO returns to the basic scan and avoid behavior. If any of the acquisition behaviors fail for one reason or another, MOJO shakes his head to indicate failure and then returns to the basic behavior.

When controlling the servos on the arm, each behavior moves the arm slowly, by changing the pulse width a little bit at a time. This make the movements smoother and gives the sensors time to react when the arm moves around.

The basic software design goal was to have modular behaviors that could be turned on and off and also set certain triggers when sensor values go to a certain point. In my code, Appendix C, each behavior is a function called in a continuos loop, and one function takes in all of the triggers and starts and stops behaviors appropriately.

Conclusion

In summary I was able to get MOJO to recognize and pick up an object. Some of the behaviors, such as alignment, and gripping worked very well. Others, such as the acquire behavior that gets objects between the fingers performed poorly. Collision avoidance worked moderately well. The biggest problem that I found was that MOJO could not tell the difference between an object and approaching a wall at an angle. If the sensors on the fingers got different values from a wall at an angle, MOJO would attempt to align the object, when it failed, it would return to scan mode and immediately detect the same thing. This illustrates the fundamental flaw in my design. I believed that the majority of the robot's time would be spent looking for objects, and only a small portion would be spent looking at them. In fact the opposite is true. Even with many more sensors and much more code, scanning for objects, while trying to avoid obstacles is not practical. If it takes 30 seconds to recognize an object, then even if MOJO knows not to pick it up, it would have to inspect it again just to find out that it is the same object. Without some sort of complicated mapping system, MOJO would never know if he was checking the same object over and over again.

Instead there should be a fixed area to scan for objects, such as a conveyor belt, than can be started and stopped by the robot. In this case the environment is more restricted, and the robot can do tests on the object without worrying about the fact that it could be checking the same object over and over.

Still this is still an interesting challenge and even MOJO could be outfitted with more sensors and given more complicated behaviors to achieve a practical task. For future work I would like to improve the torque of a few servos to get more crisp and precise movements. I would like to perfect the color sensor, and add a more accurate distace ranging device. These improvements in combination with better software and a more restricted environment would allow me to accomplish all of the goals that this robot was originally conceived to do.

Appendix A - Vendor Information

- Force sensing resistor — Interlink Electronics (805)484-1331
  $3.50
- Color filters — Edmund Scientific (609) 573-6250
- Servos (including the clear 60 oz) — Hobby Shack (IMDL Home page)
  $10 - 42oz-in $22 - 60 oz-in
- Gear Head Motors — Jameco Electronics
  (www.jameco.com) part #155838
  $19.95

Appendix B - Advice for IMDLers

General Advice

- Work on your sensors from the very beginning.
- When constructing hardware try to stay away from glue, secure things with screws instead.
- Hardware stores have lots of good things, check out what they have to help in your mechanical design.
- Always get more servo power than you think you will need.
- Ask for engineering samples from companies, they give away a lot of free stuff.

Debugging your programs - a list of common mistakes

- Is the flag being cleared in your ISR?
- Is the ISR in the right spot in vectors.c?
- When is the next interupt being set to go off?
- Is the correct ISR being initialized? Are interupts being turned on?
- Using = instead of ==
- array[8] is indexed as array[0..7] **not** array[0..8]
- remember to run initialization routines
- use prototypes for functions
- don't #include the same file more than once, even in separate files
- variables declared inside a function will loose their value when you exit the function
- use -e in the comme11.bat file to accept C++ style code

Appendix C - Code

```
//-- Written by Preston Faiks, 1999
// The Main Robot Program

#include "vectors.c"
// ********* Global variables and functions from Included Files ********
// **** Only shows prototypes and variables intended for global use ****

// ** servo.c **
//            void init_servos(void);                    // initializes Servos

// ** ansmotor.c**
//  void init_motors(void);                 // initializes Motors
//  void motor(int index, int per_cent_duty_cycle)
//       Sets duty cycle and direction of motor specified by index
//       -100% <= per_cent_duty_cycle <= 100%  A negative % reverses the motor direction

// ** serial2.c **
//            void init_serial(void)                     // initializes Serial functions
//            char get_char(void)                        // gets character from serial port
//            void put_char(char outchar) // outputs character from serial port
//            void write(char strng[80])  // outputs a string of characters to serial port
//            void put_int(int number)       // outputs an integer to serial port
//            void write_int(int number)  // same as put_int with extra space at end

// ** analog2.c **
//            void AD_on(void);                          // Turn on A/D system
//            void AD_off(void);                         // Turn off A/D system
//            void AD_normal(void);                      // Update ANLG[0] to ANLG[7]
//            void AD_mux(void);                                 // Update ANLG[8] to ANLG[15]
//            void AD_all(void);                         // Update ANLG[0] to ANLG[15]
//            unsigned char ANLG[16];     // All analog values

// **digital.c **
//            void Update(void);                         // Updates all Digital Values
//            void Update_IN(void);                      // Update Digital Inputs
//            void Update_OUT(void);                     // Update Normal Outputs
//            void Update_40k(void);                     // Update 40k Outputs
//            void Update_Servo(void);     // Update Servo Values
//            unsigned char DO[8];                       // Normal Digital Outputs
//            unsigned char DI[8];                       // Normal Digital Inputs
//            unsigned char D40k[8];                     // 40kHz Digital Outputs
//            int spulse[5],spwr[5];                     // Servo Values for power and pulse length

// **display.c **
//            void display (void);             //Displays all Global values on Screen

/***************************** Prototypes ***********************************/
void Initialize(void);                          // Initializes the Robot
void Servo_Control(void);     // Manual Servo Control
int  poww(int , int);                           // like pow(x,y) but for intergers only
void SimpleAvoid(void);                         // Collision Avoidance
void Scan(void);              // Scan for Objects
void Detect(void);            // Detect Objects
void Allign(void);            // Allign Object
void Aquire(void);            // Aquire Object
void Move(void);              // Move the Object
```

```
void Shake(void);               // Shake Head
void Dance(void);               // Do victory Dance
void Evade(void);               // Quick Avoid
void Flaghandle(void);                  // Handles flags

/***************************** Globals **********************************/
enum modes {basic,allign,aquire,evade,move,dance,shake} mode;   // Operation Modes
int mmove ;  //move mode
char shakemode; //used for shake routine
int alligntimer; // timer for allign mode
int dancetimer; // timer for victory dance
int s0mode,s1mode,s2mode,s3mode,s4mode;  // used for dance
char alligned,gotit,objectnear,tooclose;
char toobig,movedone,fail,resume,eflag;          // sensor triggers
/************************** Main Program ********************************/

int main(void)
{
int multi;
INTR_OFF();
Initialize();  //This will turn Interupts back on
Update();
spulse[0] = 3000;  //base rotator
spulse[1] = 3800;  //wrist (up/down)
spulse[2] = 3000;  //wrist rotator
spulse[3] = 2800;  //gripper
spulse[4] = 3800;  //main arm
s0mode = s1mode = s2mode = s3mode = s4mode = 5;
mmove = 5;
alligntimer = 0;
multi=0;
mode = basic ;
objectnear = 0;
tooclose = 0;
toobig = 0;
fail = 0;
alligned = 0;
gotit = 0;
movedone = 0;
resume = 0;

while(1)
{
        multi+=1;
        if (multi >= 50) {multi=0;}
        Update();
        AD_all();
        if (multi == 1)       display();
        D40k[0] = 1; D40k[1] = 1; D40k[2] = 1;
        D40k[3] = 1; D40k[4] = 1; D40k[5] = 1;
        D40k[6] = 1; D40k[7] = 1;
//      Servo_Control();
        SimpleAvoid();
        Scan();
        Detect();
        Allign();
        Aquire();
        Move();
        Dance();
        Shake();
```

```
		Evade();
		Flaghandle();
}
}

/************************** Functions *********************************/

void Initialize()
{
		int i;
		for (i=0 ; i<=4 ;i++)
		{
			spulse[i] = 3000 ;
			spwr[i] = 1;
		}
		for (i=0 ; i<=7 ; i++)
		{
			D40k[i]=0;
			DO[i]=0;
		}
		init_serial();
		AD_on();
		init_servos();
		init_motors();
		motor(0,0);motor(1,0);
		INTR_ON();
}

void Servo_Control()
{
		int i,buffpulse;
		unsigned char tempnum,temppwr,temppulse[4];
		if (SCSR & 0x20)
		{
			write("Type Motor number ");
			tempnum = get_char();
			put_char(tempnum);
			tempnum = tempnum - 0x30;

			write("\n\rPower on/off ? (1/0) ");
			temppwr = get_char();
			put_char(temppwr);
			temppwr = temppwr - 0x30;

			write("\n\rNew Pulse width ? ");
			buffpulse = 0;
			for (i = 3; i>=0 ; i--)
			{
				temppulse[i] = get_char();
				put_char(temppulse[i]);
				temppulse[i] = temppulse[i] - 0x30;
				buffpulse = buffpulse + (temppulse[i] * poww(10,i));
			}

			write("\n\rPress Y to confirm: Num= "); put_int(tempnum);
			write("  Power= "); put_int(temppwr);
			write("  Pulse Length= "); put_int(buffpulse); write(" \n\r");
			if (get_char() == 'y')
			{
				put_char(0x1B);put_char(0x5B);put_char(0x32);put_char(0x4A);
```

```
                                spwr[tempnum] = temppwr;
                                spulse[tempnum] = buffpulse;
                                servo(tempnum,spulse[tempnum]);
                                power(tempnum,spwr[tempnum]);
                        }
        } //end if

}

int poww(int x, int y)
{
        int n,ans;
        ans = 1;
        for (n=1 ; n<=y ; n++)
        {
                ans = ans * x ;
        }
        return ans;
}

void SimpleAvoid()
{
        int diff;
        diff = ANLG[5] - ANLG[6];
        if (mode == basic)
        {
        //write("avoid ");
        if ((diff < -8)||(diff >8))
        {
                motor(1,5*diff);motor(0,-5*diff); //turn
        }
        else
        {
          if ((ANLG[5] >96) || (ANLG[6] >96))
                {motor(1,-50);motor(0,-50);} // back up
          else  {motor(1,50);motor(0,50);}   //go forward
        }
        }
        else {motor(1,0);motor(0,0);}  // stop
}

void Scan()
{
        if (mode == basic)
        {
        //write("scan ");
        if (spulse[0] > 4000) mmove = -3;
        if (spulse[0] < 2200) mmove = 3;
        spulse[0] += mmove;
        spulse[1] = 3800;  //wrist (up/down)
        spulse[2] = 3000;  //wrist rotator
        spulse[3] = 2550;  //gripper
        spulse[4] = 3800;  //main arm
        }
}

void Detect()
{
        int diff;
        diff = ANLG[3] - ANLG [4] ;
```

```
        if (mode == basic)
        {
        //write("detect ");
        if ((diff < -10) || (diff >10))
        { objectnear = 1;}
        else
        { if ((ANLG[3]>100)||(ANLG[4]>100))
          { tooclose = 1;}
        }
        }
}

void Allign()
{
        int diff;
        diff = ANLG[3] - ANLG [4] ;
        if (mode == allign)
        {
                write("allign ");
                alligntimer += 1;
                if ((diff < -5)&&(spulse[0] >2200))
                spulse[0]+= -1;
                if ((diff>5)&&(spulse[0] <4000))
                spulse[0]+= 1;
                if (((diff > -3)&&(diff <3))&&(alligntimer>400))
                  {
                   if ((ANLG[3]>100)||(ANLG [4]>100))
                     {toobig = 1;}
                   else
                        {alligned = 1;}
                  }
        }
}

void Aquire()
{
        if (mode == aquire)
        {
        write("aquire");
        motor(1,25);motor(0,25);
        if (ANLG[2] > 100)
        {eflag = 1;}
        if (eflag == 1)
        {
                motor(1,0);motor(0,0);
                spulse[3] += 3;
                if (ANLG[8] > 150)
                  {gotit = 1;}
                if (spulse[3] > 4000)
                  {fail = 1; spulse[3] =2550;}
        }
        }

}

void Evade()
{
        if (mode == evade)
        {
        //write("evade ");
```

```
            motor(0,100),motor(1,-100);
            if (spulse[4] <4200)
              {spulse[4] += 5;}
            if ((ANLG[3]<100)&&(ANLG[4]<100))
            {
              resume = 1;
              spulse[4] = 3800;
            }
            }
}

void Shake()
{
            if (mode == shake)
            {
             //write("shake ");
             if (shakemode == 0)
               {
                 spulse[2] += 15;
                 if (spulse[2] > 5000)
                   { shakemode = 1;}
               }
             if (shakemode == 1)
               {
                 spulse[2] += -15;
                 if (spulse[2] < 1800)
                   { resume = 1;
                          spulse[2] = 3000;
                         }
               }
             }
}

void Move()
{
            if (mode == move)
            {
              //write("move ");
              spulse[0] += -4;
              spulse[1] = 4500;
              spulse[4] = 3600;
              if (spulse[0] <= 2200)
               {
                        spulse[3] = 2900;
                        movedone = 1;
               }
            }
}

void Dance()
{
            if (mode == dance)
            {
              write("dance ");
              motor(0,-25);motor(1,25);
              if (spulse[0] >= 4000)
              { s0mode = -5 ; }
              if (spulse[0] <= 2200)
              { s0mode = 5 ; }
```

```
            if (spulse[1] >= 5000)
            { s1mode = -5 ; }
            if (spulse[1] <= 1800)
            { s1mode = 5 ; }

            if (spulse[2] >= 5000)
            { s2mode = -5 ; }
            if (spulse[2] <= 1800)
            { s2mode = 5 ; }

            if (spulse[3] >= 3800)
            { s3mode = -5 ; }
            if (spulse[3] <= 2800)
            { s3mode = 5 ; }

            if (spulse[4] >= 4600)
            { s4mode = -5 ; }
            if (spulse[4] <= 3500)
            { s4mode = 5 ; }

          spulse[0] += s0mode;  //base rotator
          spulse[1] += s1mode;  //wrist (up/down)
          spulse[2] += s2mode;  //wrist rotator
          spulse[3] += s3mode;  //gripper
          spulse[4] += s4mode;  //main arm
          if (dancetimer > 1000)
            { resume = 1;}
          dancetimer += 1;
          }
}

void Flaghandle(void)
{
          if (objectnear == 1)
          {
                    objectnear = 0;   // Clear Flag
                    alligntimer = 0;  // Set timer
                    mode = allign;  // Switch to align mode
          }
          if (tooclose == 1)
          {
                    tooclose = 0;          // Clear Flag
                    mode = evade;       // Switch to emergency avoid
          }
          if (alligned ==1)
          {
                    alligned = 0;          //Clear Flag
                    mode = aquire;       // Switch to Aquire mode
                    eflag = 0;
          }
          if (toobig == 1)
          {
                    toobig = 0;
                    mode = shake;   //Switch to shake head
                    shakemode = 0;
          }
          if (fail == 1)
          {
                    fail = 0;
                    mode = shake;   //Switch to shake head
```

```
                shakemode = 0;
        }
        if (gotit == 1)
        {
                gotit = 0;
                mode = move;    // Switch to move object
        }
        if (movedone == 1)
        {
                movedone = 0;
                dancetimer = 0;  // Clear timer
                mode = dance;       // Switch to dance mode
        }
        if (resume == 1)
        {
                resume =0;
                mode = basic;  //Switch back to basic mode
        }

}



/*********************************************************************
* Vectors for my Code
*
* Programmer: Preston Faiks
*********************************************************************/

/****************************** Includes *********************************/
#include <hc11.h>
#include <mil.h>
#include "servo.c"
#include "serial2.c"
#include "ansmotor.c"
#include "analog2.c"
#include "digital.c"
#include "display.c"

extern void _start(); /* entry point in crt11.s */
/*****************************************************************************/

/****************************** Vectors **********************************/
#pragma abs_address:0xffd6
/* change the above address if your vector starts elsewhere
 */
void (*interrupt_vectors[])() =
        {
        /* to cast a constant, say 0xb600, use
          (void (*)())0xb600
         */
        (void (*)())0x0,  /* SCI */
        (void (*)())0x0,  /* SPI */
        (void (*)())0x0,  /* PAIE */
        (void (*)())0x0,  /* PAO */
         (void (*)())0x0,  /* TOF */
        (void (*)())0x0,  /* TOC5 */
         (void (*)())0x0,  /* TOC4 */
   motor1,                          /* TOC3 */
   motor0,                         /* TOC2 */
```

```
   servo_ISR,                    /* TOC1 */
            (void (*)())0x0,  /* TIC3 */
            (void (*)())0x0,  /* TIC2 */
            (void (*)())0x0,  /* TIC1 */
            (void (*)())0x0,  /* RTI */
            (void (*)())0x0,  /* IRQ */
            (void (*)())0x0,  /* XIRQ */
            (void (*)())0x0,  /* SWI */
            (void (*)())0x0,  /* ILLOP */
            (void (*)())0x0,  /* COP */
            (void (*)())0x0,  /* CLM */
                        _start   /* RESET */
            };

#pragma end_abs_address


//-- Written by Preston Faiks, 1999


// 16 servo motor driver

// This code can only be used for servos.  Actually, the total pulse lengths
// can not add up to more than $FFFF(32 ms) This is fine for servos, but
// motors will need another ISR.
/****************************** Includes **********************************/
//#include <hc11.h>
//#include <mil.h>

/************************* Output ports where servos are ******************/
#define OUT1        *(unsigned char volatile *)(0x4000)
#define OUT2        *(unsigned char volatile *)(0x2000)

/**************************************************************************/

/***************************** Prototypes *********************************/
#pragma interrupt_handler servo_ISR()

void init_servos(void);
void servo(unsigned char,int);
void power(unsigned char, unsigned char);
void servo_ISR(void);
/**************************************************************************/


/****************************** Globals ***********************************/
unsigned int servo_pulse[16];
unsigned char servo_power[16];
unsigned char index;
const int last = 4 ; //* number of servos attached(0..last)*/
/**************************************************************************/


// This routine initializes the servos and the OC1 interupt

void init_servos()
{
  char i;
  INTR_OFF();
  OC1M = (0x00);     //interrupt will affect no pins in port A
```

```
  for(i = 0 ; i <= last ; i++)
  {
    if (i <= last )
    {
            servo_power[i] = 1;                     //Turn on servo
            servo_pulse[i] = 3000;          //Set to mid position
            }
    else
            {
                    servo_power[i] = 0;                     //Turn off servo
                    servo_pulse[i] = 3000;          //Set to mid position just in case
            }
  }

  TOC1 = 0; //start interupts at 0

  SET_BIT(TMSK1,0x80);  /* Enable OC1 interrupt */

//  INTR_ON();
}


void power(unsigned char servonum, unsigned char on_off)
{
            if (servonum <= last )
      {
        servo_power[servonum] = on_off;
      }

}


void servo(unsigned char servonum, int pulse_len)
{
  if (servonum <= last )
      {
        servo_pulse[servonum] = pulse_len;
      }


}

void servo_ISR()
{

            CLEAR_FLAG(TFLG1,0x80); // Clear OC1 flag

            while ( (servo_power[index]==0) && (index<=last))
                    index++;                                                // Skip to servo that is ON

            if ((index < 7) && (index <= last))
                    {
                     OUT1 = (1 << index);                   //index bit goes on, others are turned off
                     TOC1 += servo_pulse[index]; //it will stay high for this length
                     index++;
                    }
            else
              if (index <= last)
                    {
```

```
                OUT1 = (0x00);
                OUT2 = (1 << (8 - index));    //index bit goes on, others are turned off
                TOC1 += (servo_pulse[index]); //it will stay high for this length
        }
        else
        {
                TOC1 = (0x0000);   // All servos are done
                index = 0 ;                     // next Interupt will start with index=0
                OUT1 = OUT2 = (0x00);
        }
}
```

```
//-- Written by Preston Faiks, 1999


// Analog input reader for systems with or without analog MUX

// This code will read all of the analog inputs and store them
// to the global array variable ANLG[15]

// ANLG[0] to ANLG[7] represent the normal Port E inputs
// ANLG[8] to ANLG[15] represent the analog mux inputs

// NOTE: since ANLG[7] is used by the analog MUX its value
//     should never be used with an AMUX setup.  Use 8-15 instead.

/***************************** Includes **********************************/
//#include <hc11.h>
//#include <mil.h>
#define MUX         *(unsigned char volatile *)(0x5000)

/***************************** Prototypes ********************************/
void AD_on(void);               // Turn on A/D system
void AD_off(void);              // Turn off A/D system
void AD_normal(void);           // Update ANLG[0] to ANLG[7]
void AD_mux(void);                      // Update ANLG[8] to ANLG[15]
void AD_all(void);              // Update ANLG[0] to ANLG[15]
/*************************************************************************/


/***************************** Globals ***********************************/
unsigned char ANLG[16];

/*************************************************************************/


// This routine turns on the A/D system
void AD_on()
{
        int wait;
        SET_BIT(OPTION,0x80);
        asm("psha\n"                            // 3 cycles
                "ldaa #0x26\n"                  // 2 cycles
```

```
                "lloop1 : deca\n"     // 2 cycles
        "bne lloop1\n"                // 3 cycles
        "pula");                      // 4 cycles
   //wait 200 e-cycles for system to power up
}

// This routine turns off the A/D system
void AD_off()
{
        CLEAR_BIT(OPTION,0x80);
}

// This routine updates ANLG[0] to ANLG[7]
void AD_normal(void)
{
        ADCTL = 0x10 ;  //Start Conversion
        while((ADCTL & 0x80) != 0x80);
          //wait for conversion to complete
        ANLG[0] = ADR1;
        ANLG[1] = ADR2;
        ANLG[2] = ADR3;
        ANLG[3] = ADR4;

        ADCTL = 0x17 ;  //Start Conversion
        while((ADCTL & 0x80) != 0x80);
          //wait for conversion to complete
        ANLG[4] = ADR1;
        ANLG[5] = ADR2;
        ANLG[6] = ADR3;
        ANLG[7] = ADR4;
}

//This routine updates ANLG[8] to ANLG[15]
void AD_mux(void)
{
        unsigned char i;
        for (i=0 ;i<=7 ; i++)
        {
                MUX = i;
                ADCTL = 7;  //Start Conversion
                while((ADCTL & 0x80) != 0x80);

//              asm("psha\n"                            // 3 cycles
//                      "ldaa #0x05\n"                  // 2 cycles
//                      "lloop2 : deca\n"     // 2 cycles
//      "bne lloop2\n"                  // 3 cycles
//      "pula");                        // 4 cycles
        //wait 34 E-cycles for conversion to complete

        ANLG[8+i] = ADR1;           //Store Value
   }
}

//This routine updates ANLG[0] to ANLG[15]
void AD_all(void)
{
        AD_normal();
        AD_mux();
}
```

```
//-- Written by Preston Faiks, 1999

#define DOUT       *(unsigned char volatile *)(0x5000)
#define DIN        *(unsigned char volatile *)(0x4000)
#define DOUT40k *(unsigned char volatile *)(0x7000)

/*************************** Prototypes *********************************/
void Update(void);              // Updates all Values
void Update_IN(void);           // Update Digital Inputs
void Update_OUT(void);      // Update Normal Outputs
void Update_40k(void);      // Update 40k Outputs
void Update_Servo(void);// Update Servo Values

/************************************************************************/


/****************************** Globals *********************************/
unsigned char DO[8], DI[8], D40k[8] ;
unsigned int spulse[5];
unsigned char spwr[5];

/************************************************************************/


// This routine Updates the Digital Inputs
void Update_IN()
{
        unsigned char temp;
        temp=DIN;
        DI[0] = ((temp & 1)==1) ;
        DI[1] = ((temp & 2)==2) ;
        DI[2] = ((temp & 4)==4) ;
        DI[3] = ((temp & 8)==8) ;
        DI[4] = ((temp & 16)==16) ;
        DI[5] = ((temp & 32)==32) ;
        DI[6] = ((temp & 64)==64) ;
        DI[7] = ((temp & 128)==128) ;

}


// This routine Updates the Digital Outputs
void Update_OUT()
{
        unsigned char temp,i;
        temp = 0;
        for (i=0 ;i<=7 ;i++)
        {
                DO[i] = (DO[i] && 1);  //Ensures DO is either 0 or 1
                temp += (DO[i] << i);
        }
        DOUT = temp;
}


// This routine Updates the 40kHz Digital Outputs
void Update_40k()
{
        unsigned char temp,i;
        temp = 0;
```

```
        for (i=0 ;i<=7 ;i++)
        {
                D40k[i] = (D40k[i] && 1);  //Ensures D40k is either 0 or 1
                temp += (D40k[i] << i);
        }
        DOUT40k = temp;
}


// This routine Updates the Servo values
void Update_Servo()
{
        int i;
        for (i=0 ;i<=4 ;i++)
        {
                spwr[i] = (spwr[i] && 1);

                if (spulse[i] > 5000)
                        spulse[i] = 5000;

                if (spulse[i] < 1200)
                        spulse[i] = 1200;

                power(i,spwr[i]);
                servo(i,spulse[i]);
        }
}


// This routine Updates all Values
void Update()
{
        Update_IN();        // Update Digital Inputs
        Update_OUT();       // Update Normal Outputs
        Update_40k();       // Update 40k Outputs
        Update_Servo();     // Update Servo Values
}
//-- Written by Preston Faiks, 1999

void display (void);

void display()
{
        int i;
//      put_char(0x1B);put_char(0x5B);put_char(0x32);put_char(0x4A);
   put_char(0x1B);put_char(0x5B);put_char(0x3B);put_char(0x48);
//  ANSI sequence to clear screen and move cursor to home

        write("Analog Inputs\n\r");

        write("ANLG[0]: "); put_int(ANLG[0]); write("    ");
        write("ANLG[1]: "); put_int(ANLG[1]); write("    ");
        write("ANLG[2]: "); put_int(ANLG[2]); write("    ");
        write("ANLG[3]: "); put_int(ANLG[3]); write("       \n\r");
        write("ANLG[4]: "); put_int(ANLG[4]); write("    ");
        write("ANLG[5]: "); put_int(ANLG[5]); write("    ");
        write("ANLG[6]: "); put_int(ANLG[6]); write("    ");
        write("ANLG[7]: "); put_int(ANLG[7]); write("       \n\r");
        write("ANLG[8]: "); put_int(ANLG[8]); write("    ");
        write("ANLG[9]: "); put_int(ANLG[9]); write("    ");
```

```
        write("ANLG[10]: "); put_int(ANLG[10]); write("    ");
        write("ANLG[11]: "); put_int(ANLG[11]); write("        \n\r");
        write("ANLG[12]: "); put_int(ANLG[12]); write("    ");
        write("ANLG[13]: "); put_int(ANLG[13]); write("    ");
        write("ANLG[14]: "); put_int(ANLG[14]); write("    ");
        write("ANLG[15]: "); put_int(ANLG[15]); write("        \n\r");

        write("\n\r Digital Inputs\n\r");
        write("DI[0]  DI[1]  DI[2]  DI[3]  DI[4]  DI[5]  DI[6]  DI[7]\n\r   ");
        for (i=0 ;i<=7 ;i++)
        {
                put_int(DI[i]);
                write("      ");
        }
        write("\n\r\n\r");


        write("Normal Digital Outputs\n\r");
        write("DO[0]  DO[1]  DO[2]  DO[3]  DO[4]  DO[5]  DO[6]  DO[7]\n\r   ");
        for (i=0 ;i<=7 ;i++)
        {
                put_int(DO[i]);
                write("      ");
        }
        write("\n\r\n\r");


        write("40kHz Digital Outputs\n\r");
        write("D40k[0]  D40k[1]  D40k[2]  D40k[3]  D40k[4]  D40k[5]  D40k[6]  D40k[7]\n\r     ");
        for (i=0 ;i<=7 ;i++)
        {
                put_int(D40k[i]);
                write("        ");
        }
        write("\n\r");

        write("Servo Status\n\r");
        for (i=0 ; i<=4 ; i++)
        {
                write("Servo "); put_int(i); write(" : ");
                put_int(spulse[i]);
                put_char(' ');
                put_int(spwr[i]);

                write("    ");
                put_int(servo_pulse[i]);
                put_char(' ');
                put_int(servo_power[i]);
                write("\n\r");
        }
}
/*
 *  Title:      serial2.c
 *  Programmer:  Reid Harrison recoded for ICC11 by Scott Jantz
 *  Date:       July 4,1996
 *  Updated:            February 1, 1999 by Preston Faiks
 *          Version:    2
 *  Description:
 *      Serial Port I/O Routines for ICC11 on the 68HC11
 *      init_serial must be called in order to use any of the
```

```
 *       functions.  Enables ICC11 programs to use the serial
 *       port without buffalo
 *
 *       include in programs that need to use the serial port for I/O
 *       interface with baud=9600, 8 data bits, 1 stop bit, parity none, no flow control
 *       requires hc11.h to work
 */

/*************Functions*******************************/

/*
  init_serial

  Initializes the SCI port on the 68HC11 to operate at 9600 baud.  This
  function must be called at the beginning of your program if you wish
  to use any of the functions in this library.

  Example:
    init_serial();
*/
//#include <hc11.h>
//#include <mil.h>

void init_serial()
{
  CLEAR_BIT(SPCR,0x20);
  BAUD = 0xb0;  /* 0xbO is 9600 0x35 is 300 baud */
  SCCR2 = 0x0C; /* Enable Transmit and Recieve*/
}


/*
  get_char

  Waits for a character to be received by the serial port, then returns
  its ASCII value.

  Examples:
    x = get_char();
    if (get_char() == 'F') fd(0);
    get_char();
*/
char get_char()
{
  int test = 0;

  while (test == 0) {

              test = SCSR & 0x20;
  }

  return(SCDR);
}


/*
  put_char

  Writes an ASCII character to the serial port.
```

```
  Examples:
    put_char(65);
    put_char('A');
*/
void put_char(char outchar)
{
  int test = 0;

  while (test == 0)
  {
            test = SCSR & 0x80;
  }

  SCDR = outchar;
}

/*
  write

  Writes a string of text to the serial port.
  Any escape sequence supported by ICC11 will work.

  Sequence          Value     Char      What it does

            \b                0x08      BS              Backspace
            \f                0x0C      FF              Formfeed
            \n                0x0A      LF              Newline (linefeed)
            \r                0x0D      CR              Carriage return
            \t                0x09      HT              Tab (horizontal)
            \v                0x0B      VT              Vertical tab
            \\                0x5c      \               Backslash
            \'                0x27      '               Single quote (apostrophe)
            \"                0x22      "               Double quote
            \?                0x3F      ?               Question mark


  Examples:
    write("Hello, world!\n\r");
    write("one\n\rtwo\n\rthree\n\r");
*/
void write(char strng[80])
{
  int index = 0;

  while(strng[index] != 0)
  {
    put_char(strng[index]);
    index++;
  }
}


/*
  put_int

  Writes an integer to the serial port.
*/
void put_int(int number)
{
  char digits[5] ;
```

```
  int extra;
  int i= 0 ;

  if (number < 0)
  {
    put_char('-');
    number *= -1;
  }

  if (number == 0)
  {
            put_char('0');
  }

  if (number != 0)
  {
    digits[0] = number/10000;
    extra = digits[0] * 10;

    digits[1] = (number/1000) - (extra);
    extra = (extra+digits[1]) * 10 ;

    digits[2] = (number/100) - (extra);
    extra = (extra+digits[2])*10;

    digits[3] = (number/10) - (extra);
    extra = (extra+digits[3]) * 10 ;

    digits[4] = number - extra;

            while(digits[i] == 0)
              i++;

            while(i <= 4)
                    put_char(digits[i++] + 48);
  }
}

/*
  write_int

  Print an integer to the serial port prefixed by a space and followed
  by a newline.

  Examples:
    write_int(x);
    write_int(analog(7));
*/
void write_int(int number)
{
  put_char(' ');
  put_int(number);
  put_char(13);
  put_char(10);
}
#include "vectors.c"
//#include "serial2.c"
//#include "servo.c"

int poww(int , int);
```

```
int main(void)
{       int pulse[5],pwr[5];
        int i,buffpulse;
        unsigned char tempnum,temppwr,temppulse[4];
        INTR_OFF();
        init_serial();
        init_servos();
        INTR_ON();
        for (i=0 ; i<=4 ;i++)
        {
                pulse[i] = 2000 ;
                pwr[i] = 1;
        }
        while (1) {
        write("Servo test and config program\n\r");
        write("Current Values\n\r");
        for (i=0 ; i<=4 ; i++)
        {
                write("Servo "); put_int(i); write(" : ");
                put_int(pulse[i]);
                put_char(' ');
                put_int(pwr[i]);
                write(" \n\n\r");
        }
        write("Type Motor number ");
        tempnum = get_char();
        put_char(tempnum);
        tempnum = tempnum - 0x30;

        write("\n\rPower on/off ? (1/0) ");
        temppwr = get_char();
        put_char(temppwr);
        temppwr = temppwr - 0x30;

        write("\n\rNew Pulse width ? ");
        buffpulse = 0;
        for (i = 3; i>=0 ; i--)
        {
                temppulse[i] = get_char();
                put_char(temppulse[i]);
                temppulse[i] = temppulse[i] - 0x30;
                buffpulse = buffpulse + (temppulse[i] * poww(10,i));
        }

        write("\n\rPress Y to confirm: Num= "); put_int(tempnum);
        write("  Power= "); put_int(temppwr);
        write("  Pulse Length= "); put_int(buffpulse); write(" \n\r");
        if (get_char() == 'y')
        {
                write("Values changed.\n\r\n\r\n\r");
                pwr[tempnum] = temppwr;
                pulse[tempnum] = buffpulse;
                power(tempnum,pwr[tempnum]);
                servo(tempnum,pulse[tempnum]);
        }
}
 return 0;
}
```

```
int poww(int x, int y)
{
        int n,ans;
        ans = 1;
        for (n=1 ; n<=y ; n++)
        {
                ans = ans * x ;
        }
        return ans;
}


/* Title        motor.c
 * Programmer   Keith L. Doty.
 * Date         June 19, 1996
 * Version      1
 * Description
 *      This module includes motor initialization, motor speed control
 *      and two PWM interrupt drivers motor0 and motor1.
 *      motor0 uses OC2 and motor1 uses OC3.
 *      This module can be used with the TALRIK robot
 *
 */

/***************** Includes ************************************/
//#include <hc11.h>
//#include <serial.h>
//#include <mil.h>

/***************** Constants ***********************************/
#define PERIODM 65,500
#define PERIOD_1PC 655

#pragma interrupt_handler motor0 motor1
void motor0();
void motor1();

/********************** Data ***************************************/
int duty_cycle[2];  /* Specifies the PWM duty cycle for two motors */


/*************** Functions *************************************/
void init_motors(void)
/* Function: This routine initializes the motors
 * Inputs:   None
 * Outputs:  None
 * Notes:    This routine MUST be called to enable motor operation!
 */

{

  INTR_OFF();

/* Set OC2 and OC3 to output low */
  SET_BIT(TCTL1,0xA0);
  CLEAR_BIT(TCTL1,0x50);

/* Set PWM duty cycle to 0 first */
   duty_cycle[0] = duty_cycle[1] =0;
```

```
/* Associate interrupt vectors with motor routines */
/*
  *(void(**)())0xFFE6 = motor0;
  *(void(**)())0xFFE4 = motor1;
*/

/* Enable motor interrupts on OC2 and OC3 */
  SET_BIT(TMSK1,0x60);


/* Specify PD4 and PD5 as output pins.
 * PD4 controls direction of Motor 1 and PD5 the direction of Motor 0.
 */

  SET_BIT(DDRD,0x30);
  INTR_ON();
}

void motor(int index, int per_cent_duty_cycle)
/* Function: Sets duty cycle and direction of motor specified by index
 * Inputs:   index in [0,1]
 *           -100% <= per_cent_duty_cycle <= 100%
 *           A negative % reverses the motor direction
 * Outputs:  duty_cycle[index]
 *           0 <= duty_cycle[index]<= PERIOD (Typically, PERIOD = 65,500)
 * Notes:    Checks for proper input bounds
 */
{
  if (per_cent_duty_cycle < 0)
     {
              per_cent_duty_cycle = -per_cent_duty_cycle; /* Make positive */
     /* Set negative direction of motors */
              if (index == 0) CLEAR_BIT(PORTD,0x20);
              if (index == 1) CLEAR_BIT(PORTD,0x10);
     }
  else
     {
     /* Set positive direction of motors */
              if (index == 0) SET_BIT(PORTD,0x20);
              if (index == 1) SET_BIT(PORTD,0x10);

     }

/* At this point per_cent_duty_cycle must be a positive number less
 * than 100. If not make it so.
 */
  if (per_cent_duty_cycle > 100) per_cent_duty_cycle = 100;
  duty_cycle[index] = per_cent_duty_cycle*PERIOD_1PC;

}

void motor0 ()
/* Function: This interrupt routine controls the PWM to motor0 using OC2
 * Inputs:   duty_cycle[0] (global)
 * Outputs:  Side effects on TCTL1, TOC2, TFLG1.
 * Notes:    init_motors() assumed to have executed
 */

{
```

```
/* Keep the motor off if no duty cycle specified.*/


  if(duty_cycle[0] == 0)
   {
    CLEAR_BIT(TCTL1, 0x40);
   }
  else
    if(TCTL1 & 0x40)
     {
      TOC2 += duty_cycle[0];                /* Keep up for width */
      CLEAR_BIT(TCTL1,0x40);                    /*  Set to turn off */
      /* Set XPORTB bit */
     }
    else
     {
       TOC2 += (PERIODM - duty_cycle[0]);
       SET_BIT(TCTL1,0x40);                 /* Set to raise signal */
       /* Clear XPORTB bit */
     }
  CLEAR_FLAG(TFLG1,0x40);                       /* Clear OC2F interrupt Flag */
}

void motor1()
/* Function: This interrupt routine controls the PWM to motor1 using OC3
 * Inputs:   duty_cycle[1] (global)
 * Outputs:  Side effects on TCTL1, TOC2, TFLG1.
 * Notes:    init_motors() assumed to have executed
 */

{
 /* Keep the motor off if no duty cycle specified.*/


  if(duty_cycle[1] == 0)
   {
    CLEAR_BIT(TCTL1, 0x10);
   }
  else
    if(TCTL1 & 0x10)
     {
      TOC3 += duty_cycle[1];                /* Keep up for width */
      CLEAR_BIT(TCTL1,0x10);                    /*  Set to turn off */
     }
    else
     {
      TOC3 += (PERIODM - duty_cycle[1]);
      SET_BIT(TCTL1,0x10);                  /* Set to raise signal */
     }
  CLEAR_FLAG(TFLG1,0x20);                       /* Clear OC3F interrupt Flag */
}
```