Phil Thomas

Intelligent Machine Design Lab--EEL5666

Final Report

Tuesday, April 23, 2002

Sister Roboto

Table of Contents

ABSTRACT.......... 3
EXECUTIVE SUMMARY 4
INTRODUCTION 5
INTEGRATED SYSTEMS 5
MOBILE PLATFORM.......... 7
ACTUATION 8
SENSORS 8
Bump Sensors:.......... 9
IR Detector:.......... 9
Bend Sensor: 10
Microswitches: 10
BEHAVIORS.......... 11
Searching:.......... 11
Line up: 11
Punching:.......... 12
Celebration:.......... 12
EXPERIMENTAL RESULTS.......... 13
CONCLUSION.......... 14
APPENDIX A.......... 15
APPENDIX B.......... 28

ABSTRACT

Sister Roboto searches for targets and attempts to knock them over using a punching nun finger puppet. If she is successful at knocking over a target, she does a victory dance and moves on to the next target. If she is unsuccessful at knocking down her target after an allotted period of time, she pouts and then moves on to a new target.

EXECUTIVE SUMMARY

Sister Roboto is designed to seek out targets and knock them over by punching them. Her punching mechanism is controlled by two servos, which actuate a pair of levers. The levers, in turn, extend her arms. On the tips of the arms are two switches that detect when a hit has been made. Also attached to each of her arms, is a bend sensor, which measures the degree of extension of her arms. All of her destructive power would be wasted if she could not move around and search for targets, so *Sister Roboto* has two wheels at her base, each powered by a hacked servo.

Sister Roboto does not distinguish between targets; there is no need to. However, she is designed for a specific type of target in mind, those which are both sturdy enough to trigger her bump sensor, but unsteady enough to be knocked down by her punches. At the center of *Sister Roboto*'s electronics is a Tecel Electronics P500 board, which contains a Philips 80C552 microcontroller. *Sister Roboto* draws her power from 8 rechargeable AA NiMH batteries and relies on a series of 5 bump sensors, an IR detector, two flex sensors and a pair of microswitches to tell her about her surroundings.

INTRODUCTION

The world is full of things that could and should be knocked over. From fruit displays at the grocery store to precariously leaning Italian monuments, there are items all over the world that are practically begging to be pummeled until they come toppling down. My robot, *Sister Roboto*, sets out, on a very diminished scaled, to make things right in the world by punching random objects and attempting to knock them down. The main objective of my project was to design an autonomous punching machine using a punching nun finger puppet.

INTEGRATED SYSTEMS

I built *Sister Roboto* using a TJ inspired design I created myself. The entire robot can be seen in figure one. The microcontroller board uses the onboard sensors to detect the state of *Sister Roboto*'s surroundings, and then uses the servos to position it for optimal punching. Once properly positioned, Sister Roboto unleashes a barrage of punches on her target, hopefully knocking it over. If she is successful in knocking over her target, she celebrates by spinning around, if she is unsuccessful, she shakes back and forth. After either pouting or celebrating, *Sister Roboto* moves on to find another target.

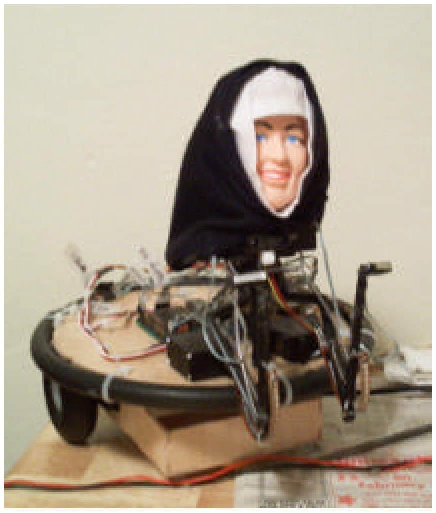

Figure 1 ***Sister Roboto***

The microcontroller I used for *Sister Roboto*, the Philips 80C552, an 8051 derivative, was very well suited for controlling a robot. The chip features 8 10-bit A/D converters and 2 PWM channels as well as the rest of the 8051's normal features. The first 80C552 I received had a broken A/D converter; I had to buy a new one

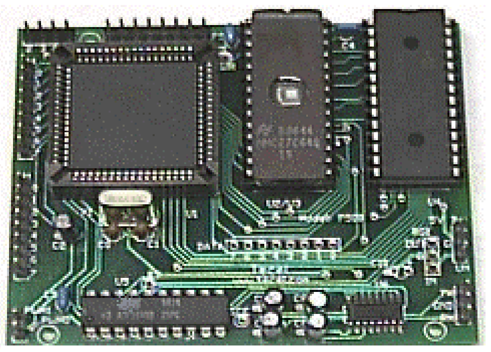
Figure 2 P500 Board

from Tecel. The P500 board, seen in figure two, was well designed, however the accompanying EPROM monitor and development environment had some major problems. When using the included EPROM monitor, a developer is forced to use a DOS based program for editing, compiling and downloading his code. As a Linux user, I did not appreciate this fact especially since it was not advertised as such on the dealer's website. Throughout the course of the semester, I was unable to get the development program to run under Windows 98 in the IMDL lab, I always had to reboot the machine into DOS mode. The DOS program seems to communicate its intention to upload a program with the EPROM monitor using non-printing control characters that I could not reproduce in a terminal program. The included "Small-C" compiler had several galling bugs. If more than 32 variables are used, the compiler silently fails, producing invalid code that executes partway, but then fails. Although the documentation claims that the compiler supports arrays, elements of arrays cannot be changed and are always equal to 0xFFFF. The compiler does not support floating point, has irregular syntax and functions can neither be passed parameters, nor do they return values. The only error message ever returned by the compiler in the case of a failed compile is "Invalid character" and a line number that the invalid character appears on. There does exist an open source monitor for the 8051 called PaulMon, which in retrospect, I should have used. With this monitor, I would have been able to write standard ANSI C code and compiled it with SDCC, a GCC based microcontroller compiler. This combination of tools would have made the development of *Sister Roboto* much easier.

MOBILE PLATFORM

Sister Roboto's mobile platform is a TJ-inspired design which I designed myself in AutoCAD and cut out of a balsa wood sheet using the T-Tech machine. Figure three shows the constructed platform along with the skeleton of the punching nun finger puppet inserted in the hole in the front. Two wheels are located towards the back of the robot with a caster in the front. I probably could have made my platform smaller, as it is, there is a lot of unused space on the edges and in the back. The batteries sit underneath the platform and the microcontroller board is bolted onto the top.

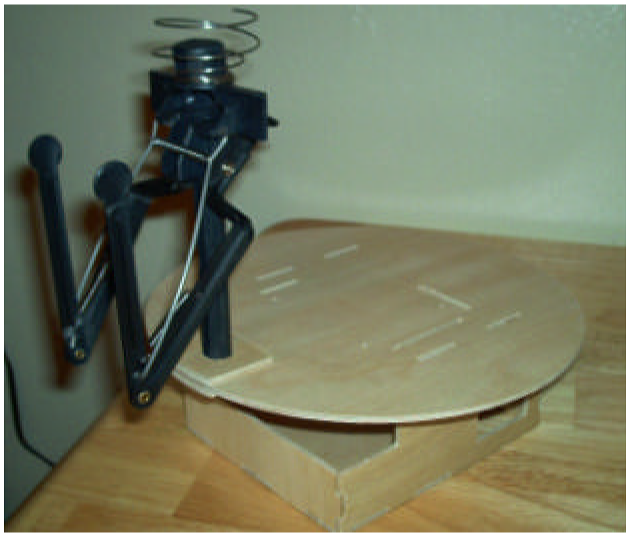
Figure 3 Mobile Platform plus finger puppet frame

ACTUATION

Sister Roboto has 4 servos, two standard and two hacked. The hacked servos control the wheels while the other two are glued onto the top of the platform next to the finger puppet. These servos are attached to the levers, which control the extension of the arms via wire linkages. Figure four shows how the arm-extending levers operate. A hole was drilled in the plastic of the lever and a wire was attached from the hole to the arm on the corresponding servo. This setup allowed for an adequate range of motion and strength of punch, I believe that in order to punch faster or harder, I would have had to upgrade to a stronger, more expensive servo than the standard ones I used. When dealing with my servos, I learned the hard way that it pays to have all wires clearly labeled, I reversed the polarity of one of my servos and burned it up, forcing me to purchase a new one. This gave rise to my perhaps distracting, but thorough labeling system.

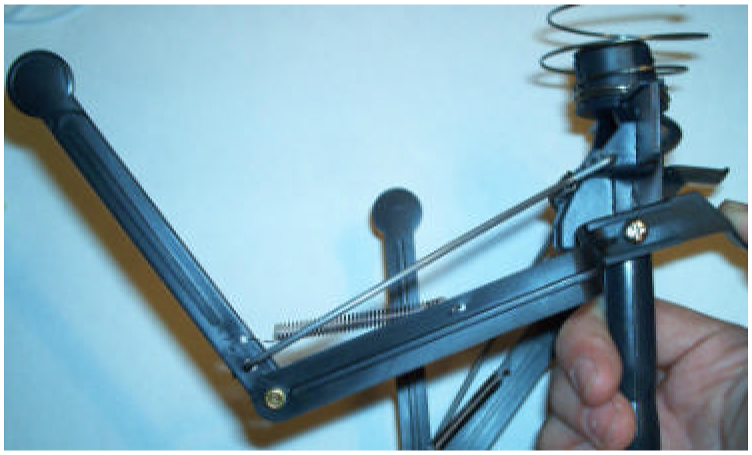
Figure 4 Arm Extension

SENSORS

My robot has 4 different types of sensors, bump detecting sensors on the periphery of the platform, a single IR detector attached to the middle of the finger puppet, a bend sensor for each arm, and a switch for the tip of each arm.

Bump Sensors:

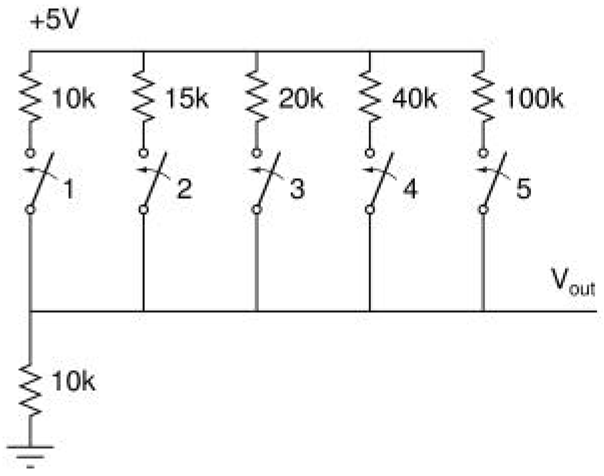

Figure 5 Bump Sensors

I arranged my bump sensors in the resistor network shown in figure five. Each switch in the figure represents one of the push buttons we were given in lab. These switches are arranged along the outer edge of the platform and a rubber hose is run over the top of them so bumping into an object triggers them. The circuit is a simple voltage divider when any of the switches are closed. My code uses a series of if statements to determine which buttons are pressed after using an A/D channel to read V_{out}. The values corresponding to each button were determined experimentally and are shown in my code in appendix A.

IR Detector:

For my robot, I used an integrated IR detector package purchased from Jameco. My board, unlike the Mekatronix board, did not have a simple way to generate a 30Khz signal needed to drive an IR LED to use a Sharp IR can, so I decided to use the integrated package. The device simply plugs into a power source and supplies an analog voltage depending on the distance of the object the IR beam is reflected from. The further away an object is, the lower the analog value, peaking at a digital value of about 800, and then falling off slightly as the object gets closer until the analog reading is zero because the object is blocking either the transmitting LED or the detector. *Sister Roboto* uses her IR detector to scan for objects she can hit. I believe that my design could benefit from another IR detector, I purchased two, but one broke, these detectors are very fragile. The second detector could be placed in the front at the very bottom to aid in avoiding already knocked over targets, which seem to be a problem.

Bend Sensor:

My robot has a bend sensor mounted on each arm to help it determine how far each arm is extended. I purchased these flex sensors from Jameco. When they are straight, the resistance across the two terminals is about 10kΩ, when they are bent, the resistance reaches about 40kΩ. The interface circuit for the flex sensor is shown in figure six.

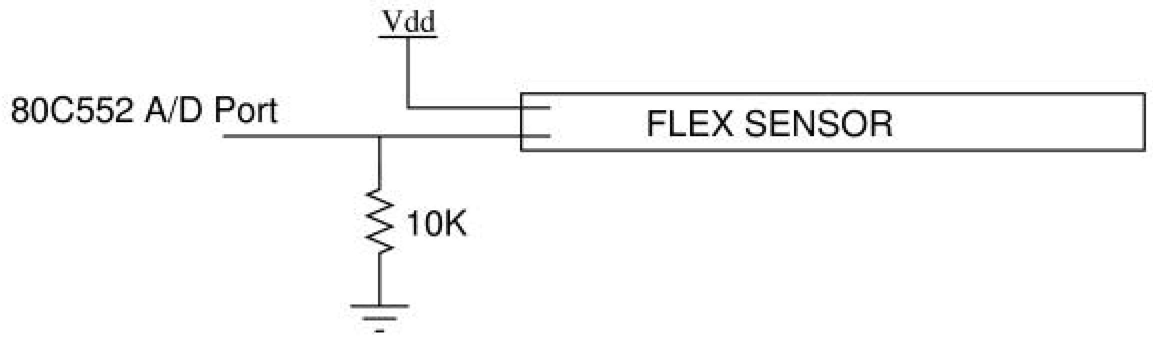

Figure 6 Flex Sensor Circuit

Microswitches:

Probably the most important of the sensors in my design, I got the microswitches on the tips of the arms from RadioShack. At first I tried using the bump sensors we were given in lab, but they had far too great a resistance to being pushed, these sensors have a little metal bar covering the button, which makes it much easier to push. At first, I had this mounted vertically but it was not quite easy enough to push. At the advice of Aamir, I mounted the switches horizontally, thereby making the switches at just the right angle so that when the robot hits something, it actually trips the switch. These switches are wired to the external interrupt which, when pressed, set a flag called rhit or lhit, depending on which arm scored the hit, this flag helps the robot know how far it needs to extend its arm for the next punch. There really is no circuit involved with the microswitches; they simply connect the external interrupt pin to +5V.

BEHAVIORS

Sister Roboto has four basic behaviors, searching, lining-up, punching and celebrating.

Searching:

At the very beginning, *Sister Roboto* sits in a tight loop until the back bumper is pressed. Then she begins searching using her IR detector to scan for a target and performing obstacle avoidance with her bump sensors. First she turns a random distance, all the while scanning with her IR detector to see if it sees anything closer than IR_LONG_THRESH. If she does, she stops immediately and begins moving forward, if she does not see anything this close, she completes the random turn and begins to move forward. She moves forward until the IR detector senses something closer than IRTHRESH. When it does it moves into the next mode.

Line up:

This mode is the first time the external interrupts are used, so they are activated and the arms are extended one bit at a time. They are extended until they either are at their maximum and haven't hit anything, or until they receive a hit. If one arm receives a hit and the other doesn't, the robot turns in the appropriate direction and tries again. If neither arm scores a hit, the robot checks the IR detector to make sure that the target is still there, if so, it moves forward a bit and starts line_up again, if not, it goes back into search mode. Once both arms have detected a hit in line_up mode, the robot moves into punching mode.

Punching:

When *Sister Roboto* enters punching mode, she knows how far she has to extend each arm to hit her target. So, she executes a random series of combinations with the external interrupts enabled. This allows her to make sure she is still hitting her target. Finally, when she has either executed 4 combos, or determined that her target is knocked down, *Sister Roboto* goes into celebrate mode.

Celebration:

Sister Roboto celebrates when she thinks that she has knocked over her target. She backs up, spins back and forth and then goes back into search mode. If she was unable to knock down her target, she backs up and shakes back and forth then goes back to search mode.   I was originally going to have music play while she was celebrating, but I was unable to implement this.

EXPERIMENTAL RESULTS

On demo day, *Sister Roboto* did not quite perform as well as she could have. I did not spend much time testing the targets I selected; I used several different books, some of which were far too short to even be detected by the IR detector. I already mentioned the poor placement of the touch sensors on the tips of the arms: even when she should have been registering hits, she was not. Since then I've significantly tweaked the code, rotated the bump sensors on the tips of the arms so they are better able to make contact, and I have selected better targets and tested them all. I am much more satisfied with my robots performance now.

CONCLUSION

My primary goal in my robot design was a punching robot capable of knocking over random targets. I believe I have fulfilled this goal. Despite several setbacks including compiler errors, burned out servos, and problems with my A/D converter, I was able to accomplish my goal. For a while, I did not think that *Sister Roboto* would actually be able to effectively knock down many targets, with the tweaks I've made since demo day thought, I am very pleased with her ability to wreak havoc. The main limitations of my design are the relative weakness of the arm extending servos, the lack of possible program optimization due to the faulty compiler, and the lack of music during celebration. If I could start the project over from scratch, I certainly would use a different development environment, I probably would get a bit more expensive servos, and I would have better investigated my options regarding playing music.

APPENDIX A

Program Code:

All of the code for the robot was organized in one file, robot.c since the compiler was unable to deal with include files or compiling from multiple files. I apologize for any inconsistent indentation; my editor did not like the fact that no lines ended in semicolons.

```
int line, ad_result, i, j, k, l, button, rand, z
int lbend_rest, rbend_rest, lineup_count
int lbend_result, rbend_result, count
int ltarget_dist, rtarget_dist, ltarget_bend, rtarget_bend
int bend_result, ir_result, rdist, ldist
int lhit, rhit, rand_dir, distance, arm, wheel
int hi, lo, arm_wheel, dir, extnd, retrct, diff, final
int  s
int contact, combo_cnt

define BUMP 0
define IR 1
define LBEND 3
define RBEND 4
define IRTHRESH 550
define IR_LONG_THRESH 150
define IR_TOO_CLOSE 750
define WHEEL 0
define ARM 1
define FORWARD 0
define STOP 1
define BACK 2
define RIGHT 3
define LEFT 4
define BOTH 5

define RF 220
define RB 200
define LF 240
define LB 250

define REXT 224
define RRET 190
define LEXT 159
define LRET 187

        org 0x8000

main(){
    main2()
}

        org 0x8003

exint0vector(){
    extint0()
    }

        org 0x8013

exint1vector(){
    extint1()
    }

        org 0x8100

main2(){
```

```
// enables free running counter
TMOD = 33
TCON = 208

 bend_sensor_calibrate()

 // waits for the back bumper to be hit
 start

 //     print("starting\n")
 do_bump()

 if (button != 5) {
   goto start
 }

go

 bump_sens_avoid()

 bend_sens_avoid()

 bend_sensor_calibrate()

//     goto start

rand_turn()

 move_fw()

 bump_sens_avoid()


 // search for a target

 search
 //     print("searching\n")
 do_ir()

 bump_sens_avoid()
 bend_sens_avoid()
 bend_sensor_calibrate()

 if (ir_result <= IRTHRESH) {
   goto search
 }



// if ir_result > IRTHRESH, then we should stop
stop()

 // only try to line up 5 times
 lineup_count = 0


 // line up our target
 line_up
 print("line_up\n")

 bump_sens_avoid()
 bend_sens_avoid()


 lineup_count++

 // if we have tried to line up more than 5 times, back up
 // and start over
 if (lineup_count > 5) {
   move_bw()
```

```
    for(i 0, 0xFFF) {
      for(j 0, 3) {}
    }
    stop()
    goto go
  }

// make sure object is still there
  do_ir()

  if (ir_result < IRTHRESH) {
    goto go }

  rdist = 0
  ldist = 0

  EA = 1        // enable all interrupts
  EX0 = 1      //  enable external interrupt 0
  EX1 = 1      //  enable external interrupt 1

  rhit = 0
    lhit = 0

  for (z 0, 13) {
    distance = z

    if (rhit == 0) {
      arm = RIGHT
      move_arm()
    }
    if (lhit == 0) {
      arm = LEFT
      move_arm()
    }
    if (lhit == 0){
      for (j 0, 0x8FFF) {}
    }
    if (rhit == 0) {
      for (j 0, 0x8FFF) {}
    }
  }


distance = 0
  arm = RIGHT
  move_arm()
  arm = LEFT
  move_arm()
  for (j 0, 0xFFFF) {}

if (lhit != 1) {
  if (rhit != 1) {
    do_ir()
      if (ir_result <= IRTHRESH) {
        move_bw()
         for (i 0, 0xFFFF) {
           for (j 0, 1) {}
         }
        stop()
        goto go
         }
    if (ir_result >= IR_TOO_CLOSE) {
      move_bw()
        for (i 0, 0xFFFF) {}
      stop()
        goto go
        }
    move_fw()
    for(i 0, 0x1FFF) { }
    stop()
     goto line_up
```

```
   }
  if (lhit == 0) {
    turn_r()
    for(i 0, 0x1FFF) { }
    stop()
    goto line_up
  }
}
if (lhit == 1 ){
  if (rhit != 1) {
    turn_l()
      for(i 0, 0x1FFF) {}
    stop()
      goto line_up
      }
}
if (rhit == 1) {
  if (lhit == 1) {
    goto punch }
}

goto line_up

//      goto line_up

// we've hit with both sensors, so now we go on to punching

contact = 0

punch
  print("punching\n")

  for (i 0, 0xFFFF) {}

  EX0 = 1
  EX1 = 1

  lhit = 0
  rhit = 0
  contact = 0

    for (combo_cnt 0, 5) {

      EX0 = 1
      EX1 = 1

      // one of the glove must contact every combo
      contact = 0

      do_combo()

      // if no glove has hit, go to line_up which will
      // check if the target is still there, if so, it will
      // adjust the robot to hit it
      if (rhit == 0) {
        if (lhit == 0) {
         if (contact == 0) {
           goto line_up
         }}}

      // re-enable the interrupts which may have been disabled
      // when the gloves hit
      EX0 = 1
      EX1 = 1

      if (rhit == 0) {
        if (lhit == 0) {
         if (contact == 1) {
           goto celebrate
         }
        }
```

```
            }

            // if we made contact with at least one glove, then
            // we will punch again, until we've tried 4 times
            if (combo_cnt == 4) {
              goto pout
            }

          }

        celebrate
        print("celebrate\n")
        move_bw()
        for (i 0, 0x1FFF) {}
        spin_r()
        for (i 0, 0xFFFF) {
          for (j 0, 5) {}
        }
        spin_l()
          for (i 0, 0xFFFF) {
            for (j 0, 2) {}
          }
        stop()
          goto go

          pout
          for (s 0, 5) {
            move_bw()
              for (i 0, 0x8FFF) {}
            spin_r()
              for (i 0, 0x4FFF) {}
            spin_l()
              for (i 0, 0x4FFF){}
            stop()
              }
              goto go


          }

bend_sensor_calibrate() {

// calibrates bend sensors--assumes sensors at rest
    line = LBEND
    do_bend()
     lbend_rest = bend_result
    line = RBEND
    do_bend()
     rbend_rest = bend_result

}

extint0() {

  print("foo\n")
  // right glove ext int.
  EX0 = 0
  rtarget_dist = z
    rhit = 1
    contact = 1
    line = RBEND
    do_bend()
    rtarget_bend = bend_result

  reti()

    }

extint1() {
  // left glove ext int.
  print("bar\n")
  EX1 = 0
    ltarget_dist = z
    lhit = 1
    contact = 1

    line = LBEND
    do_bend()
    ltarget_bend = bend_result

  reti()

    }

do_combo() {

  EX0 = 1
    EX1 = 1

  rhit = 0
  lhit = 0

  rand = TL0
  rand = rand / 0x40

    // rand = 3

    print("doing combo #rand\n")

    if (rand == 0) {
      for (s 0, 3) {
        distance = ltarget_dist
        //      print("ldist = #ltarget_dist\n")
        left_punch()
        for(j 0, 0xFFFF) {}
        distance = rtarget_dist
        right_punch()
        for(j 0, 0xFFFF) {}
        lpunch_end()
        for (j 0, 0xFFFF) {}
        rpunch_end()

      }
    }


  if (rand == 1) {
    for (s 0, 3) {
      distance = ltarget_dist
        left_punch()
        for(j 0, 0x8FFF) {}
        lpunch_end()
        for(j 0, 0xFFFF) {}
        distance = ltarget_dist
        left_punch()
         for(j 0, 0x2FFF) {}
      distance = rtarget_dist
        right_punch()
        lpunch_end()
        for(j 0, 0xFFFF) {}
        rpunch_end()
        left_punch()
        for (j 0, 0xFFFF) {}
        }
  }

  if (rand == 2) {
    for (s 0,3 ) {
      distance = rtarget_dist
        right_punch()
        for(j 0, 0x2FFF) {}
      distance = ltarget_dist
```

```
       left_punch()
       for(j 0, 0x8FFF) {}
       lpunch_end()
       for(j 0, 0xFFFF) {}
      distance = ltarget_dist
       left_punch()
       for(j 0, 0xFFFF) {}
       lpunch_end()
       rpunch_end()
         for( j 0, 0xFFFF) {}
         }
  }

  if (rand == 3) {
    for(s 0, 3) {
      distance = ltarget_dist
       left_punch()
       for(j 0, 0x8FFF) {}
      lpunch_end()
       for(j 0, 0xFFFF) {}
    }

    distance = rtarget_dist
      right_punch()
      for(j 0, 0xFFFF) {}
    rpunch_end()
      for(j 0, 0xFFFF) {}
  }


  // if this is our fourth hit, try the super-duper finishing move
  if (combo_cnt == 4) {
    move_fw()
      for (j 0, 0x1FFF) {}
    stop()
      for (j 0, 0xFFF) {}

    distance = rtarget_dist
      right_punch()
      for(j 0, 0xFFFF) {}
    // move back so we don't get too close to target
    rpunch_end()
    move_bw()
      for(j 0, 0x1FFF) {}
    stop()
    for (j 0, 0x8FFF) {}
  }
}

left_punch() {

  distance = ltarget_dist + 1
    arm = LEFT
    move_arm()
    line = LBEND
    do_bend()
    i = ltarget_bend - 50
    j = ltarget_bend + 50

    }

lpunch_end() {

  distance = 0
    arm = LEFT
    move_arm()
    }

right_punch() {

  distance = rtarget_dist + 1
```

```
    arm = RIGHT
    move_arm()
    line = RBEND
    do_bend()
    i = rtarget_bend - 50
    j = rtarget_bend + 50

    }

rpunch_end() {

  distance = 0
    arm = RIGHT
    move_arm()
    }


bump_sens_avoid() {

  line = BUMP

    do_bump()

  if (button == 0) {
  ret()
    }
  if (button == 5) {
    move_fw()
      for (i 0, 0xFFF) {
       for (j 0, 1) {}
      }
    stop()
      ret()
      }

    move_bw()
      for (i 0, 0xFFF) {
       for (j 0, 1) {}
      }
    stop()

}

rand_turn() {

    // gets a pseudo random # from counter
    rand = TH0
    rand = rand * 0xF0
    rand = rand + TL0

    rand_dir = rand / 0x8000

    if (rand_dir == 0) {
      turn_l()
    }
    if (rand_dir == 1) {
      turn_r()
    }

    // wait for rand clock cycles
    for (i 0, rand) {
      line = IR
       do_ir()
       if (ir_result >= IR_LONG_THRESH) {
         stop()
         ret() }
    }
    // wait at least this long
    //    for (i 0, 0xFFFF) {}

    stop()
```

```
}

bend_sens_avoid() {


  line = LBEND
    do_bend()
    i = lbend_rest - 10
    j = lbend_rest + 10
    if (bend_result < i) {
      if (bend_result > j) {
        move_bw()
        for (i 0, 0xFFF) {
          for (j 0, 1) {} }
        stop()
        ret()
      }

      line = RBEND
      do_bend()
      i = rbend_rest - 10
      j = rbend_rest + 10
      if (bend_result < i) {
        if(bend_result > j) {
          move_bw()
          for (i 0, 0xFFFF){
            for (j 0, 1) {}}
          stop()
        }

      }

    }
}

do_ir() {


  line = IR
  do_ad()
    ir_result = ad_result

}

do_bend() {

  do_ad()
    bend_result = ad_result

}

do_ad() {

        ADCON = 0x00
        line = line + 8
        ADCON = line

        repeat
          //      s = 0x80
          // s = s & ADCON
          //if (s == 0) {
          // goto repeat }
          for(count 1,100) {}
        ad_result = ADCH * 4
        i = ADCON / 64
        ad_result = ad_result + i
}

do_bump() {
  line = BUMP
  do_ad()

    hi = ad_result + 10
    lo = 0

    if (ad_result >= 10 ) {
      lo = ad_result - 10
    }

  button = 0

    if ( ad_result < 20 ) { button = 0 }

  if ( hi >= 729) { if (lo <= 729) { button = 1 }}

  if (hi >= 582) { if (lo <= 582) {button = 2}}

  if (hi >= 488) { if (lo <= 488) {button = 3}}

  if (hi >= 291) { if (lo <= 291) {button = 4 }}

  if (hi >= 131) { if (lo <= 131) {button = 5}}

  if (hi >= 912) { if (lo <= 912) {button = 6}}
  if (hi >= 786) { if (lo <= 786) {button = 7}}
  if (hi >= 628) { if (lo <= 628) {button = 8}}
  if (hi >= 878) { if (lo <= 878) {button = 9}}
  if (hi >= 699) { if (lo <= 699) {button = 10}}

}


move_fw() {

        arm_wheel = WHEEL
        dir = FORWARD
        servo()
}

move_bw() {
        arm_wheel = WHEEL
        dir = BACK
        servo()
}

turn_l() {
        arm_wheel = WHEEL
        dir = LEFT
        servo()
}

turn_r() {
        arm_wheel = WHEEL
        dir = RIGHT
        servo()
}

spin() {

  turn_l()
    turn_r()
  //  arm_wheel = WHEEL
  //dir = RIGHT
  //servo()
  //dir = LEFT
  //servo()
    }

stop() {
        arm_wheel = WHEEL
        dir = STOP
        servo()
        }

arm_rext() {

        arm_wheel = ARM
        arm = RIGHT
        servo()
}

arm_lext() {

        arm_wheel = ARM
        arm = LEFT
        servo()
}

servo() {
        if (arm_wheel == WHEEL) {
                move_wheel()
                 for(i 0, 0x1FFF) {}

        }
        if (arm_wheel == ARM) {
                move_arm()
                 for(i 0, 0x1FFFF) {}
        }
}

move_wheel() {

       if (dir == FORWARD) {
               r_f() l_f() }
       if (dir == BACK) {
               r_b() l_b() }
       if (dir == RIGHT) {
               r_s() l_f() }
       if (dir == LEFT) {
               r_f() l_s() }
       if (dir == STOP) {
               r_s() l_s() }

}

r_f() {
P4.0 = 1
P4.1 = 0
PWMP = 200
PWM0 = RF
}


r_s() {
P4.0 = 1
P4.1 = 0
PWMP = 200
PWM0 = 0
}

r_b() {
P4.0 = 1
P4.1 = 0
PWMP = 200
PWM0 = RB
}

l_f() {
P4.0 = 1
P4.1 = 0
```

```
PWMP = 200
PWM1 = LF
}


l_s() {
P4.0 = 1
P4.1 = 0
PWMP = 200
PWM1 = 0
}

l_b() {
P4.0 = 1
P4.1 = 0
PWMP = 200
PWM1 = LB
}

spin_r() {
  P4.0 = 1
    P4.1 = 0
    PWMP = 200
    PWM0 = RF
    PWM1 = LB
    }

spin_l() {
  P4.0 = 1
    P4.1 = 0
    PWMP = 200
    PWM0 = RB
    PWM1 = LF
    }


move_arm() {

        P4.0 = 0
        P4.1 = 1

        if (arm == RIGHT) {
                diff = REXT - RRET
        }
         if (arm == LEFT) {
          diff = LRET - LEXT
            }

        //      diff = extnd - retrct


         final = diff / 10
         final = final * distance
          //             print("final = #final\n")

          if (arm == RIGHT) {
            final = final + RRET
            //      print("right\n")
          }
        if (arm == LEFT) {
          final = LRET - final
            //      print("left\n")
            }

        //      print("final = #final\n")

          if (distance == 0) {
            final = 0}

        PWMP = 100
        if (arm == RIGHT) {
                PWM0 = final }
        if (arm == LEFT) {
                PWM1 = final }

}
```

APPENDIX B

Price List:

Item	Vendor	Price each
4 Fubata Standard Sports Servos	Servo City	4 x $10.95
1 P500 Microcontroller Board (11.05 Mhz)	Tecel Electronics	$70.00
2 Flex Sensors	Jameco	2 x $10.15
1 IR Detector	Jameco	$13.35
1 Punching Nun Fingerpuppet	Spencer Gifts	$9.95
2 Microswitches	RadioShack	2 x $2.95
Total:		$163.30