April 23, 2002
Derrick Moy
TA: Aamir Qaiyumi
Instructor: A. A Arroyo

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

# serverBOT

# Table of Contents

## Abstract

The serverBOT is a basic human interaction autonomous agent. It is modeled after the idea of the robotic receptionist. The role of a robotic receptionist is the basic step towards human interaction. Currently robots are made to do specific tasks autonomously but these tasks are usually done in a controlled environment. Though I believe the next step towards the evolution of robotics is the ability to interface a robot with a human. Though serverBOT is as mundane as it gets when it comes to human interaction it is the first step for me and hopefully I will be able to learn as much as I can so that I can take serverBOT to the next level towards the ideal robotic receptionist/companion

## Executive Summary

The serverBOT is an autonomous agent designed to serve food to people. In its current state it utilizes Sharp GP2D12 IR Range Finders, Eltec 442-3 Pyroeletric Sensors, a CDS cell, and an Interlink Force Sensitive Resistor to aid it in its task.

The brains of the serverBOT is the ATMEL AT90S8535 microcontroller. This board was wire-wrapped because I did not have the chance to get the circuit printed. Details on this microcontroller is located below, in the microprocessor section

## Introduction

The serverBOT is a robotic butler that serves drinks or food to people. Once its tray is empty it returns to the kitchen (home). The serverBOT has a serving tray in which food or items are placed on top. When activated the serverBOT searches for people and stops in front of them so that the items can be removed from the tray. After a momentary pause, the serverBOT randomly turns in any direction and begins to look for more targets and moves towards them. While the serverBOT is performing this serving routine, it constantly checks its tray to see if it is empty. Once the tray is empty the serverBOT returns home for a tray refill.

The main focus in this task is the accurate and reliable detection of a human. To accomplish this, a pyroelectric sensor will be used to detect a human and provide a directional vector so the serverBOT knows where to go. The complexity of this problem lies in the sensitivity of the pyroeletric sensor and the algorithm the drivs the detection of this sensor. Details about this problem will be addressed in the unique sensor section of this report.

The remaining part of this document will focus on the design of serverBOT in its entirety.

## Microprocessor

For the serverBOT, I decided to build my own microprocessor controller and use the ATMEL AVR microprocessor. The Atmel 8-bit AVR is a family of general purpose RISC-based microcontrollers that come in a wide variety of configurations to suit your application needs. These configurations vary in the size and availability of the following features:

- Flash Memory
- EEPROM
- RAM
- 10-bit Analog to Digital Converter
- Programmable UART
- Master/Slave SPI Serial Interface
- 8/16-bit Timer/Counters with Input Capture and Output Compare
- 8/9/10-bit PWM
- External Interrupts
- Programmable Watchdog Timer with On-chip oscillator
- Analog Comparator
- Package Size
- Low Operating Voltages
- Speed Grades

All Atmel AVR microcontrollers feature a universal set of instructions to simplify upgrade paths with little modification if needed. Most of these instructions execute in a single clock cycle. Executi ng at up to 1 cycle per instruction @ 8 Mhz, it is possible to achieve 8 MIPS.

With a wide range of features and speed, the Atmel AVR was designed for self-contained applications without the need of expansion.

For the serverBOT, I initially designed a board to be printed, but I was trying to get it milled and wasn't able to generate the proper files for the t-tech. I should've tried to send it off to a printing house and gotten my board etched. Since I had to stop trying to get my circuit board printed, I had to wire-wrap my entire board. I cannot guarantee the effectiveness of my schematic because I did not have the change to verify it. It was doine in eaglecad. I do believe I accidentally switched the filtering capacitors that are routed to the LM2940 voltage regulator.

Main Components:

8Mhz External Oscillator

ATMEL AT90S8535

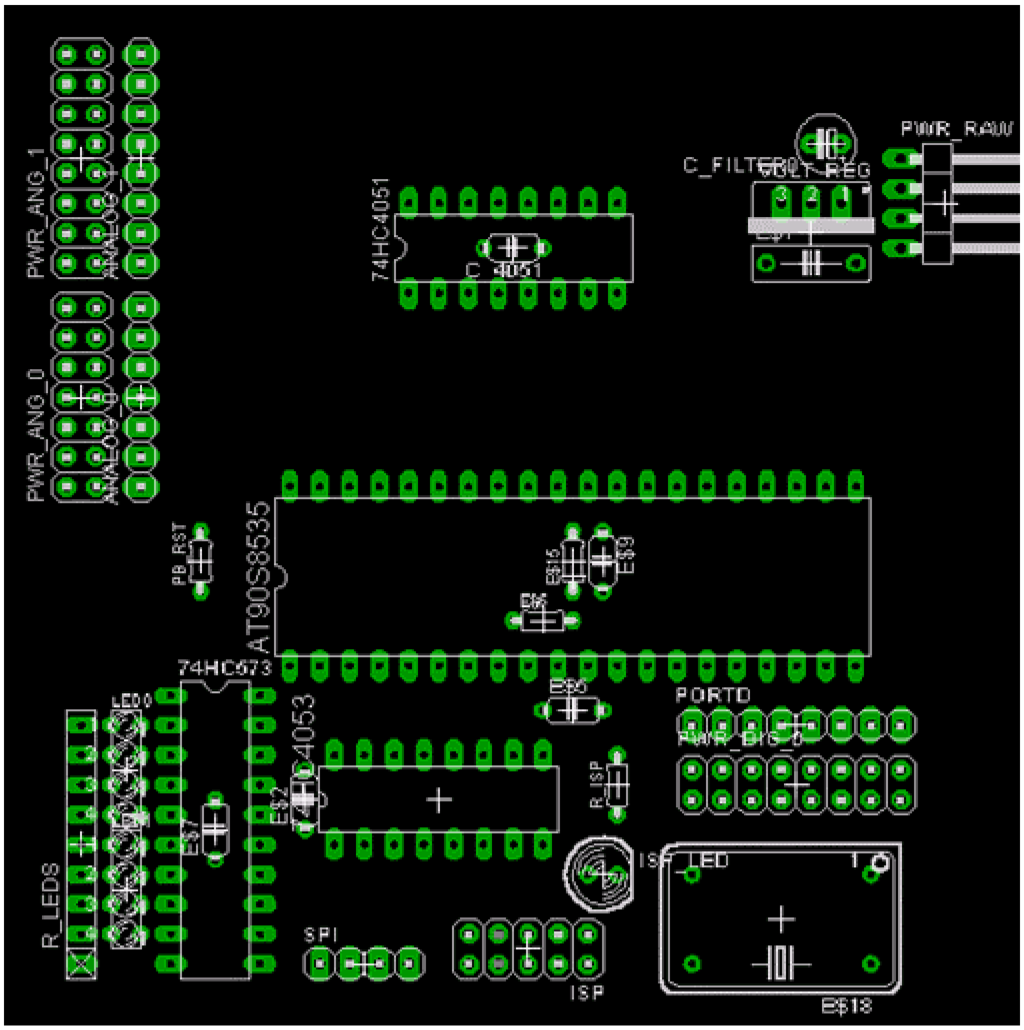
74’573

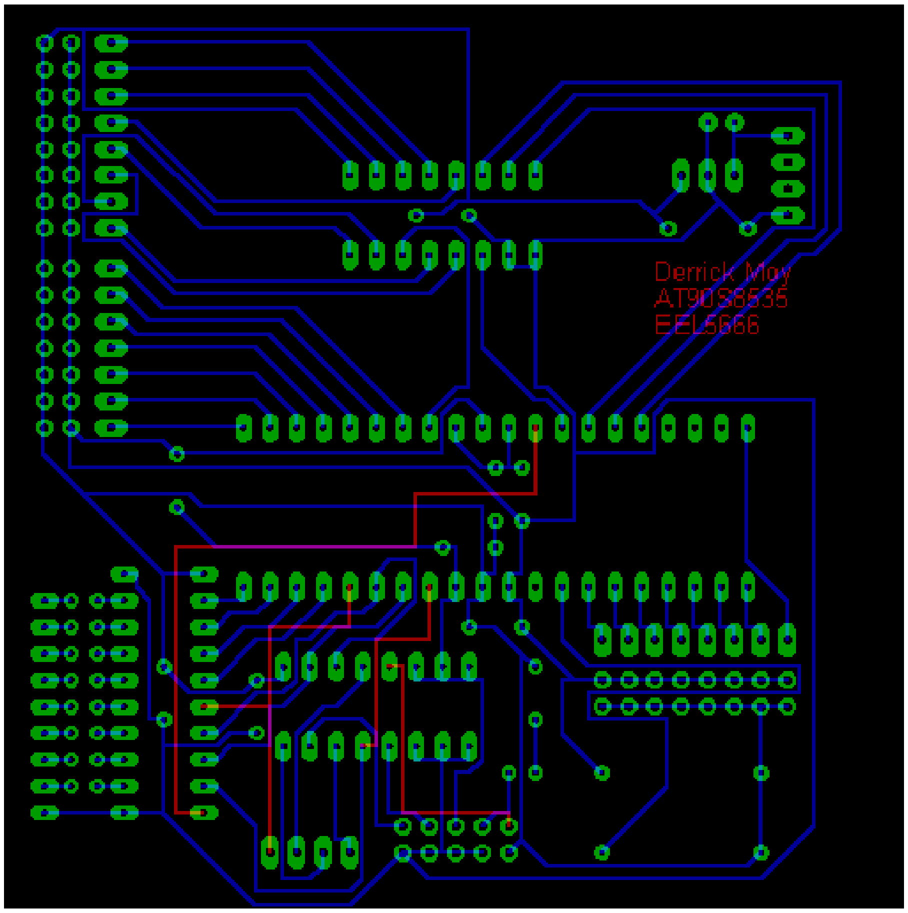
74’4051

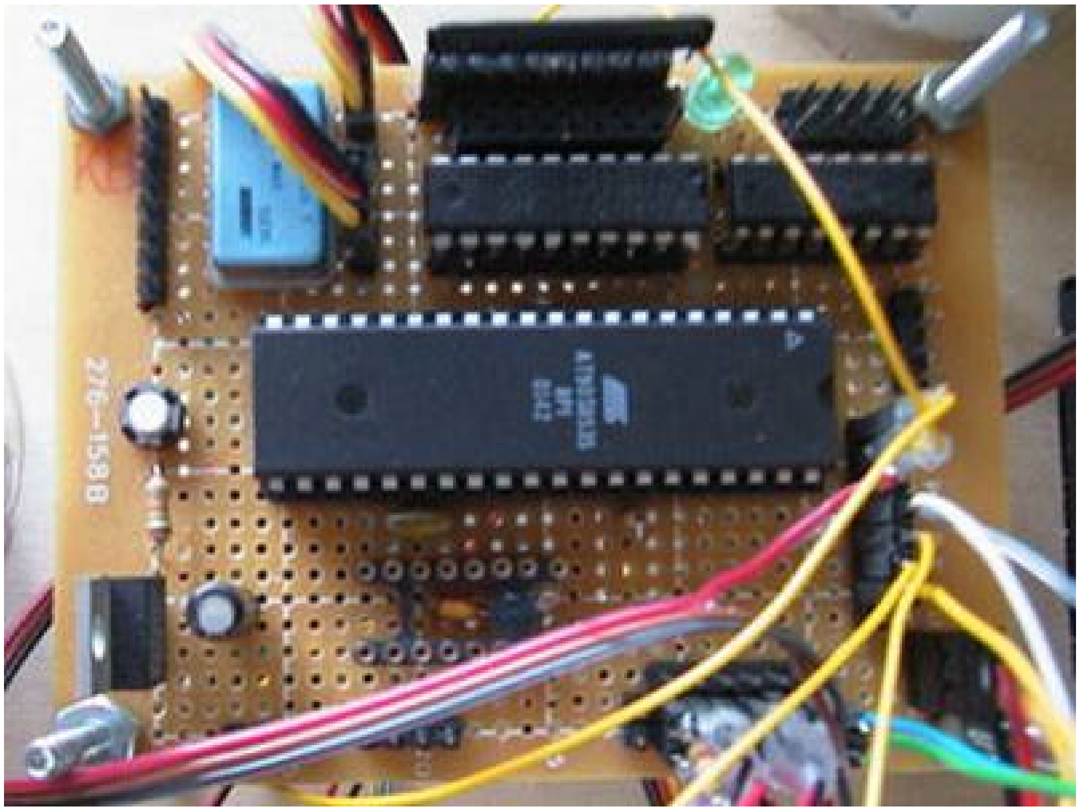
74’4053

LEDS

LM2940 – Voltage Regulator

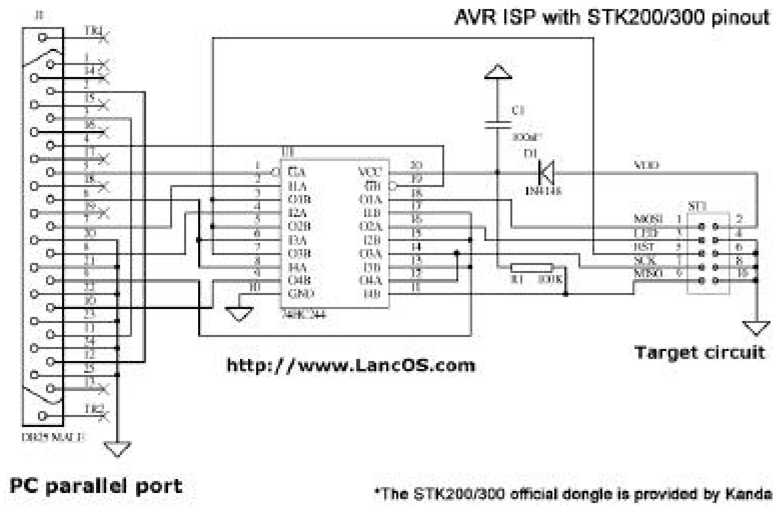


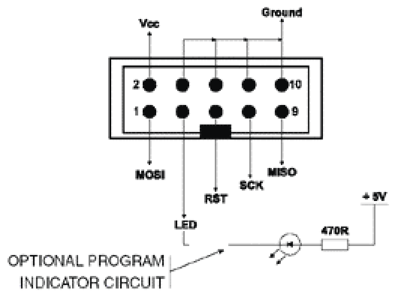

Derrick Moy
AT90S8535
EEL5666

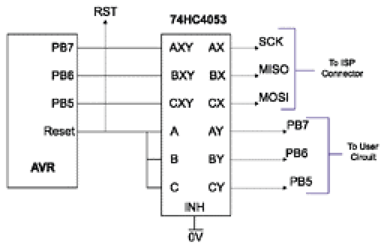


## Interfacing

The ATMEL AVR utilizes the SPI Serial Interface to allow it to be programmed while in-circuit. This type of programming requires an ISP compatible dongle which is easy to build or can be purchased from Kanda Systems

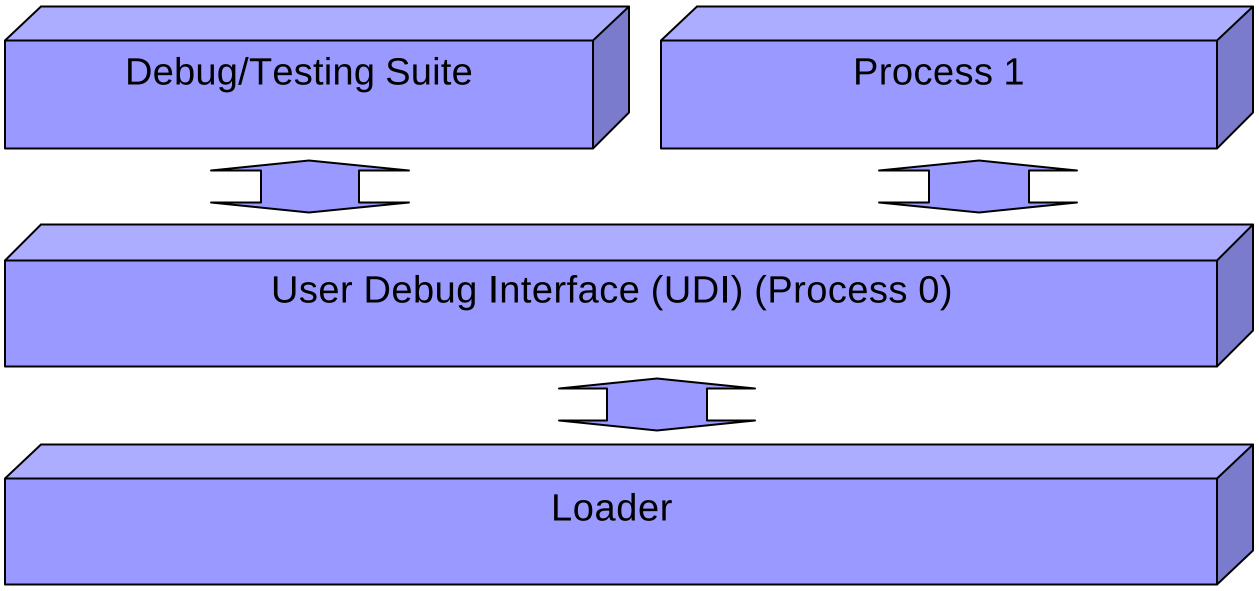






## Software

For the ATMEL microcontroller, I designed a loader that allows the ATMEL microcontroller to run two processes at once. Each process is time multiplexed by CLK/1024. On top of the loader I've written a User Debug Interface that allows me to output information out to the screen while the serverBOT is running in real time. The User Debug Interface also can be set to control some features of the serverBOT to test with. Process 1 is the main program of the serverBOT and it is loaded on initialization. From the User Debug Interface, the serverBOT process 1 can be killed at any time.



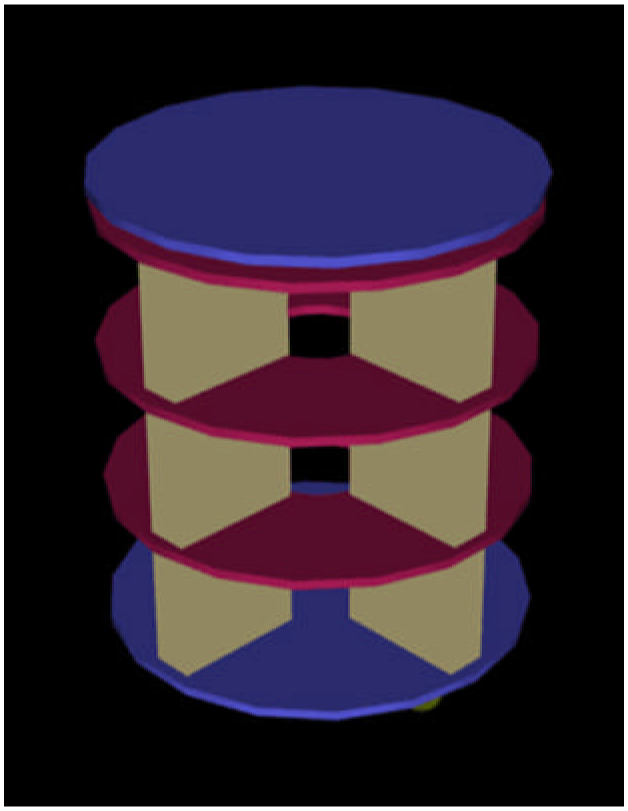
Problems that arose while designing the multi-tasking handler for the serverBOT is the prioritizing of interrupts and trying to isolate interrupt events so that they do not interfere with other processes.

## Mobile Platform

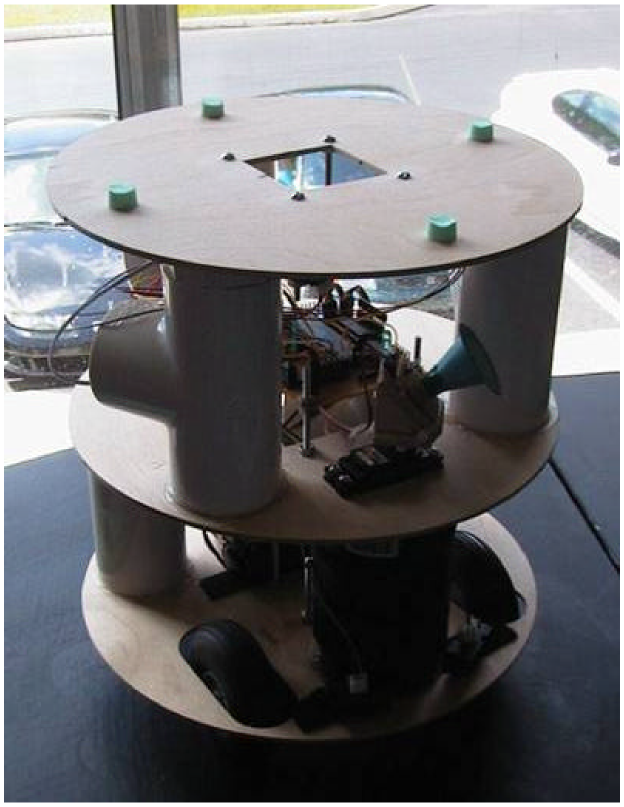
The functionality of the serverBOT requires it to be large enough for human interaction but, due to the size of the robot and the limited resources the initial build of the serverBOT will be 1/8th scale. The basic body of the serverBOT will be cylindrical and it will run on 2 motorized wheels and a pivoting wheel. The motors will be driven by standard 43 oz-in hacked servos. The tray will be placed directly on top of the serverBOT. Underneath the serving tray will be the force sensor that checks the tray to see if its empty. Inside the cylindrical body will be 2 orthogonal base boards which will hold the microcontroller board and sensors.

*Drawing files are attached in the disk and made by autocad 2000

### Conceptualization



### Realization



## Actuation

There are three main types of actuation on the serverBOT. The first type of actuation sweeps the motion detector around in a 120 degree arc. The second type of actuation controls the motors that drive the serverBOT around. The third type of actuation uses a dental silicone cylinder mold to provide a pressure transfer from the tray to the force sensitive resistor.

### *Type I – Motion Detector Sweep*

Components – 1 Hitec Servo at ~ $12.75 each from http://www.servocity.com

Servos are digitally controlled motors that have a turning arc of approximately 180 degrees. They are controlled by pulse width modulation and its position is dependent on the duty cycle and period of the input signal. For the ATMEL AVR microcontrollers the neutral position of the servos is located at 0x18 for an 8-bit PWM signal with clock input of 8Mhz / 256 = 31.25Khz. The PWM located on the ATMEL starts count from 0x00 to 0xFF and toggles the PWM output when the timer matches the PWM signal.

*Sample Code:*

```
;*******************************************************************************************************************
TIM2_PWM:       sbi DDRD, 7
                ldi TEMP, $66               ; PWM, NON-INVERTING, CLK=CLK/256
                out TCCR2, TEMP

                ;ldi TEMP, $18              ; Set OCR2 to ($18) Center Position
                ;out OCR2, TEMP

                ret                         ; Return From Subroutine

;*******************************************************************************************************************
                inc MOT_LOC                         ; inc MOT_LOC
SWEEP_LP1:      out OCR2, MOT_LOC                   ; Incremental Clockwise Servo Swing

;*********************************************************************************************************
```

### *Type II – serverBOT Motor Controls*

Components – 2 Hitec Servos at ~ $12.75 each from http://www.servocity.com

Type II actuation uses a modification of the type I actuation. In the type II actuation, 2 servos are hacked to drive the serverBOT. In a hacked servo, the tab stops in the servo are removed to allow the servo to swing in a 360 degree arc. The potentiometer in the servo is also disabled and set to the neutral position at all times. This provides a simple speed control function to the servo. Setting the duty cycle close to the neutral position will cause the servo to move slowly on the contrary setting the duty cycle to a farther position will cause the servo to move faster. In the standard operation of the servo, if the targeted position is far away from the current position the servo will move faster and when it is closer it will move more slower. This is to provide some sort of accuracy so that the target position is reached quickly but not overshot.

For more information on hacking servos please refer to the following website http://www.rdrop.com/~marvin/explore/servhack.htm

*Sample Code*

```
;*****************************************************************************************
Init_Motors:        sbi DDRD, OC1A              ; portD pin 5(OC1A) = output
                    sbi DDRD, OC1B              ; portD pin 4(OC1B) = output

                    ldi TEMP, $A1               ; Setups non-inverting 8-bit pwm on
                    out TCCR1A, TEMP            ; channels A  & B

                    ldi TEMP, $04               ; Timer1CLK = CLK/256
                    out TCCR1B, TEMP            ;

                    ret                         ; Return from Subroutine

;*****************************************************************************************
Test_Motors:        ldi TEMP2, $00              ;
                    out OCR1AH, TEMP2
                    out OCR1BH, TEMP2

                    ldi TEMP2, $20              ; Counter Clockwise on A
                    out OCR1AL, TEMP2           ;

                    ldi TEMP2, $16              ; Clockwise on B
                    out OCR1BL, TEMP2           ;

                    ret

;*****************************************************************************************
```

## *Type III – Cylindrical Pressure Transfer Mold*

Components – ½Copper Coupling Tube $1.00 from Lowes, 2 Part Dental Silicone Molding Paste from the University of Florida Dental School.

The cylindrical pressure transfer mold is a piece of dental silicone molded from a ½ copper coupling tube. Since the height of the force sensitive resistor platform is adjustable, it allowed me to use the coupling tube without any worry about it being too short. The cylindrical pressure transfer mold is used as an transfer medium to allow the pressure or weight from the tray and objects on the tray to be transferred into the force sensitive resistor so a force to resistance value could be obtained.

## Sensors

The serverBOT contains 4 basic sensors. Three of the sensor applications are used to help navigate the serverBOT, while the fourth sensor is used to detect if the serving tray is empty. ServerBOT's sensor suite consists of the following sensor technologies:

1. IR Range Detection
2. Human Detection
3. Force Sensing Resistors
4. Beacon Detection
    a. IR Beacon*
    b. CDS Cell

### IR Range Detection

Components – Sharp GP2D12, $10.95 obtained from Mekatronix.

The sharp GP2D12 IR Range Finders provide an analog signal that is inversely proportional to its detection range. It provides a signal from 0-5 VDC depending on how close the object is located. The effective distance ranging the GP2D12s provide is from 10-80 cm. The datasheets for the GP2D12s are provided in the appendix. The GP2D12s are fairly easy to integrate. The only suggestions I have for this is to make sure there is no other interference from other IR sources. Another thing to note is that the GP2D12 provides an linear analog progression until the objects get too close. Once the objects are too closer than its detection distance, its output fluctuates.

*Sample Code*

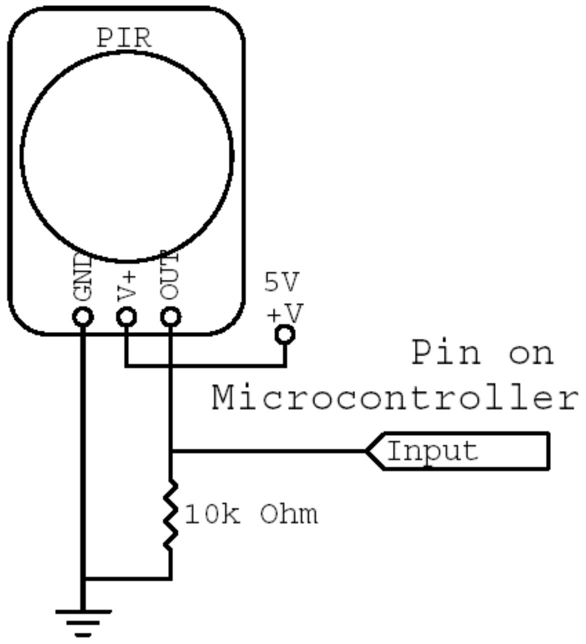
```
ldi GENIOI, $01       ; select ADMUX to $01 for IR Sensor
rcall ADC_8bit        ; call ADC_8bit conversion
cpi GENIOI, $60       ; compare IR detectors to $60
brsh Right_Turn       ; If detectors > $60 turn right
```

## Motion Sensing

A large amount of time was spent, trying to provide an accurate and reliable way to detect humans. 3 Attempts were made until I finally found a working algorithm and sensor to provide such an accurate and reliable detection. The first and second attempts used a different pyroeletric sensor but used the same detection algorithm with a slight variation. The third attempt used the same sensor as the second attempt but used a different detection algorithm.

| Attempt # | Sensor Used | Algorithm Used |
|---|---|---|
| 1 | 1 | 1 & 1b |
| 2 | 2 | 1c |
| 3 | 2 | 2 |

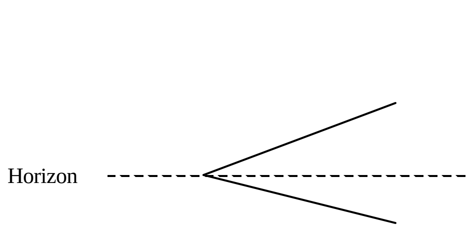
### Sensor 1 – HVW Technologies PIR Sensor



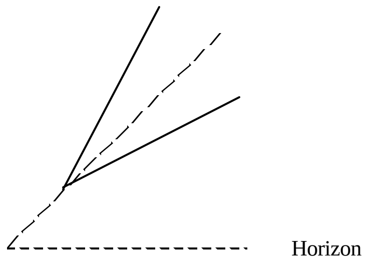
- Passive Infrared Sensor
- Range: 60º Omnidirectional
- Distance: 5m
- Pulses high on signal when motion is detected for approximately 1 second
- $10.50 from hvwtech
- http://www.hvwt ech.com/dnload/PIRManual10.pdf

The pir sensor will be mounted 8" above the ground on top of a servo on the serverBOT.

To provide a higher angle of detection along the horizon, the pir sensor will be mounted at a 45 degree angle. This elevated angle will set the elevated detection angle from 0-30 degrees to 30-60 degrees from the horizon.

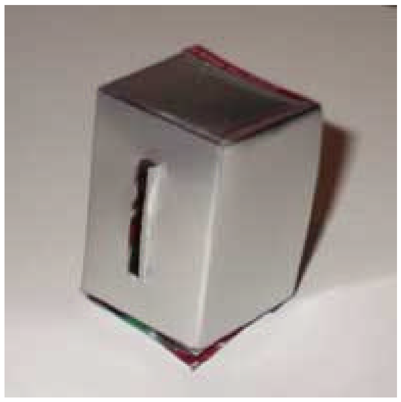


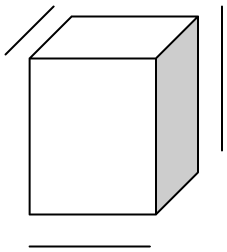
60 degrees



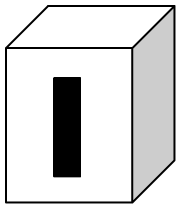
45 degree offset

A cut out aluminum can will be used to limit the active area of the pir sensor to approximately 15 degrees. The device is shaped like a box and fits right over the pir sensor assembly. In the center of this device is a rectangular cutout to limit the azimuth of detection for the pir sensor.

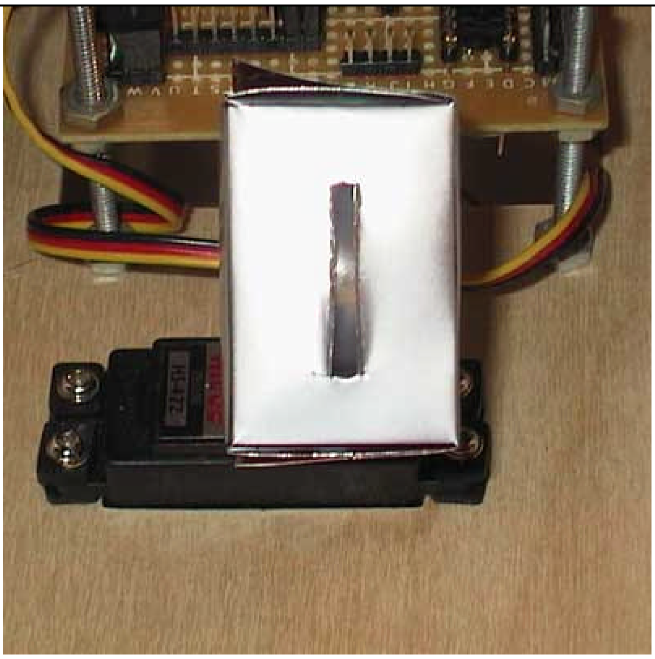
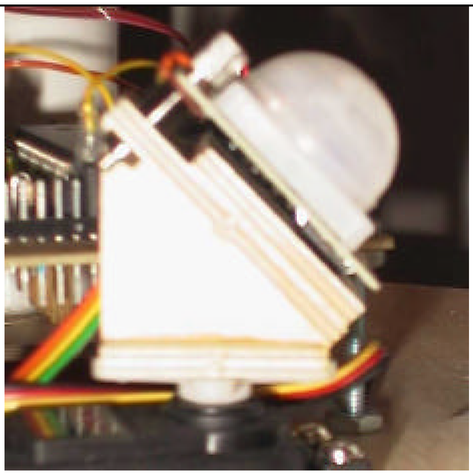




Aluminum shield dimentions:
LxWxH = 1.5"x1"x1"

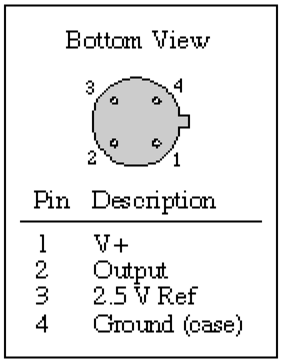


Aluminum shield with cutout hole to limit the azimuth to < 15 degrees.

| | |
|---|---|
|  |  |

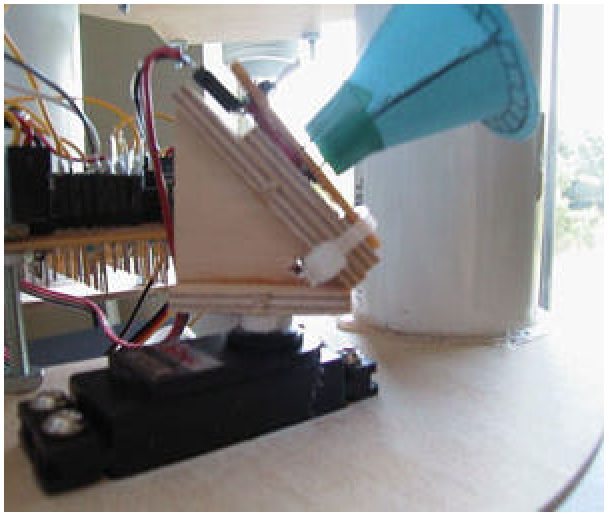
## Sensor 2: Eltec 443-2 Pyroelectric Detector

- Analog Output
- More immune to IR noise
- Better Fresnel Lens
- Analog output gives more flexibility and allows the user to set the thresholds of detection properly
- Floats around 2.5V when there is no detection and voltage moves up or down depending on the direction of motion across the dual elements
- http://www.acroname.com/robotics/parts/R1-442-3.html



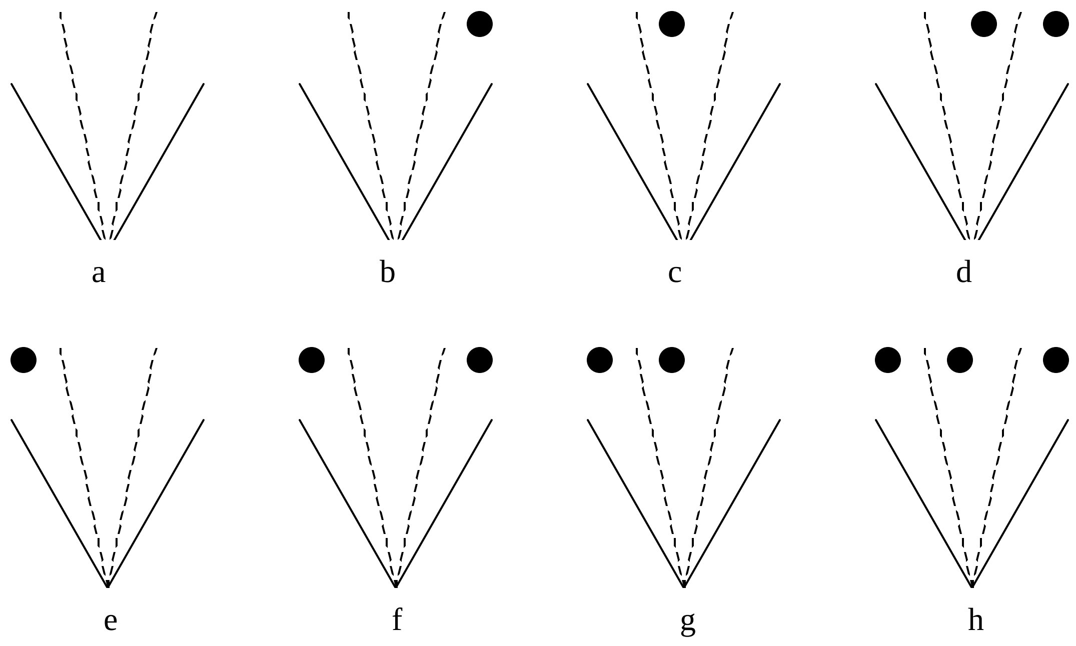
| Pin | Description |
|---|---|
| 1 | V+ |
| 2 | Output |
| 3 | 2.5 V Ref |
| 4 | Ground (case) |

The eltec 443-2 was mounted the wedge the same way the first sensor was mounted except the fresnel lens provided with the eltec 443-2 was cone shaped.



### Sensing Algorithm 1

The basic theory of human detection with a direction vector is to slowly sweeping while pausing in between each increment to see if a human was sensed at that directional vector. The design I created is a slight variation of this basic theory. In my design I will sweep the pir sensor in a constant velocity twice, once in the clockwise direction and once in the counter-clockwise direction. While generating a sweep the uController will record the first detected value in each direction. By using this method in combination with fuzzy logic, the uController can determine which direction the human will be. This also creates a basic algorithm in which there are multiple humans and determines which direction the serverBOT should head. This will provide 8 possible scenarios. In the following chart, the 8 possible scenarios are depicted with the black dot representing a person or a cluster of people.



Using these 8 possible scenarios and the concept of fuzzy logic, the uController will determine which direction to head towards. The range of detection for the pir sensor is from $0E to $21 providing the following boundaries. Using the values on the left of the table below to produce values for the clockwise and counter-clockwise detection. It is placed into the table on the right to provide a fuzzy logic response to determine the direction to head towards.

Left – $0E to $12
Middle – $15 to $1B
Right - $1C to $21

| CW:CCW | Left | Middle | Right |
|---|---|---|---|
| Left | L | LM | M |
| Middle | LM | M | RM |
| Right | M | RM | R |

*Sample Code*

```
;***************************************************************************************
;* Subroutine - Motion_Sweep
;* Description: This subroutine is used to sweep the motion sensor CW then CCW
;*                  as the motion detector is being swept, only the first value obtained
;*                  in each direction of motion is recorded resulting in two values
;*                  one stored in DET_CW, and DET_CCW
;* Uses: TEMP
;* Output: DET_CW, DET_CCW
;***************************************************************************************
Motion_SWEEP:   sbr STATUS, $01         ; Setup STATUS to show CW Sweep
                clr DET_CW              ;
                clr DET_CCW             ;
                rcall TIM2_PWM          ; Set Timer2 to PWM Mode and PortD OC2 to output
                ldi TEMP, $0E           ; Set Servo to Left Most Position
                mov MOT_LOC, TEMP
                out OCR2, MOT_LOC       ;

                rcall Motion_Delay   ; Wait until SERVO SWINGS to leftmost position
                rcall INT0_EN           ; Enable External Interrupt 0

                inc MOT_LOC             ;
SWEEP_LP1:      out OCR2, MOT_LOC       ; Incremental Clockwise Servo Swing

                rcall Motion_Delay
                inc MOT_LOC             ;
                mov TEMP, MOT_LOC
                cpi TEMP, $22           ; Check for Rightmost Position
                brne SWEEP_LP1

                cbr STATUS, $01         ; Setup STATUS to show CCW Sweep
SWEEP_LP2:      out OCR2, MOT_LOC               ;

                rcall Motion_Delay

                dec MOT_LOC
                mov TEMP, MOT_LOC               ;
                cpi TEMP, $0E           ; Check for Leftmost position
                brne SWEEP_LP2

                rcall INT0_DIS          ; Disable External Interrupt 0

;               ldi TEMP, $18           ; Return to Neutral Position
;               out OCR2, TEMP          ;
;               rcall Motion_Delay   ; Wait until servo swings to neutral position
                ret
;***************************************************************************************
;* Subroutine - External Interrupt 0 Enable
;* Description: Enables the External Interrupt 0 and sets it to trigger off of the
;*                  rising edge
;***************************************************************************************
INT0_EN: push TEMP
                ldi TEMP, $03                   ; Set INT0 to trigger off of rising edge
                out MCUCR, TEMP

                ldi TEMP, $40                   ; Enable External Interrupt 0
                out GIMSK, TEMP
                pop TEMP
```

```
                  ret
;**************************************************************************************
;* Subroutine - External Interrupt 0 Disable
;* Description: Disables the External Interrupt 0
;**************************************************************************************
INT0_DIS:         ldi TEMP, $00                      ; Disable External Interrupt 0
                  out GIMSK, TEMP
                  ret
;**************************************************************************************
;* Interrupt - External Interrupt 0 Handler
;* Description: When an external interrupt occurs on INT0, a motion is detected by
;*                the sensor. The external Interrupt will check the current direction
;*                and the direction vector to make sure it is the f irst value for that
;*                direction and if so it records it else it exits
;* Uses: TEMP
;**************************************************************************************
EXT_INT0:         push TEMP                          ;
;                 mov GENIOI, TEMP                   ; Debug test, output all values
;                 rcall HEX2ASCII                    ; detected and exit interrupt
;                 pop TEMP                           ;
;                 reti                               ;

                  sbrc STATUS, 0
                  rjmp EXINT0_CW
EXTINT0_CCW:      tst DET_CCW                        ; Check to see if First Value of CCW
                  brne EXINT0_EX                     ; If not exit
                  in TEMP, OCR2
                  mov DET_CCW, TEMP                  ; Else record first value on CCW

EXINT0_EX:        pop TEMP                           ;
                  reti

EXINT0_CW:        tst DET_CW                         ; Check to see if First value of CW
                  brne EXINT0_EX                     ; If not exit
                  in TEMP, OCR2
                  mov DET_CW, TEMP                   ; Else record first value on CW

                  rjmp EXINT0_EX
;*
```

## Sensing Algorithm 1b

Algorithm 1b is a variation in algorithm 1 in the sense that when a hit was detected, the sweeping stopped and waited another 4 seconds in the same position of the hit. If another hit was detected than it would count as a human if it wasn't detected within the timeout period, then the servo would continue to sweep the motion sensor.

*Sample Code*

```
;**************************************************************************************
;* Interrupt - External Interrupt 0 Handler
;* Description: When an ex ternal interrupt occurs on INT0, a motion is detected by
;*                the sensor. The external Interrupt will check the current direction
;*                and the direction vector to make sure it is the first value for that
;*                direction and if so it records it else it exits
;* Uses: TEMP
;**************************************************************************************
EXT_INT0:         push TEMP                          ;
;                 mov GENIOI, TEMP                   ; Debug test, output all values
;                 rcall HEX2ASCII                    ; detected and exit interrupt
;                 pop TEMP                           ;
;                 reti                               ;
                  sbrc STATUS, 0                     ; If status bit0 = 0 then execute CCW
                  rjmp EXINT0_CW                     ; else execute CW
```

```
EXTINT0_CCW:    tst DET_CCW                      ; Check to see if First Value of CCW
                brne EXINT0_EX                   ; If not exit else record first value on CCW

                rcall MOT_ErrAv                  ; Call Motin Detection Error Avoidance
                tst GENIOI                       ; Check GENIOI
                brne EXINT0_EX                   ; If GENIOI != 0 then bad value and exit

                in TEMP, OCR2                    ; record first value on CCW
                mov DET_CCW, TEMP

EXINT0_EX:      pop TEMP                         ;
                reti

EXINT0_CW:      tst DET_CW                       ; Check to see if First value of CW
                brne EXINT0_EX                   ; If not exit else record first value on CW

                rcall MOT_ErrAv                  ; Call Motion Detection Error Avoidance
                tst GENIOI                       ; Check GENIOI
                brne EXINT0_EX                   ; If GENIOI != 0 then bad value and exit

                in TEMP, OCR2                    ; record first value on CW
                mov DET_CW, TEMP

                rjmp EXINT0_EX

                ; Error Avoidance alrogithm
MOT_ErrAv:      rcall INT0_DIS                   ; Disable EXT_INT0 so it cannot be
                                                 ; retriggered once global interrupts
                                                 ; are re-enabled for nested interrupts

                cbi DDRD, 2                      ; Make sure PORTD bit 2 is set to input
EXINT0_CLR:     sbic PIND, 2                     ; Wait for EXT_INT0: falling edge
                rjmp EXINT0_CLR                  ;

                cbr STATUS, $02                  ; Clear Timeout bit in STATUS
;               andi STATUS, $FD                 ;
                sei                              ; enable global interrupts for TIMER2_INT
                CALL_TIMER1 MOT_Timeout,$02      ; MACRO MOT_Timeout
EXINT0_WT1:     sbrc STATUS, 1                   ; Check if Timer Timeout has occured
                rjmp MOT_ErrAvEX2                ; If so Exit External Interrupt

                sbis PIND, 2                     ; Check for rising edge on ext_int0
                rjmp EXINT0_WT1                  ; Wait for Timeout condition or another
                                                 ; trigger on PORTD bit 1
MOT_ErrAvEX:    cli                              ; Disable Global interrutps so it cannot
                                                 ; intefere with the rest of this interrupt
                rcall INT0_EN                    ; Re-enable EXT_INT0
                clr GENIOI
                ret                              ; return from subroutine

MOT_ErrAvEX2:   cli
                rcall INT0_EN
                ser GENIOI
                ret
                ;End Error Avoidance Algorithm
```

## Sensing Algorithm 1c

Sensing Algorithm 1c is a slight variation of Sensing algorithm 1. Since a analog signal is now used to detect motion the code had to be modified to provide a check on the analog port while swinging. The provides less code because it did not need to use the external interrupts.

### *Sample Code*

```
;****************************************************************************************
;* Subroutine - Motion_Sweep
;* Description: This subroutine is used to sweep the motion sensor CW then CCW
;*                  as the motion detector is being swept, only the first value obtained
;*                  in each direction of motion is recorded resulting in two values
;*                  one stored in DET_CW, and DET_CCW
;* Uses: TEMP
;* Output: DET_CW, DET_CCW
;***************************************************************************************
Motion_SWEEP:   rcall Init_ADC

                sbr STATUS, $01                 ; Setup STATUS to show CW Sweep
                clr DET_CW                      ;
                clr DET_CCW                     ;
                rcall TIM2_PWM                  ; Set Timer2 to PWM Mode and PortD OC2 to output
                ldi TEMP, $0E                   ; Set Servo to Left Most Position
                mov MOT_LOC, TEMP
                out OCR2, MOT_LOC               ;

                ldi TEMP2, $08                  ;
                rcall Motion_Delay          ; Wait until SERVO SWINGS to leftmost position

                inc MOT_LOC                     ;
SWEEP_LP1:      out OCR2, MOT_LOC               ; Incremental Clockwise Servo Swing

                ldi TEMP2, $08
                rcall Motion_Delay
                ;
                rcall Motion_Detect         ; Check Motion_Detector
                ;
                inc MOT_LOC                     ;
                mov TEMP, MOT_LOC
                cpi TEMP, $21                   ; Check for Rightmost Position
                brne SWEEP_LP1

                ldi TEMP2, $0C
                rcall Motion_Delay
                dec MOT_LOC
                cbr STATUS, $01                 ; Setup STATUS to show CCW Sweep
SWEEP_LP2:      out OCR2, MOT_LOC               ;

                ldi TEMP2, $08
                rcall Motion_Delay
;
                rcall Motion_Detect         ; Check Motion_Detector
;
                dec MOT_LOC
                mov TEMP, MOT_LOC               ;
                cpi TEMP, $0E                   ; Check for Leftmost position
                brne SWEEP_LP2

                ret
;*
;***************************************************************************************
;* Interrupt - External Interrupt 0 Handler
;* Description: When an external interrupt occurs on INT0, a motion is detected by
;*                  the sensor. The external Interrupt will check the current direction
;*                  and the direction vector to make sure it is the first value for that
;*                  direction and if so it records it else it exits
;* Uses: TEMP
;***************************************************************************************
Motion_Detect:  push TEMP
                sbrc STATUS, 0                  ; If status bit0 = 0 then execute CCW
                rjmp MOTDET_CW                  ; else execute CW

MOTDET_CCW:     tst DET_CCW                     ; Check to see if First Value of CCW
                brne MOTDET_EX                  ; If not exit else record first value on CCW

                ldi GENIOI, $02                 ;
                rcall ADC_OneRun            ; Call ADC_OneRun with ADMUX set to port2
                cpi GENIOI, $00                 ; $00                   ;
                ;brne MOTDET_EX                 ; If GENIOI != 0 then bad value and exit
                brne CCW_ALT                    ; If GENIOI != 0 then bad value and exit

CCW_REC:        in TEMP, OCR2                   ; record first value on CCW
                mov DET_CCW, TEMP

MOTDET_EX:      pop TEMP                        ;
```

```
                reti

CCW_ALT:        cpi GENIOI, $01                 ; Secondary Threshold for CCW = $01 72
                brne MOTDET_EX                  ; if higher byte != $01 then exit
                mov GENIOI, GENIOR              ;
                cpi GENIOI, $AF                 ; Check if lower byte < $72
                brlo CCW_REC                    ;
                rjmp MOTDET_EX                  ; If not exit

MOTDET_CW:      tst DET_CW                      ; Check to see if First value of CW
                brne MOTDET_EX                  ; If not exit else record first value on CW

                ldi GENIOI, $02                 ; Call ADC_OneRun with ADMUX set to port2
                rcall ADC_OneRun        ;
                cpi GENIOI, $03                 ; $03
                ;brne MOTDET_EX                 ; If GENIOI != 0 then bad value and exit
                brne CW_ALT                     ; If GENIOI != 0 then bad value and exit

CW_REC:                 in TEMP, OCR2                   ; record first value on CW
                mov DET_CW, TEMP

                rjmp MOTDET_EX
CW_ALT:         cpi GENIOI, $02                 ; Secondary Threshold for CCW = $02 80
                brne MOTDET_EX                  ; if higher byte != $01 then exit
                mov GENIOI, GENIOR              ;
                cpi GENIOI, $80                 ; Check if lower byte < $80
                brlo MOTDET_EX                  ; If so exit
                rjmp CW_REC                     ; else record value
```

**Sensing Algorithm 2**

Algorithm 2 uses status bits are used to indicate a motion detector hit and a direction of motion detection. In one detection cycle, only one clockwise or counter-clockwise sweep is made. The sweep that is made in that detection cycle is determined by the status bit. While the servo is sweeping the motion detector, the motion detector analog values are checked for a human (hit). If there is a hit, then the detection cycle ends and the results are analyzed making a decision to turn or not. If there are no hits detected when the servo reaches the end of the motion sweep, the status bit is toggled so that the next motion sweep goes in the other direction. Example: If the first sweep is CW, and no hits are found then the next week will be a CCW sweep.

Since there is only one sweep, the range fro $0E-$21 on the single sweep is compared to values to generate the 5 degrees of turning resolution.

- Left - $0E-$14
- Middle Left - $15-$17
- Middle - $18-$1B

- Middle Right - $1C-$1E
- Right - $1F-$21

In order to provide a smarter sweeping and detection algorithm, if a hit was made in the middle left or left direction, the status bit was changed so that the sweep starts in the left. If a hit was detected on the middle right or right direction then the sweep would start from the right.

Algorithm 2 also counted the number of No Detects, and when a certain number of No Detect conditions was met, it would randomly turn serverBOT in any direction.

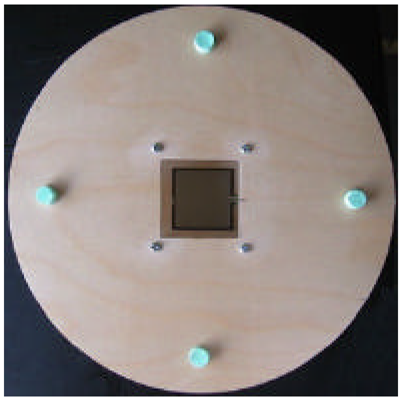
Algorithm 2 is different from algorithm 1 because the motion detector is swept at a faster speed, and swept while the serverBOT is in motion. Algorithm 2 also produces 1 sweep per detection cycle. With algorithm 1, the behavior of serverBOT would check the motion detector and then check the other analog ports in sequential order. In algorithm 2, since the serverBOT is in motion while detecting for a human hit, the other analog ports are checked at the same time the motion detector is checked.

*Sample Code*

```
;***************************************************************************************
;* Subroutine - Motion_Sweep
;* Description: This subroutine is used to sweep the motion sensor CW then CCW
;*                  as the motion detector is being swept, only the first value obtained
;*                  in each direction of motion is recorded resulting in two values
;*                  one stored in DET_CW, and DET_CCW
;* Uses: TEMP
;* Output: DET_CW, DET_CCW
;***************************************************************************************
Motion_SWEEP:   ;DISP_STRING    nxtline
                cbr STATUS, $08                 ; Clear Status bit 3
                                                ; When Status bit 3 is set then hit recorded
                cbi PORTB, 3
                sbi PORTC, 0
                cbi PORTC, 0

                clr MD_HIT                      ;
                ;mov GENIOI, MD_HIT
                ;rcall HEX2ASCII
                ;DISP_STRING    nxtline

                sbrc STATUS, 4                  ; If Status bit4=1, Jump to CCW_SETUP
                rjmp CCW_SETUP                  ;
CW_SETUP:       ldi GENIOI, MDET_LM             ; else CW_SETUP
                sts MDET_ST, GENIOI             ; Start at Leftmost
                ldi GENIOI, MDET_RM             ;
                sts MDET_END, GENIOI            ; End at RIghtmost
                ;               ;
SWEEP_ST:       rcall TIM2_PWM                  ; Set Timer2 to PWM Mode and PortD OC2 to output
                lds MOT_LOC, MDET_ST            ; Set Servo to Start Position
                out OCR2, MOT_LOC               ;
                ;
                ldi TEMP2, $08                  ; Wait until SERVO SWINGS to Start Position
                rcall Motion_Delay      ;

SWEEP_RST:      rcall CA_HH                     ; Collision Avoidance Human Hit Handler
                ldi GENIOI, $02                 ; Wait for RESET value
                rcall ADC_8bit                  ;
                andi GENIOI, $F0        ;
                cpi GENIOI, $80                 ;
                brne SWEEP_RST                  ;

                sbrc STATUS, 4                  ; If Status bit4=1,CCW
                dec MOT_LOC                     ;       CCW = dec MOT_LOC
                sbrs STATUS, 4                  ; If Status bit4=0,CW
                inc MOT_LOC                     ;        CW = inc MOT_LOC
SWEEP_LP1:      out OCR2, MOT_LOC               ; Incremental Clockwise Servo Swing
                ;
                ldi TEMP2, $03                  ; Delay
                rcall Motion_Delay      ;
                rcall CA_HH                     ; Collision Avoidance Human Hit Handler
                ;
                rcall Motion_Detect     ; Check Motion_Detector
                sbrc STATUS, 3                  ; If Status bit3=1, HIT exit SWEEP
                rjmp SWEEP_EX                   ;
                ;
                sbrc STATUS, 4                  ; If Status bit4=1, CCW
                dec MOT_LOC                     ;       CCW = dec MOT_LOC
                sbrs STATUS, 4                  ; If Status bit4=0, CW
                inc MOT_LOC                     ;        CW = inc MOT_LOC
                ;
                lds TEMP, MDET_END              ; TEMP = MDET_END
                cp TEMP, MOT_LOC                ; Compare TEMP to MOT_LOC
                brne SWEEP_LP1
                ;
                sbrc STATUS, 4                  ; If Status bit 4=1,CCW
                rjmp CCW_CH                     ;       CCW = Jump to CCW_CH
                sbr STATUS, $10                 ; CW = SET STATUS bit 4
SWEEP_EX:       ret
                ;
CCW_CH:                 cbr STATUS, $10                 ;       CCW = CLR STATUS bit 4
                rjmp SWEEP_EX                   ; exit
                ;
CCW_SETUP:      ldi GENIOI, MDET_RM             ; CCW_SETUP
                sts MDET_ST, GENIOI             ; Start at Rightmost
                ldi GENIOI, MDET_LM             ;       ;
                sts MDET_END, GENIOI            ; End at LeftMost
                rjmp SWEEP_ST                   ; Begin Sweep
;*
;*****************************************************************************************
;* Interrupt - External Interrupt 0 Handler
;* Description: When an external interrupt occurs on INT0, a motion is detected by
;*              the sensor. The external Interrupt will check the current direction
;*              and the direction vector to make sure it is the first value for that
;*              direction and if so it records it else it exits
;* Uses: TEMP
;*****************************************************************************************
Motion_Detect:  push TEMP
                ldi GENIOI, $02
                rcall ADC_8bit
                ;push GENIOI
                ;rcall HEX2ASCII
                ;pop GENIO I

                sbrc STATUS, 4                  ; If status bit4=1, CCW
                rjmp MOTDET_CCW                         ;       CCW = Execute CCW Detection

MOTDET_CW:      cpi GENIOI, $92                 ; $90 = Hit @ MDelay=$03
                brmi MOTDET_EX                  ; If GENIOI-$A0 < 0, not a valid value

MD_REC:                 sbr STATUS, $08                 ; set status bit3=1 for hit
                in TEMP, OCR2                   ; record value into DET_CW

                mov MD_HIT, TEMP                ;
                sbi PORTB, 3
                sbi PORTC, 0
                cbi PORTC, 0
```

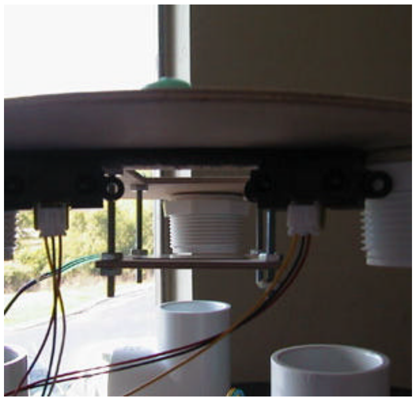
```
MOTDET_EX:      pop TEMP
                ret

MOTDET_CCW:     cpi GENIOI, $70                 ;
                brpl MOTDET_EX                  ; If GENIOI-$6B > 0 not a valid value
                rjmp  MD_REC                    ; record hit value
```

## Force Sensing Resistors



Components – Interlink FSR Part #406 (1-1/2" Square)

The Interlink FSR provides a variable resistance based on the pressure placed on the resistor. The output resistance is inversely proportional to the pressure applied to the FSR. The resistance changes from 100K Ohms to approximately 300 Ohms when at least 10 Grams is applied. The circuit used for the FSR is a basic voltage divider. In this case a 33K Ohm resistor was used in series with the FSR, with the output taken from the 33K Ohm resistor. When there is no pressure applied it is VCC * 33K/FSR = 5V * 33K/OPEN = 0V when there is a lot of pressure applied it is VCC * 33K/FSR where FSR is ~ 300 Ohms so the output is approximately 4.58 Volts.



*Sample Code*

```
;*******************************************************************************
CheckTray:      ldi GENIOI, $03
                rcall ADC_8bit
                cpi GENIOI, $08
                brlo NoTray
                lds TEMP, NFSR_L
                cp GENIOI, TEMP
                brlo TrayEmpty
CTray_EX:       ret
```

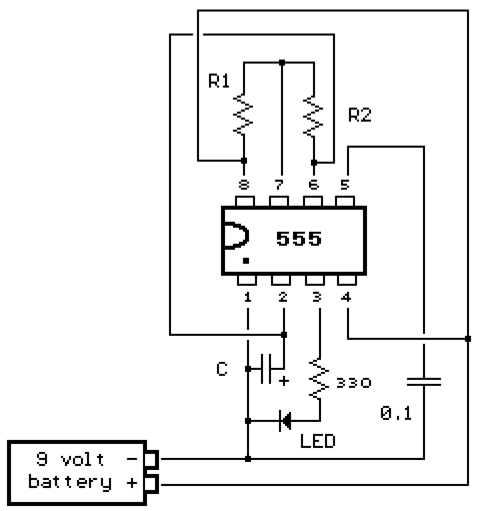
When the serverBOT first loads, it performs a zero calibration in which the current pressure sensed on the FSR is recorded as the neutral position. At any point in which the pressure sensed on the FSR is lower than neutral position recorded then the serverBOT will return the TrayEmpty condition.

**Beacon Detection**

Initially the beacon detection was to be implemented with IR detectors and emitters modulated at 30Khz. The IR detector and emitter should be modulated at 30Khz to prevent any interference with the GP2D12 range finders. Due to the problems I had encountered with generating an accurate and reliable human detection and the limited availability of parts, I did not have enough time and had to change the beacon to a silver piece on the floor. In essence this is not truly a beacon because the robot would have to randomly roll over it before it knew where or what it was, but it does provide a marker for home which is a slight variation of the original design of the serverBOT. Though it is not currently implemented, I will go over the implementation behind setting up an IR detector/emitter at modulated at 30Khz.

**IR Beacon**

The IR beacon can be setup using IR emitters and a 555 timer obtained from radio shack. I do not know the costs of these items, but they are relatively inexpensive.
The follow circuit is used to create an IR beacon from these two components.



Positive Time Interval (T1) = 0.693 * (R1+R2) * C
Negative Time Interval (T2) = 0.693 * R2 * C
Frequency = 1.44 / ( (R1+R2+R2) * C)

R1 = 1K Ohms
R2 = 18.4 K Ohms
C = .001 uF
T1 = 0.0134 ms
T2 = 0.0128 ms
Frequency = 38.095 Khz

These values were obtained by using the 555 timer calculator from this webpage.

http://ourworld.compuserve.com/homepages/Bill_Bowden/555.htm

## IR Detector

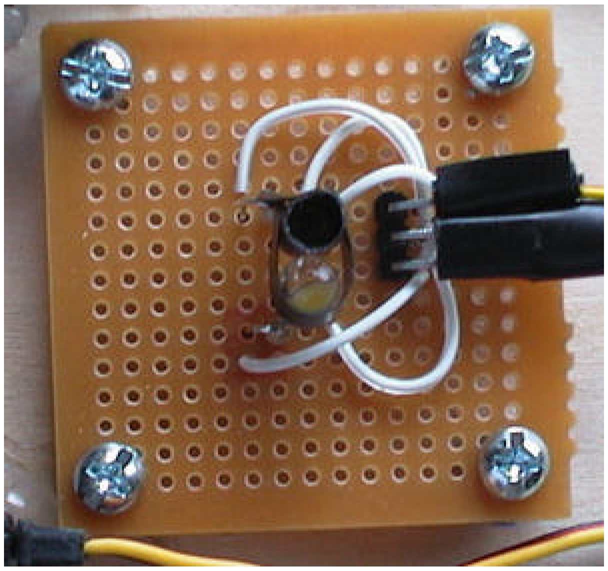
I'm not sure if this works because I haven't tried it to verify it but for the IR detector I would hack the ir cans from Radio Shack. Instructions for hacking them are found in the IMDL website. Once hacked, these IR detectors provide an analog output depending on if the ir emitters were detected.

## CDS Cells

Components – 1K, 320 Ohm Resistors, Ultra Bright White LED, CDS Cell.

The CDS cells act like a variable resistance depending on the intensity of the light that is passed into it. I've mounted the CDS cells underneath the serverBOT with a white LED right next to it. As the serverBOT passes along the floor, the white LED shines into the floor and reflects into the CDS. The color of the floor absorbs or reflects the white led into the CDS cell. As a marker I choose a piece of aluminum foil as a marker for home. The reason why I choose a piece of aluminum foil is because it represents the highest reflective index and would reflect the white light into the CDS cell.

Voltrage Divider with 2 resistors, one driving the LED and the other to provide the voltage divider circuit with the CDS cell



## Behaviors

While using IR range detection for object avoidance, the serverBOT will utilize motion sensors to locate human targets. Once a human target is found the serverBOT will move towards the human target and stop once it reaches the human target. The serverBOT will then stop moving so that items can be removed from the tray. After a timeout period the serverBOT will continue to randomly search for human targets. Once the tray is empty the serverBOT search for home. Once the serverBOT is home it will power down waiting for a reset or powerup command. The reason why the serverBOT is powered down instead of sensing if objects is placed on the tray is because if placing multiple objects on the tray the first object placed on the tray will trigger serverBOT to restart its serving process.

## Conclusion

In the amount of time given, I was able to complete 90% of serverBOT. This was due to the problems I had while detecting with the motion detector. It was very hard trying to get the 10 dollar motion detector to work as opposed to the 60 dollar one. A lot of problems I encountered with the 10 dollar sensor was improper filtering of human ir from the fresnel lens, and sporadic outputs from the signal output. I spent about 2-3 weeks on trying to get the hvwtech pir sensor to work. Most of the time spent was due to testing, trying to figure out how to limit the 60 degree sensing window to a smaller window and trying to make something out of the signals I was getting. I also spent a lot of time trying to get a circuit board printed for my microprocessor. If I had initially wire-wrapped my microprocessor board instead of trying to get it printed and purchased the eltec pyroeletric sensor, I would've completed serverBOT.

The main constraint in the design of serverBOT was the time limitation. Had I known that some sensors would work A LOT better than others I would've used the other sensor from the start. One of the general limitations I encountered was that after spending so much time on trying to get a system to work, I found it really hard to ditch all that I had done to try another way. In which clearly the other way was much better. I also found that time constraints forced me to think less clearly. I spent hours trying to debug two things at once. In the end under less pressure and more sleep, it took me 10 minutes to solve one problem and 2 hours to figure out my other problem. But because I was pressured on time I was certain that I knew what was causing the problem. But in actuality I should've been looking at something else.

Another thing I realized is that I was trying to make things WAY too complex and it made debugging a lot harder for me since everything was in assembly.
I should’ve followed the KISS philosophy. I believe my algorithm for motion detection is still too slow mainly because I provide such a large arc of sweep. I think I am also sampling the motion detector too often and it would’ve been easier to sample it less and provide a smaller turning resolution. In doing so I could’ve sped up the sweep sensing to account for any errors that the serverBOT could’ve encountered. Speed vs quality. In this application speed would’ve probably gotten better results over quality.

## Appendix