Date: *12/12/03*
Student Name: Cem Tozeren
TA : Uriel Rodriguez
Louis Brandy
Instructor. A. A Arroyo

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

CATRAY

FINAL REPORT

Table of Contents:

Abstract:

Catray is a robot that works at parties as a server. It will wander around the room and offer people drinks and snacks while performing obstacle avoidance and collision detection. Every 15 seconds, it will stop and monitor a pressure sensor located on the top platform. The pressure sensor is placed beneath the glasses to observe the pressure exerted by the glass. If there is no glass on the tray, the output of the pressure sensor will reduce below a threshold. The robot will then begin searching for a beacon which actually represents a predefined location to load more drinks. After loading drinks, Catray will continue roaming and serving.

Introduction:

Mobile robots designed to serve people during parties has been an attractive idea since it would eliminate waiting lines for drinks and snacks. Similar server robots were designed in the IMDL lab in the past such as BEERbot(Don Mcmann) and CATE(Mark Antilla). One of the major disadvantages of such robots was the fact that they were working very close to the ground level which made them inadequate servers. Other problems associated with previous robots include the complexity of the serving mechanisms which results in some awkward rotating behaviors and inaccuracy of human detection sensors.

The main motivation for Catray is to build the tray high above the ground to enable easy interaction with people. This will be accomplished by using wooden legs of approximately 1 meter length on top of the carrier platform which will hold the motors and wheels. The legs will support a platform to carry drinks and snacks. A pressure sensor will be implemented on the surface of the platform to alert the microcontroller when the platform is empty or when a drink is served. In the case of an empty tray, the robot will start looking for a designated area to load the tray again. In the case where a drink is served, the robot will interact with the person through an LCD display. IR detectors and bump switches will be used to implement object detection and collision avoidance.

This report describes the physical design, integrated sensors and the behaviors that are built-in to Catray in detail.

Integrated System:

The controller that is used to design the Catray is the Programmable System on Chip microcontroller from Cypress semiconductor. The microcontroller doesn't come on a development board. However being a System-on-a-Chip device, it includes most of the electronics used for the robot internally. Psoc microcontroller is a reconfigurable device. The user can configure the internal structure of the chip via an IDE(Integrated Development Environment) called the Psoc Designer. Psoc Designer has three main parts. In the first part which is called the Device Editor, the programmer chooses the modules to be used in the project. The choices include ADC, DAC, PWM, Counters, LCD interfaces, Uarts and many more. The second part is called the Application Editor where the programmer develops the code for the project. After the code is successfully compiled, the Debugger part enables the programmer to set breakpoints and trace variables to inspect the operation of the software.

Catray weighs 12 pounds. Two servos with sufficient torque were used to carry the weight of the body. These servos are to be interfaced with drivers which are controlled by the microcontroller through pulse with modulation.

The sensors integrated into the robot include a pressure sensor, infrared sensors, bump switches and a photo-camera. The pressure sensor array will be used to provide information about the state of the tray, e.g. full or empty. In the case someone takes a drink from the tray, the microcontroller senses the change in the pressure level and respond to this by displaying a message. When the pressure level on the tray becomes lower than a certain threshold value, the microcontroller will assume that the tray is empty and it will head to reload the tray.

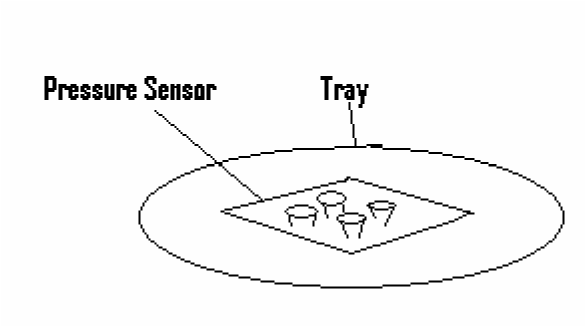

Figure 2 The pressure sensor and the tray.

The infrared sensors and bump switches will be used to avoid collisions and detect obstacles. They will be connected to the analog ports of the microcontroller.

Mobile Platform:

Party Tray is a wheeled robot using two servo-driven wheels arranged in a differential drive style. Differential drive is chosen since it simplifies to turn in place and move in an arc. Two casters are used to balance the platform. The servos used are HS-700BB from Servocity.

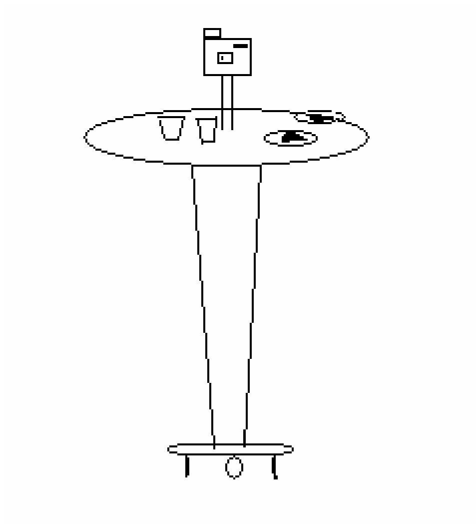

Figure 3 The physical design of the Party Tray robot.

When designing a 35'' platform, stability becomes an important issue. Unstable designs like the one in figure 3 easily fall forward when the motor comes to a jerky stop. In order avoid this problem the bottom part of the robot is made larger than the top part. Above the drive platform, four wooden legs were used to support the tray. The control board is placed beneath the tray to make close contact to the bump switches, pressure sensors and infrareds. The tray and the legs are purchased from Lowe's. The servos are driven by a single battery with separate regulators. The microcontroller interfaces the servos through three wires that run from top part to the bottom. The wires are for ground, VCC and Pulse-Width-Modulation signals.

Another important issue is to use springs under the casters to provide suspension. Especially when the robot is moving on non-ideal surfaces, the wheels tend to disconnect from the ground if springs are not used.

The finalized platform is shown below:

Fig 4: Final Platform for Catray

Sensors:

In order to detect obstacles three front sensor is used on the tray. The infrared detectors are Sharp GP2D12 sensors. GP2D12 finds the range to a target between 3.9 inches and 31.5 inches. The GP2D12 sensors produce an analog voltage output which depends on the range of the detected obstacle. The analog output is connected to an A/D converter in the microcontroller. When the output of the sensor goes above a certain threshold, Catray turns away from the obstacle. The figure below shows the calibration data for GP2D12

IR sensors. It is taken from www.hwmtech.com. It shows the distance-voltage relationship for 4 randomly selected GP2D12 devices.

Distance (cm)	Sample #1	Sample #2	Sample #3	Sample #4	Average Voltage
10	2.451	2.446	2.376	2.423	2.424
12	2.083	2.100	2.025	2.128	2.084
14	1.811	1.845	1.769	1.841	1.817
16	1.620	1.638	1.580	1.645	1.621
18	1.461	1.471	1.415	1.489	1.459
20	1.310	1.336	1.278	1.341	1.316
22	1.211	1.224	1.174	1.226	1.209
24	1.099	1.131	1.081	1.141	1.113
26	1.022	1.050	1.005	1.069	1.037
28	0.965	0.974	0.920	0.992	0.963
30	0.907	0.927	0.883	0.930	0.912
32	0.851	0.870	0.833	0.881	0.859
34	0.800	0.822	0.795	0.841	0.815
36	0.757	0.784	0.739	0.805	0.771
38	0.720	0.742	0.700	0.768	0.733
40	0.695	0.704	0.676	0.730	0.701
42	0.656	0.684	0.637	0.704	0.670
44	0.639	0.665	0.617	0.670	0.648
46	0.612	0.622	0.580	0.650	0.616
48	0.593	0.608	0.561	0.615	0.594
50	0.564	0.582	0.542	0.594	0.571
52	0.543	0.569	0.523	0.575	0.553
54	0.522	0.550	0.503	0.556	0.533
56	0.503	0.531	0.484	0.537	0.514
58	0.483	0.512	0.465	0.525	0.496
60	0.464	0.512	0.465	0.499	0.485
62	0.445	0.493	0.446	0.491	0.469
64	0.447	0.474	0.427	0.469	0.454
66	0.428	0.474	0.407	0.450	0.440
68	0.427	0.455	0.409	0.437	0.432
70	0.413	0.438	0.400	0.429	0.420

Fig 5: Distance vs. Output Voltage Relationship for GP2D12

GP2D12 is highly immune to ambient and laser light levels. However experiments showed that when a light bulb points to it at an angle close to 90 degrees, it can produce false results. The figure below shows the placement of IR range sensors. The rear detector is omitted since Catray never goes back.

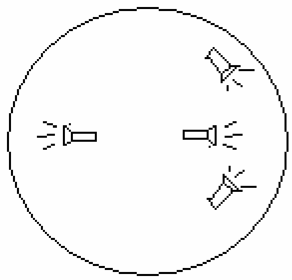

Figure 6: Location of infrared sensors

The sensor in the middle will determine the objects that are in front of the tray while the other two monitors the side ways. Since the tray will be approximately 20 inches in diameter, the infrared sensors will be carefully placed in order to monitor the environment in a continuum.

Bump switches are located at the bottom with uniform displacements. Three bump switches are used to detect collisions. They are connected to digital port and implemented using an Interrupt Service Routine. When the switch is closed, a low-to-high interrupt is triggered and the microcontroller begins to execute the associated ISR. Bouncing signals was a major bottleneck while implementing the bump switches. They caused the interrupt to trigger more than once when the contact switch is closed. It is important to disable the pending interrupts while servicing for the first interrupt in the ISR. Another way to avoid bouncing is to use a capacitor across the bump switch terminals. However I haven't tried this method.

2 Lite-on IR detectors operating at 56 KHz are used to detect the position of an IR beacon. These detector cans normally produce a digital output and are hacked to produce analog signals. The method for hacking Lite-on IR detector is explained in Michael Hatterman's report.

The beacon is built using a microcontroller, 940 nm IR LED’s and 1K resistors. Instead of using a timer to generate the required 56 KHz pulse, I used microcontroller’s PWM modules. The schematic for the IR emitter is given on page 128 of Mobile robots(Jones, Flynn,Seiger). It is important to modulate the 56KHz signal with a 600μsec pulse.

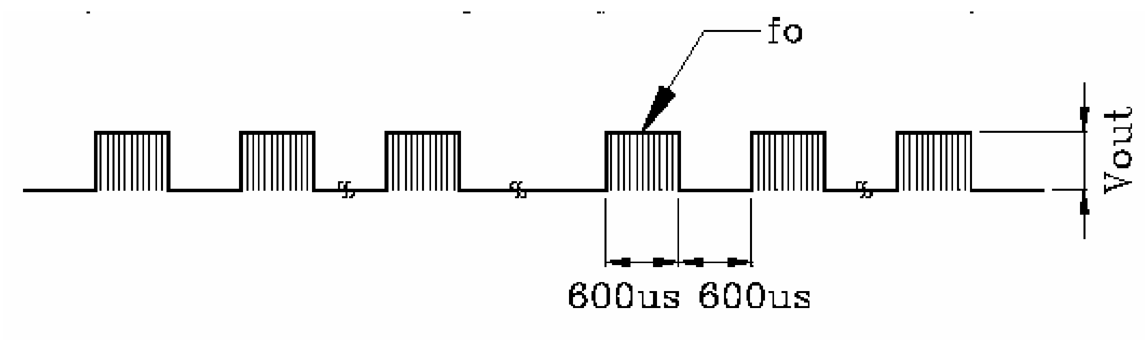

Figure 7: The output of the transmitter (F0=56KHz.)

Modulation improves the SNR of the Lite-on detectors. The schematic for the IR transmitter is shown below.

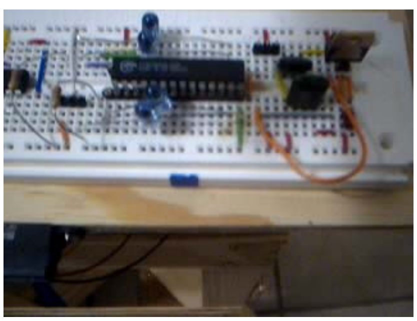

Figure 8: Beacon will determine the place where Catray is to be loaded again.

IESP-12 Pressure sensors from CUI Inc. are used to monitor the contents of the tray. The IESP has a special rubber membrane which bends under pressure. While bent, membrane makes contact with a ceramic plate with resistive traces. As the applied pressure increases, the resistive traces are covered by ceramic and the output resistance drops. The figure below shows the resistance vs. applied pressure graph.

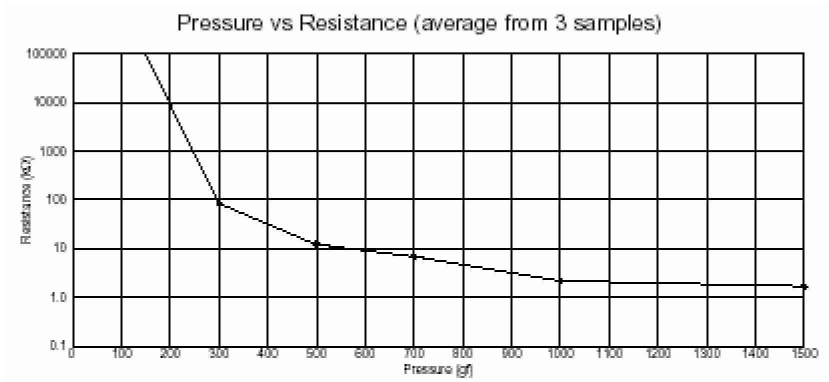

Figure 9: Pressure Vs. Output Resistance Graph

As can be seen, when the load is less than 100 Grams, the output resistance is ~5MΩ. However when the load is increased to 300 grams, the resistance reduces to 100KΩ. Since the glasses that are used on the tray weigh around 300 grams, a simple voltage divider circuit was built with 100K to detect the changes in the resistance. The output of the voltage divider is connected to a buffer to isolate the microcontrollers analog pin input resistance to effect the measurements. The analog output of the buffer is input to A/D converter on the microcontroller and a threshold-based code is written to interface the pressure sensor.

Figure 10: IESP-12

The pressure sensors are placed on a cup holder which is used to secure the glass on a jerky platform. The pressure sensor together with the cup holder is shown below.

Figure 11: Specialized Pressure Sensor

The last device used on the platform is the LCD display. It is a Sharp LM 242 LCD display with a Hitachi 44780 controller. The Psoc microcontroller includes an LCD module which is specifically built to interface the Hitachi 44780 LCD controllers. Using the LCD interface module, I didn’t have to get involved with control signals between the LCD and microcontroller. This greatly simplified the interface of LCD to the microcontroller.

Behaviors:

After reset, the Party Tray will start wandering around. Meanwhile it will employ collision avoidance and will output a short beep signal if there is an object close enough. Through an LCD display on the tray, it will offer people drink. Once somebody gets one, it will indicate on the LCD. After 15 seconds, it will stop and check the pressure sensor. If the pressure sensor indicates that the glass is on the tray, it will wait for 15 seconds more and then continue roaming. In case the pressure sensor indicates no glass, the robot will begin to turn in place in order to locate the beacon. Once the beacon is locked, it will move towards the beacon to load more drinks.

Conclusion:

I've proposed an improvement over past server robot designs by building a more practical party robot. I think I've mostly accomplished the goals I've set for myself. I had problems with finding a pressure sensor suitable for the design. Initially I was planning to use a device that could provide the pressure information of the overall surface. The devices I've found for this purpose had prices that are well over the project budget limit. Thus I had to use the point-load IESP pressure sensors. In the future, I am planning to modify this aspect of the robot.

Parts Used:

HS700-BB Servo Motors from Servo-city (25$ each)

GP2D12 IR Range Sensors from Acroname Electronics (11$ each)

Lite-on IR detector Cans (2$ each)

Psoc Microcontroller from Cypress Microsystems (Requested Samples)

Microcontroller Development Kit (80$)

Bump Switches (2$ each)

IESP-12 Pressure Sensor from Cui Inc (6.95$ each)

9.6 Volt Radioshack Battery (20$)

Software Written for the Party Tray

```
include "m8c.inc"
include "PGA_1.inc"
include "PGA_2.inc"
;include "PGA_3.inc"
include "AMUX4_1.inc"
include "AMUX4_2.inc"
include "Force.inc"
include "DELSIG8_1.inc"
include "lcd_1.inc"
export cont, ADCVal, turn_left, turn_right , turn_back, reverse,right_beacon,
left_beacon
export Loops_til_stop, delay_45ms
area bss(RAM)
   ADCVal:   BLK  1   ;Temp variable containing 8 most significant bits of ADC
result
         Loops_til_stop:   BLK 1
right_beacon: BLK 1
cont: BLK 1
left_beacon: BLK 1
Y: BLK 8
cont1: BLK 1
ave: BLK 1
area text(ROM,REL)

THE_STR:

DS "Welcome to the IMDL"

DB 00h              ; String should always be null terminated

Second_STR:

DS "Media Demo Day!"

DB 00h

Third_STR:

DS "Would you like a drink?"

DB 00h

Fourth_STR:

DS "Look out! Turning Right!"
```

```
DB 00h

export _main

_main:
        M8C_SetBank0
        M8C_EnableGInt                ;enable interrupts using m8c.inc macro
        mov     REG[INT_MSK0],20h
        ;M8C_SetBank0



        mov     [ADCVal],FFh
        mov     [Loops_til_stop],10h;
        mov [cont], Ah


        mov   [Y],'T'
        mov    [Y+1],'U'
        mov    [Y+2],'Z'
        ;M8C_SetBank0
        call   LCD_1_Start        ; Initialize LCD


        ;call   LCD_1_Start
   mov    A,01h           ; Set cursor postion at row = 0
   mov    X,01h           ; col = 1
   call   LCD_1_Position
   mov    A,>Second_STR      ; Load pointer to ROM string
   mov    X,<Second_STR
   call   LCD_1_PrCString    ; Print constant "ROM" string
        call delay_45ms
        call delay_45ms
        mov    A,00h           ; Set cursor postion at row = 0
   mov    X,01h           ; col = 1
   call   LCD_1_Position
   mov    A,>THE_STR      ; Load pointer to ROM string
   mov    X,<THE_STR
   call   LCD_1_PrCString    ; Print constant "ROM" string

        call delay_45ms



        mov     A,DELSIG8_1_HIGHPOWER      ;Set Power level of ADC
```

```
        call    DELSIG8_1_Start                 ;Start ADC
        call    DELSIG8_1_StartAD               ;Start getting samples
        mov     A,PGA_1_HIGHPOWER               ;Set Power level of PGA
        call    PGA_1_Start
        mov     A,PGA_2_HIGHPOWER               ;Set Power level of PGA
        call    PGA_2_Start

        call delay_45ms
        call SAR_init
        call servo_init
        call delay_45ms

loop1:                                                  ;data display occures
in DELSIG8_1INT.asm
        ;mov    REG[INT_MSK0],20h
        ;call delay_45ms
        ;call delay_45ms
        mov A,[Loops_til_stop]
        dec A
        mov [Loops_til_stop],A

        call delay_45ms
        mov  A,01h      ; specify port pin Port0_2
   call AMUX4_1_InputSelect
   call delay_45ms

        call IR_forward
        call delay_45ms
        ;call delay_45ms
        mov  A,03h      ; specify port pin Port0_6
   call AMUX4_1_InputSelect
        call delay_45ms

        call IR_left
        call delay_45ms
        mov  A,02h      ; specify port pin Port0_4
   call AMUX4_1_InputSelect
        call delay_45ms


        call delay_45ms
        call delay_45ms

        mov A,[Loops_til_stop]
        dec A
        cmp A,04h
```

```
        jnc  res
        call Force_monitor
        res:
        ;call Force_monitor
        jmp     loop1                                   ;end loop1

        delay_90degree:
        push A
        mov A,17h; To turn back use two loops
        loop_90deg:
        push A
        call delay_45ms
        pop A
        dec A
        jnz loop_90deg
        pop A
        ret

delay_20degree:
        push A
        mov A,08h; To turn back use two loops
        loop_20deg:
        push A
        call delay_45ms
        pop A
        dec A
        jnz loop_20deg
        pop A
        ret

        delay_10degree:
        push A
        mov A,04h; To turn back use two loops
        loop_10deg:
        push A
        call delay_45ms
        pop A
        dec A
        jnz loop_10deg
        pop A
        ret

        delay_45ms:
        push A
        mov A,2Bh
        loop_45ms:
```

```
push A
call delay_1ms
pop A
dec A
jnz loop_45ms
pop A
ret

delay_10ms:
push A
mov A,0Ah
loop_10ms:
push A
call delay_1ms
pop A
dec A
jnz loop_10ms
pop A
ret

delay_1ms:
push A
mov A,02h
loop_1ms:
push A
call delay_500us
pop A
dec A
jnz loop_1ms
pop A
ret

delay_500us:
mov A,AAh
loop_500Us:
dec A
jnz loop_500Us
ret


servo_init:
mov A,40h
mov X,9Ch
call PWM16_1_WritePeriod

mov A,04h
```

```
        mov X,10h
        call PWM16_1_WritePulseWidth

        call PWM16_1_DisableInt

        call PWM16_1_Start

        mov A,40h
        mov X,9Ch
        call PWM16_2_WritePeriod

        mov A,FCh
        mov X,08h
        call PWM16_2_WritePulseWidth

        call PWM16_2_DisableInt

        call PWM16_2_Start

        ret

        SAR_init:
        mov A, Force_HIGHPOWER
        call Force_Start
        ret

        IR_forward:

        ;call    DELSIG8_1_StartAD                        ;Start getting samples
        mov  A,[ADCVal]
        ;add A,7Fh
cmp A,28h
jc  nothing
;mov REG[PRT2DR],A
call turn_right
jmp something
nothing:
;mov REG[PRT2DR],00h
something:

        ret

        IR_left:
        ;call    DELSIG8_1_StartAD                        ;Start getting samples
        mov  A,[ADCVal]
        ;add A,7Fh
```

```
cmp A,28h
jc  nothingleft
;mov REG[PRT2DR],A
call turn_right
jmp somethingleft
nothingleft:

somethingleft:

        ret

        IR_right:
        ;call    DELSIG8_1_StartAD                    ;Start getting samples
        mov  A,[ADCVal]
        ;add A,7Fh
cmp A,1000h;NORMALLY 28H
jc  nothingright
;mov REG[PRT2DR],A
call turn_left
jmp somethingright
nothingright:
;mov REG[PRT2DR],00h
somethingright:
;call DELSIG8_1_StopAD
;mov  [ADCVal],A
;add  [ADCVal],7Fh
        ret


        turn_right:

        push A

        mov    A,00h          ; Set cursor postion at row = 0
 mov    X,01h          ; col = 1
 call   LCD_1_Position
 mov    A,>Fourth_STR      ; Load pointer to ROM string
 mov    X,<Fourth_STR
 call   LCD_1_PrCString    ; Print constant "ROM" string

        call delay_45ms
        call PWM16_1_Stop
        call delay_45ms
        mov A,34h
        mov X,08h
        call PWM16_1_WritePulseWidth
```

```
call PWM16_1_DisableInt
call PWM16_1_Start
call delay_90degree
;call delay_45ms

call PWM16_1_Stop
call delay_45ms
mov A,04h
mov X,10h
call PWM16_1_WritePulseWidth
call PWM16_1_DisableInt
call PWM16_1_Start
call delay_45ms
pop A

ret

turn_right_large:

push A
call PWM16_1_Stop
call delay_45ms
mov A,34h
mov X,08h
call PWM16_1_WritePulseWidth
call PWM16_1_DisableInt
call PWM16_1_Start

;call delay_45ms
;call delay_45ms
call delay_45ms

call PWM16_2_Stop
call delay_45ms
mov A,FCh
mov X,08h
call PWM16_2_WritePulseWidth
call PWM16_2_DisableInt
call PWM16_2_Start
call delay_45ms
;call PWM16_2_Start
call delay_20degree
;call delay_45ms

call PWM16_2_Stop
call delay_45ms
```

```
;call delay_45ms
;call delay_45ms
call PWM16_1_Stop
call delay_45ms
;mov A,04h
;mov X,10h

call delay_45ms
pop A

ret

turn_right_small:

push A
call PWM16_1_Stop
call delay_45ms
mov A,34h
mov X,08h
call PWM16_1_WritePulseWidth
call PWM16_1_DisableInt
call PWM16_1_Start

;call delay_45ms
;call delay_45ms
call delay_45ms

call PWM16_2_Stop
call delay_45ms
mov A,FCh
mov X,08h
call PWM16_2_WritePulseWidth
call PWM16_2_DisableInt
call PWM16_2_Start
call delay_45ms
;call PWM16_2_Start
call delay_10degree
;call delay_45ms

call PWM16_2_Stop
call delay_45ms

call PWM16_1_Stop
call delay_45ms

call delay_45ms
```

```
pop A

ret


turn_left:

push A
call PWM16_2_Stop
call delay_45ms
mov A,CCh
mov X,10h
call PWM16_2_WritePulseWidth
call PWM16_2_DisableInt
call PWM16_2_Start
call delay_90degree
;call delay_45ms

call PWM16_2_Stop
call delay_45ms
mov A,FCh
mov X,08h
call PWM16_2_WritePulseWidth
call PWM16_2_DisableInt
call PWM16_2_Start
call delay_45ms
pop A

ret

turn_left_large:

push A
call PWM16_2_Stop
call delay_45ms
mov A,CCh
mov X,10h
call PWM16_2_WritePulseWidth
call PWM16_2_DisableInt
call PWM16_2_Start
call delay_45ms
;call delay_45ms
;call delay_45ms

call PWM16_1_Stop
call delay_45ms
```

```
mov A,04h
mov X,10h
call PWM16_1_WritePulseWidth
call PWM16_1_DisableInt
call PWM16_1_Start
call delay_45ms
;call PWM16_1_Start

call delay_20degree
;call delay_45ms

call PWM16_2_Stop
call delay_45ms

call PWM16_1_Stop
call delay_45ms

call delay_45ms
pop A

ret

turn_left_small:

push A
call PWM16_2_Stop
call delay_45ms
mov A,CCh
mov X,10h
call PWM16_2_WritePulseWidth
call PWM16_2_DisableInt
call PWM16_2_Start
call delay_45ms


call PWM16_1_Stop
call delay_45ms
mov A,04h
mov X,10h
call PWM16_1_WritePulseWidth
call PWM16_1_DisableInt
call PWM16_1_Start
call delay_45ms
;call PWM16_1_Start

call delay_10degree
```

```
;call delay_45ms

call PWM16_2_Stop
call delay_45ms

call PWM16_1_Stop
call delay_45ms

call delay_45ms
pop A

ret

turn_back:
push A
call PWM16_1_Stop
call delay_45ms
mov A,34h
mov X,08h
call PWM16_1_WritePulseWidth
call PWM16_1_DisableInt
call PWM16_1_Start
call delay_90degree
call delay_90degree

call PWM16_1_Stop
call delay_45ms
mov A,04h
mov X,10h
call PWM16_1_WritePulseWidth
call PWM16_1_DisableInt
call PWM16_1_Start
call delay_45ms
pop A
ret

reverse:; Reverse should be followed by turn left right or back
push A
call PWM16_1_Stop
call delay_45ms
call delay_45ms
call PWM16_2_Stop
call delay_90degree

mov A,34h
mov X,08h
```

```
call PWM16_1_WritePulseWidth
call PWM16_1_DisableInt
call PWM16_1_Start

call delay_45ms
call delay_45ms
mov A,CCh
mov X,10h
call PWM16_2_WritePulseWidth
call PWM16_2_DisableInt
call PWM16_2_Start

call delay_90degree
call delay_90degree

call PWM16_1_Stop
call delay_45ms
call delay_45ms
call PWM16_2_Stop
call delay_45ms
call delay_45ms
mov A,04h
mov X,10h
call PWM16_1_WritePulseWidth
call PWM16_1_DisableInt
call PWM16_1_Start
call delay_45ms

call delay_45ms
mov A,FCh
mov X,08h
call PWM16_2_WritePulseWidth
call PWM16_2_DisableInt
call PWM16_2_Start
call delay_45ms
pop A
ret

Force_monitor:
mov [cont1], 00h
mov [ave], 00h
call PWM16_1_Stop
call delay_45ms
call PWM16_2_Stop
call delay_90degree
```

```
 mov    A,00h           ; Set cursor postion at row = 0
mov    X,01h           ; col = 1
call   LCD_1_Position
mov    A,>Third_STR      ; Load pointer to ROM string
mov    X,<Third_STR
call   LCD_1_PrCString    ; Print constant "ROM" string

        mov  A,01h       ; specify port pin Port0_2
call AMUX4_1_InputSelect
        call delay_45ms
        mov A,02h
        call AMUX4_2_InputSelect
        call Force_GetSample



        add A,1Fh
cmp A,05h
jnc  no_pressure
;mov REG[PRT2DR],A
;mov REG[PRT2DR],A

call find_beacon;
;jmp pressure
no_pressure:
call PWM16_1_Start
call delay_45ms
call PWM16_2_Start
mov [Loops_til_stop],10h;normally 1ah

        ret

        find_beacon:
        ;call delay_90degree; normally only one delay
        call delay_90degree
        call delay_90degree
        mov A,00h
        call AMUX4_2_InputSelect
        call delay_45ms
        call Force_GetSample
        add A,1Fh
mov [right_beacon],A


mov A,01h
```

```
        call AMUX4_2_InputSelect
        call delay_45ms
        call Force_GetSample
        add A,1Fh
    mov [left_beacon],A

    cmp A,[right_beacon]
jc  right_beacon_code
    cmp [left_beacon],11h
    jnc go_get_beacon
    call delay_45ms
    call delay_45ms
    call turn_left_large
    call delay_45ms
    call delay_45ms
    jmp find_beacon



    right_beacon_code:
    cmp [right_beacon],11h
    jnz go_get_beacon
    mov REG[PRT2DR],FFh
    call delay_45ms
    call delay_45ms
    call turn_right_large
    call delay_45ms
    call delay_45ms
    jmp find_beacon

    go_get_beacon:
    mov REG[PRT2DR],0Fh

    call delay_45ms
    mov [cont],07h
    loop:
    call delay_10degree
    call delay_45ms
        call delay_45ms
    call servo_init
        call delay_45ms
        call delay_45ms
        ;call PWM16_2_Start

    call delay_45ms
        mov  A,01h       ; specify port pin Port0_2
```

```
call AMUX4_1_InputSelect
call delay_45ms

        call IR_forward
        call delay_45ms
        ;call delay_45ms
        mov  A,03h       ; specify port pin Port0_6
call AMUX4_1_InputSelect
        call delay_45ms

        call IR_left
        call delay_45ms


call PWM16_1_Stop
        call delay_45ms
        call delay_45ms
        call PWM16_2_Stop
        call delay_45ms
        mov A,[cont]
        dec A
        mov [cont],A

mov A,00h
        call AMUX4_2_InputSelect
        call delay_45ms
        call Force_GetSample
        add A,1Fh
mov [right_beacon],A

call delay_45ms
call delay_45ms
mov A,01h
        call AMUX4_2_InputSelect
        call delay_45ms
        mov [cont1],20h
        here:
        call Force_GetSample
        add A,1Fh

mov [left_beacon],A

call delay_45ms
call delay_45ms
mov A,[right_beacon]
cmp A,[left_beacon]
```

```
jc  left
;cmp [right_beacon], 19h
;jnz leave
call delay_45ms
call delay_45ms
call turn_right_small
call delay_45ms
call delay_45ms
jmp right
left:
;cmp [left_beacon], 19h
;jnz leave
call delay_45ms
call delay_45ms
call turn_left_small
call delay_45ms
call delay_45ms
right:

mov A,[cont]
mov REG[PRT2DR],A
;dec A
cmp A,01h
jnc res1
jmp leave
res1:
jmp loop
leave:
call PWM16_1_Stop
        call delay_45ms
        call PWM16_2_Stop
call delay_45ms
loopx:
jmp loopx




ret

        ret
```