Lab 3. Mobile Robot Control

I. Purpose

The purpose of this lab is to design and implement hardware and software for a three-wheeled mobile robot (TMR) and control locally via console keyboard.

II. Problem Statement

Problem 3. Mobile Robot Control.
Implement three-wheeled mobile robot using servos and omni-wheels, and local keyboard controller for TMR to move and illuminate.

We start from the simplest and perform step-by-step improvements.

Problem 3A. Construct a Three-wheeled Mobile Robot (TMR)
Construct a Three-wheeled Mobile Robot (TMR) using servos, omni-wheels, and lights.

Problem 3B. Control Servo Command Example.
Test control servo via PWM with sysfs and command lines.

Problem 3C. Control Servo Shell Script.
Implement control individual servo via PWM with shell script. Find each duty value to stop the servo.

Problem 3D. Control TMR in C.
Implement control triple servos for TMR via PWM with C program with micro-sec input.

Problem 3E. Local Keyboard Control for TMR Servos and Lights.
Implement local keyboard control of TMR on Bone to actuate servos and lights.

III. Technical Backgrounds

First Week

A. Construct a Three-wheeled Mobile Robot (TMR)

1. Overall block diagram

Overall block diagram is illustrated in Fig 3.1. It will be soldered into universal PCB. Light circuit will be soldered also.

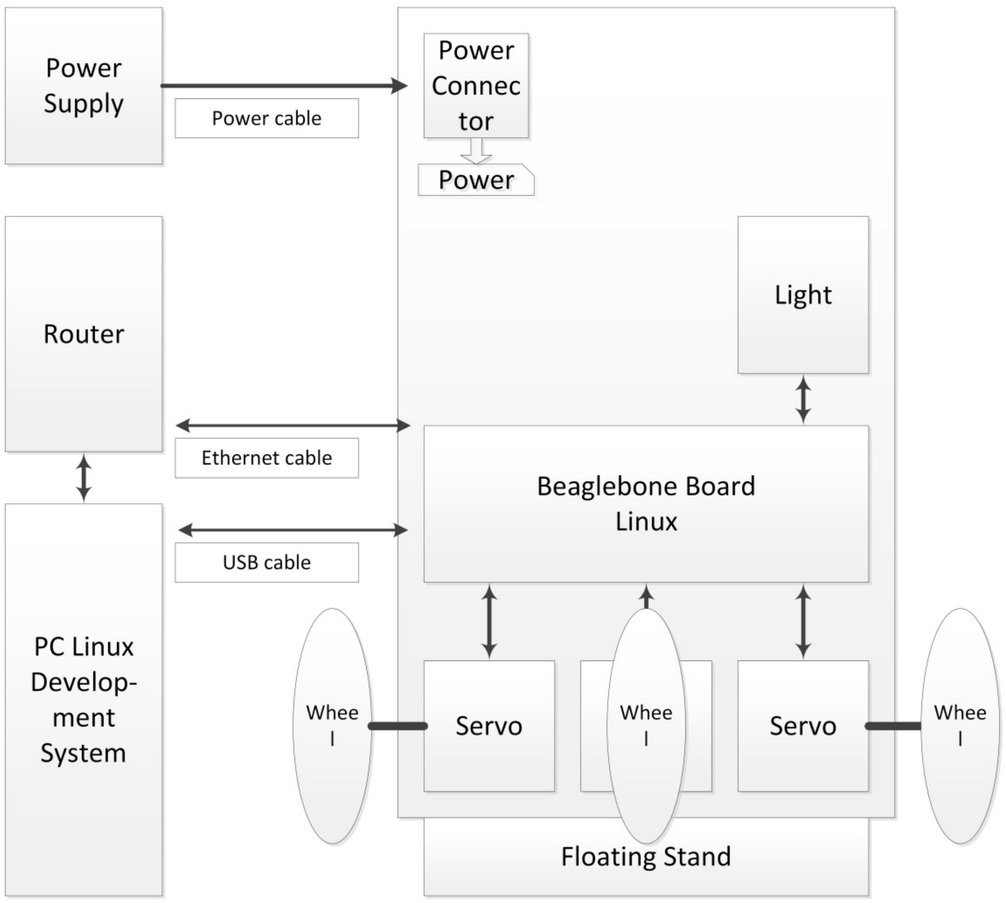


Fig 3.1 Construction of TMR.

We use test platform with wired Ethernet and power supply, and the robot floating.

2. RC Servo

How RC Servo Works

[http://pcbheaven.com/wikipages/How_RC_Servos_Works/]

RC Servos are very popular mechanisms in the world of RC models. No matter if this is a train model, or a car, or a boat, plane or helicopter, there must be at least one servo hidden somewhere within the constructions.

RC Servos are used to convert electrical signal into polar or linear movement. A simple example is the steering system of an RC car. When signal is transmitted from the control to the car, this signal is decoded and sent to a servo. According to this signal, the servo will rotate it's drive shaft for some degrees, and this rotation is translated into wheel steering.

The reason that makes those servos vary handy is that, they have a very easy (and universal) way of driving them with a simple PWM circuit, they can achieve from low to higher torques, enough to move almost everything needed in an RC model, they are very compact and reliable, and most of all, they come with very low prices according to their specifications.

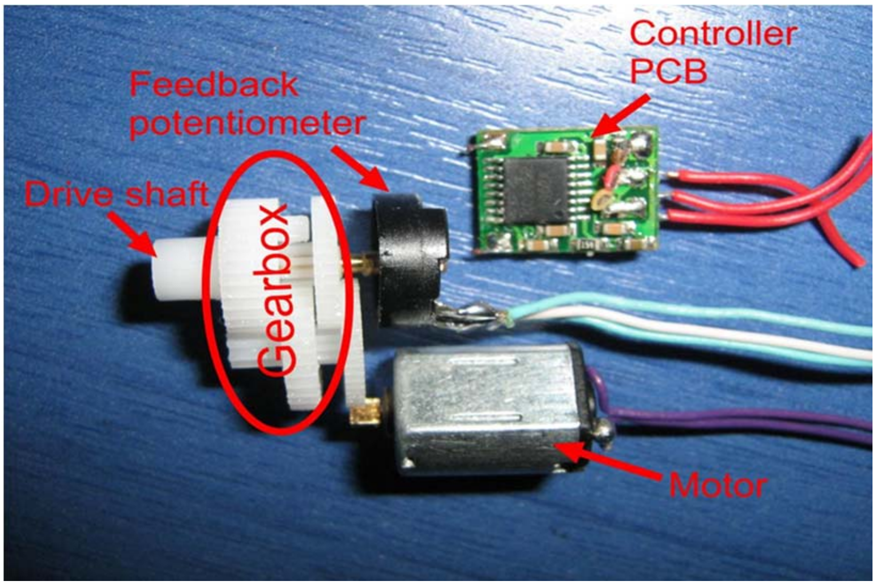


Fig 3.2 Servo components.

The vast majority of RC servos are composed with the same blocks:

- **The controller circuit:** This is the "brain" of the Servo. This circuit is responsible to read the user's input signal (pulses) and translate it into a motor revolution in such a way, that the drive shaft will be rotated to the desired position.
- **The feedback potentiometer:** The shaft of the potentiometer is attached to the drive shaft of the servo. When the drive shaft rotates, so does the potentiometer. In that way, each and every rotation angle of the drive shaft, corresponds to a different resistance of the potentiometer. By reading the potentiometers' resistance, the controller is able to know the exact angle of the drive shaft of the servo.
- **The motor:** This is usually a small high speed DC motor controlled by an H-bridge circuit attached to the servos' controller.
- **The gearbox:** The gearbox will drive the motor's revolution to the drive shaft. Also, the rpm will be significantly reduced and the torque will be increased. The torque is one of the main characteristics of RC servos.
- **The drive shaft:** When all of the above operate in perfect harmony, the drive shaft will be rotated with accuracy to the user's requested angle.

Analog and digital servos

Analog servo	30 to 50 Hz (20 ms to 33 ms)
Digital servo	300 to 400 Hz (2.5 ms to 3.3 ms)

Full torque from the beginning of movement
Tighter dead-band
Faster response time
Increased holding power and max torque
Cons: Expensive. More power.

Servo PWM period & width
[Servo Control http://en.wikipedia.org/wiki/Servo_control]

Servo Control

Servo control from a radio control receiver to the servos is done by sending each servo a PWM (pulse width modulation) signal, a series of repeating pulses of variable width.

PWM period

The servo expects to see a pulse every 20 ms (50 Hz), however this can vary within a wide range that differs from servo to servo.
With many RC servos, as long as the "frame rate" (how many times per second the pulse is sent, aka the pulse repetition rate) is in a range of 40 Hz to 200 Hz, the exact value of the frame rate is irrelevant.

PWM width

Most RC servos move to exactly the same position when they receive a 1.5 ms pulse every 6 ms (a duty cycle of 25%) as when they receive a 1.5 ms pulse every 25 ms (a duty cycle of 6%) -- in both cases, they turn to the center position (neutral position).
Most RC receivers send pulses to the RC servo at some constant frame rate, changing only the high time.

Servo PWM input voltage

Voltage needed for pwm input of servo?

[http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&p=190642]

3.3v should be fine for a servo. You could probably go lower, this will however be determined by the servo that you use. If you want to test it simply use a simple voltage divider connected to your output pin and start low and work your way up till the servo responds.

I have had no problems driving the signal input of servos from 3.3V AVRs. My servos are all quite new, I guess there could be problems with older servos.

2. Continuous Rotation Servo

Using continuous rotation servo, we can generate mechanical rotation motion with PWM control input.

Parallex continuous rotation servo, [http://www.parallax.com/product/900-00008]

The Parallax Continuous Rotation Servo is ideal for robotics and basic movement projects. It is designed for continuous rotation and is easily interfaced with any Parallax microcontroller.

Key Features:

- Bidirectional continuous rotation
- 0 to 50 RPM, with linear response to PWM for easy ramping
- Accepts four mounting screws
- Easy to interface with any Parallax microcontroller or PWM-capable device
- Very easy to control with PBASIC's or SX/B's PULSOUT commands
- Weighs only 1.5 oz (42.5 g)

Note: Servo current draw can spike while under load. Be sure that your application's power supply and voltage regulator is prepared to supply adequate current for all servos used. Do not try to power this servo directly from a BASIC Stamp module's Vdd or Vin pins; do not connect the servo's Vss line directly to the BASIC Stamp module's Vss pin.

Servo Modification to Continuous Rotation

Refer "HOW TO MODIFY A SERVO FOR 360 DEGREES OF MOTION", http://www.laureanno.com/RC/servo-rotate.htm]

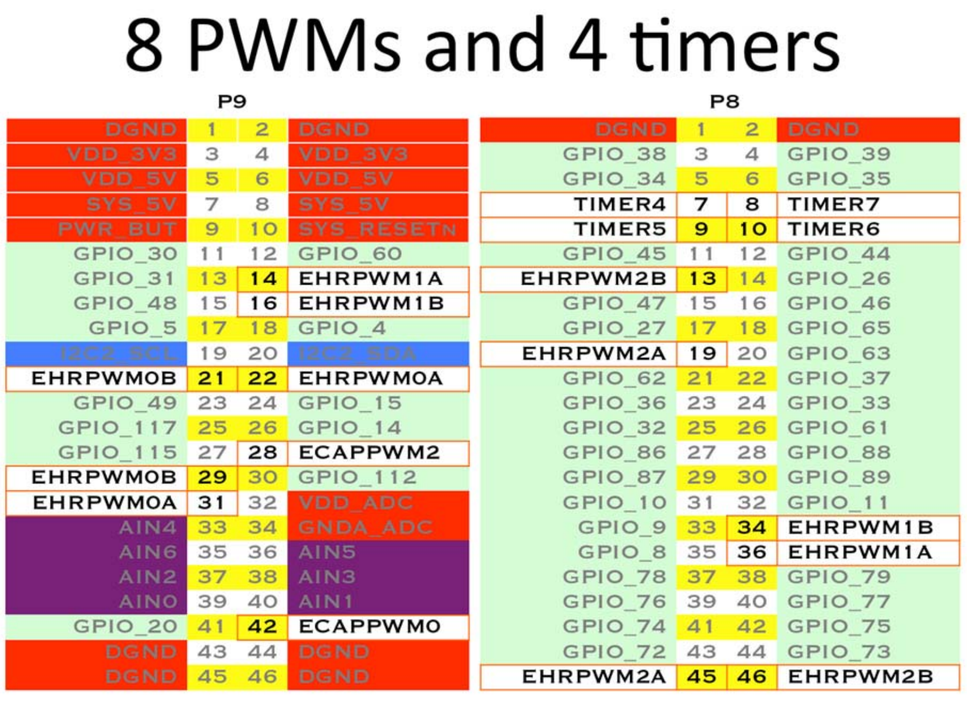
3. PWM signals on Beaglebone

PWM in Beaglebone

Refer: analogWrite(pin, value, [freq], [callback])

http://beagleboard.org/support/BoneScript/analogWrite/

PWM pins in P8/P9 are illustrated in Fig. 3.3.



8 PWMs and 4 timers

P9				P8			
DGND	1	2	DGND	DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3	GPIO_38	3	4	GPIO_39
VDD_5V	5	6	VDD_5V	GPIO_34	5	6	GPIO_35
SYS_5V	7	8	SYS_5V	TIMER4	7	8	TIMER7
PWR_BUT	9	10	SYS_RESETN	TIMER5	9	10	TIMER6
GPIO_30	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
GPIO_31	13	14	EHRPWM1A	EHRPWM2B	13	14	GPIO_26
GPIO_48	15	16	EHRPWM1B	GPIO_47	15	16	GPIO_46
GPIO_5	17	18	GPIO_4	GPIO_27	17	18	GPIO_65
I2C2_SCL	19	20	I2C2_SDA	EHRPWM2A	19	20	GPIO_63
EHRPWM0B	21	22	EHRPWM0A	GPIO_62	21	22	GPIO_37
GPIO_49	23	24	GPIO_15	GPIO_36	23	24	GPIO_33
GPIO_117	25	26	GPIO_14	GPIO_32	25	26	GPIO_61
GPIO_115	27	28	ECAPPWM2	GPIO_86	27	28	GPIO_88
EHRPWM0B	29	30	GPIO_112	GPIO_87	29	30	GPIO_89
EHRPWM0A	31	32	VDD_ADC	GPIO_10	31	32	GPIO_11
AIN4	33	34	GNDA_ADC	GPIO_9	33	34	EHRPWM1B
AIN6	35	36	AIN5	GPIO_8	35	36	EHRPWM1A
AIN2	37	38	AIN3	GPIO_78	37	38	GPIO_79
AIN0	39	40	AIN1	GPIO_76	39	40	GPIO_77
GPIO_20	41	42	ECAPPWM0	GPIO_74	41	42	GPIO_75
DGND	43	44	DGND	GPIO_72	43	44	GPIO_73
DGND	45	46	DGND	EHRPWM2A	45	46	EHRPWM2B

Fig. 3.3. PWM pins in Beaglebone

Beaglebone contains three groups of PWM blocks, with 2 PWM outputs each. Each PWM output can be assigned to one of two GPIOs to have physical output pin. Check references [Tech Ref Man].

Select 3 PWM signals with bold font:

PWM0A	GPIO0_02	Mode 3	P9.22

	GPIO3_14	**Mode 1**	**P9.31**	// Select PWMxA consistently
PWM0B	GPIO0_03	Mode 3	P9.21	
	GPIO3_15	Mode 1	P9.29	
PWM1A	GPIO2_16	Mode 2	P8.36	
	GPIO1_18	**Mode 6**	**P9.14**	// Select PWMxA consistently
PWM1B	GPIO2_17	Mode 2	P8.34	
	GPIO1_19	Mode 6	P9.16	
PWM2A	**GPIO0_22**	**Mode 4**	**P8.19**	// Select PWMxA consistently
	GPIO2_06	Mode 3	P8.45	// Seems to be used by the Kernel
PWM2B	GPIO0_23	Mode 4	P8.13	
	GPIO2_07	Mode 3	P8.46	

4. Connecting PWM output to Servo

We add 5 Kohm resistors in between: The same reason as Lab 2.

B. Control Servo Command Example

5. Control Servo Commands with sysfs

Refer
"EBC Exercise 13 Pulse Width Modulation",
http://elinux.org/EBC_Exercise_13_Pulse_Width_Modulation

"Unable to access PWM from userspace on BBB",
https://groups.google.com/forum/#!category-topic/beagleboard/beaglebone-black/wjbOVE6ItNg

Note that some other web sites explain PWM control methods with older Kernels, and hence do not work on Beaglebone with new kernel version.

Also refer Lab procedure for detailed commands.

Second week
C. Control Servos Shell Script

6. Shell script for control servo

You can control PWMs using shell script similar to Lab 2.
Collect commands listed in Step 2 of Lab Procedure and construct shell script [in Design step].

D. Control TMR in C

7. Control individual servo in C

Convert control servo shell script to C program:

```
Control_PWM0_Servo.c                      //us input
```

8. Triple PWM servo control in C

Extend the above program to construct

```
Control_Triple_PWM_Servos.c               //us input
```

E. Keyboard control of TMR.

9. Get a key input value in raw mode and process

Use the file getche.c developed in Lab 1C.

10. Keyboard control of TMR

TMR Kinematics: Convert v to w

It is described as TMR Kinematics:

w = T * v

where

v = [vx, vy, wr]’: Cartesian linear/rotational velocity,

w = [w1, w2, w3]’: Wheel rotational velocity.

Note that

vx means move forward/backward.

vy means move right/left.

wr means rotate CW/CCW.

Also **T** is a 3x3 transformation matrix given by

$$T = \frac{1}{r}\begin{bmatrix} 0 & 1 & L \\ 1/\sqrt{3} & -2 & L \\ -1/\sqrt{3} & -2 & L \end{bmatrix}$$

where

r: wheel radius (m),

L: length from robot center to wheel center (m).

The values of three PWM duties **p** (in ns) are approximately set proportional to **w** (in rpm), i.e.,

p = G * **w**

where G means a suitable gain.

IV. Equipment and Parts

1. Lab equipment

Router
IBM PC with Windows and Linux (Dual boot)
Embedded board Beaglebone with cables and 4GB SD

2. Electronic parts

ID	Part No	Description	Qty/ group	Unit price
30	-	Servo Mount	3	0
31	GA5	150 x 200 mm Universal PCB	1	8,000
32	PH01(2.54)-DS80P-20MM	Pin Header, 80 pin, 2.54 mm pitch, 20 mm height	2	900
33A	Servo	Parallax Continuous Rotation Servo	3	$12.59
34	Light Omni-wheel	Light Omni-wheel	3	5,460

V. Design

Pre-report for first week

1. Design hardware circuit for three-wheeled mobile robot (Problem 3A).

Use electronic parts listed above, and also parts listed in Lab 2.
Design circuit from Beaglebone P8/P9 to three PWM servos and lights.
Apply "SAFE GPIO OUTPUT CONNECTION HOW TO" explained in Lab 2.

2. Read Step 2 of Lab Procedure and summarize commands for PWM (Problem 3B)

Pre-report for second week

3. Prepare the control servo shell script (Problem 3C)

Collect commands listed in Step 2 of Lab Procedure and construct shell scripts.

Shell programs for triple PWMs:

Acquire_Triple_PWMs.sh	Before Control_Triple_PWMs only once after power on.
control_PWM0.sh	Loop for duty. If neg duty, stop and exit.

Control_Triple_PWMs.sh	Loop for duties. Any neg duty stop and exit.
[Release_Triple_PWMs.sh]	Stop and release 3 PWMs before power off.

Note that Release_Triple_PWMs system can be done by power off. Hence you can omit this.

Contents of Acquire_Triple_PWMs.sh

1. Acuire PWMSS.
2. Add first PWM module bone_pwm_P9_31 for PWM0A.
3. Add second PWM module bone_pwm_P9_14 for PWM1A.
4. Add third PWM module bone_pwm_P8_19 for PWM2A.
5. Confirm result: List suitable directory.

4. Design C program for servo control Problem 3D

Reuse shell "Acquire_Triple_PWMs.sh" to acquire three PWMs.

You may prepare individual servo control programs in C, and triple servo control programs in C: Test simple programs first.

Devise how to compensate the dead-band to improve low-speed behavior.
Hint. You may draw a function s=f(d): Speed s vs PWM duty d (containing dead-band), and utilize inverse function f^{-1} in your program to compensate the dead-band.

Algorithm of control_PWM0_Servo.c [Individual servo 0 control]

```
0. Print title.
1. Check if PWM0 is acquired.
2. Init PWM0A via sysfs.
5. Loop
        User input of duty_ms (1.0 to 2.0 ms)
        Break if duty_ms < 0.
        Convert linearly from duty_ms to duty_ns, and get duty_ns_str's.
        Compensate deadband
        Control PWM with duty_ns_str
8. Stop PWMs
9. Close Files
```

Algorithm of Control_PWM_Servos.c [Triple PWMs control]

```
0. Print title.
1. Check if PWMs are acquired.
2. Init PWM0A/1A/2A via sysfs.
5. Loop
        User input of three duty_us (1000 to 2000 us)
        Break if any duty_us < 0.
        Convert linearly from duty_us to duty_ns, and get three duty_ns_str's.
```

Compensate deadband
Control three PWM with duty_ns_str

8. Stop PWMs
9. Close Files

5. Design Local keyboard control program for TMR (Problem 3E).

Reuse shell "Acquire_Triple_PWMs.sh" to acquire three PWMs.

Suggested algorithm for Keyboard_Control_TMR.

Required sequence

User input:

A) Raw Key Input on Bone terminal.
B) Manage integer speed ivx, ivy, iw (integer from -10 to 10, for example) & lights RL, LL.

Control TMR:

C) Convert ivx, ivy, iw to vx, vy, and wr (m/s & rad/s)
D) Convert vx, vy, and w to wheel velocity (w1, w2, w3 in rad/s) using Kinematics.
E) Compensate deadband.
F) Actuate PWMs three PWM duty value (in ns).
G) Actuate Lights.

Data flow

```
User → Raw key input → Key input handling → Bone → Servo control → Servos on TMR
           7 keys                ivx, ivy, iw             vx, vy, w            PWMs (in_ns)

User → Raw key input → Key input handling → Bone → Light control → Lights on TMR
           2 keys                RL, LL                    RL, LL               GPIO (1/0)
```

We suggest two step approach:

Test_Key_Process.c for PC: Test key input processing on PC (Sequence A to D).
Keyboard_Control_TMR.c for Bone: Actual servo/light control for TMR on Bone (Sequence A to G).

A) Single key input from user

Key inputs on Bone using raw key input mode.

Four translation moves (forward, backward, left, and right) using w, x, a, d keys
Stop: s key.
Rotation (CW and CCW): z and c keys.
Lights: Right 'e', Left 'q' keys.

Total nine keys are located in 3x3 leftmost keys on the keyboard:
Actually we use lowercase letters, instead of uppercase letters.

Q: LL	W: +vx. Forward	E: RL
A: -vy. Left	S: Stop	D: +vy. Right
Z: Rotate CCW +w	X: -vx. Backward	C: Rotate CW -w

B) Keys to integer velocities and lights

Keys to Speed up/down control

Speed w.r.t. key input

S (Stop) key:	Set translational/rotational speeds ivx/ivy & iw to zero
A/D/W/X key:	Multiple keystroke increases/decreases the translational speed ivx/ivy by 10 %.
Z/C key:	Multiple keystroke increases/decreases the rotational speed iw by 10%.

where

ivx:	Integer Forward translational velocity. Negative means backward. -100 to 100 [%].
ivy:	Integer Left translational velocity. Negative means right. -100 to 100 [%].
iw:	Integer CCW Rotational velocity . Negative means CW. -100 to 100 [%].

Keys to lights control

Initially all lights are off.

Each key toggles light state.

Example.

Key sequence of SZWADDD causes iw = 10, ivx = 10, ivy = -20.

Key sequence of QEE cause LL = 1, RL = 0.

C) Convert to velocity with physical units

Convert ivx, ivy, iw -100 to 100 [%] to vx, vy, and wr (m/s & rad/s).

You may require gain from integer vel (from key) to float vel (to TMR).

```
vx = gain_v*ivx;
vy = gain_v*ivy;
wr = gain_w*iw;
```

D) Kinematics

Convert vx, vy, and w to wheel velocity (w1, w2, w3 in rpm) using Kinematics.

From v = [vx, vy, wr]' to w = [w1, w2, w3]', you can use

w = T * v

Up to this point, you can test and debug Test_Key_Process on PC.

Suggested Algorithm for Test_Key_Process [on PC]

```
0. Print title
4. Print Key guide
5. Loop
    A. Key input without Enter.
    B. 'ESC' or 'Ctrl-D' key terminates.
    C. Vx, vy, and w [%] from key input: Dec/inc speed by 10 %. Do saturation.
    D. Transform   vx, vy, w [%] → " [m/s, rad/s] → Kinematics → ww_rpm[]→ duty_ns.
    E. Print info key, vx/vy/w, duty_ns
    F. usleep 1 ms
```

E) Control TMR: Velocity to PWM duties

Convert wheel velocity (w1, w2, w3 in rpm) to PWM duty in ns.
Actuate PWM: Use PWM control in C in Design of 3D.

F) Control TMR: Lights

Use Light_control_c designed in Lab 2.

Up to this point, you can test and debug TMR_Control_C on Bone.

Suggested Algorithm for Keyboard_Control_TMR

```
0. Print title
1. Compute parameters
2. Check PWMs are acquired
3. Init PWMs
4. Print Key guide
5. Loop
    A. Key input without Enter.
    B. 'ESC' or 'Ctrl-D' key terminates.
    C. Vx, vy, and w [%] from key input: Dec/inc speed by 10 %. Do saturation.
    D. Transform   vx, vy, w [%] → ‟ [m/s, rad/s] → Kinematics → ww_rpm[] → duty_ns
    E. Compensate deadband
    F. Output PWM via sysfs with duty_ns considering Pos/Neg Deadband/Gain.
        Also actuate Lights.
    G. Print info key, vx/vy/w, duty_ns
    H. usleep 1 ms
```

VI. Lab Procedures

We use Wired floating TMR among three configurations below:

Configurations:	Breadboard	**Wired floating TMR**	Wireless on-floor TMR
TMR	None	Floating	On-floor
Power	Power supply	Power supply	Battery.
Ethernet	Wired	Wired	WiFi
USB cable	YES	N	N
USB Hub	N	Y	Y
Number of wires	P/E/U	P/E	None
Usage	Initial	Long-term I/O test	Short-term move test

First Week

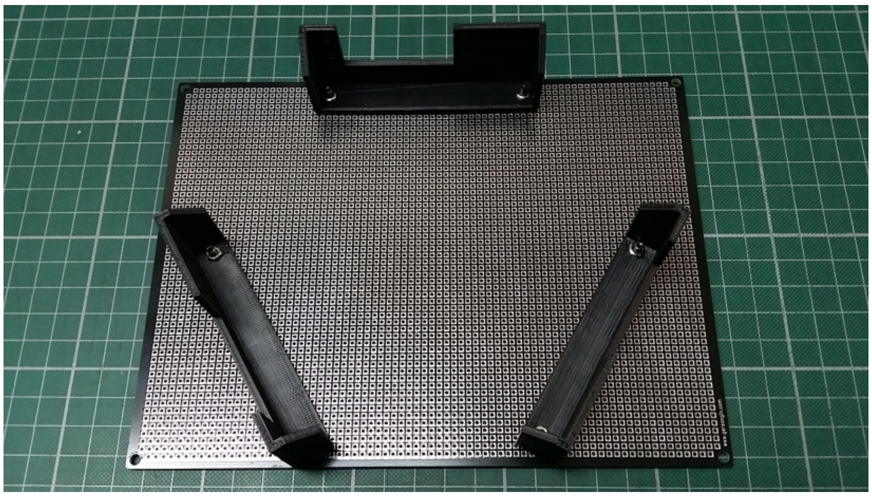
A. Construct a Three-wheeled Mobile Robot (TMR)

Construct a Three-wheeled Mobile Robot (TMR) using servos and omni-wheels.

11. Drilling:

1) PCB hole drilling – 6 holes for Servo mount adaptor.

Note that three servo mounts, more precisely, six servo mount holes should form a regular hexagon as shown below:



Since the space between two mounting holes on the servo mount is 76 mm (or 30 PCB holes with 1/10” spacing), you can place six holes (30, 0), (15, 26), (-15, 26), (-30, 0), (-15, -26), and (15, -26) in units of PCB holes with the origin (0, 0) at the center of PCB.

Six holes are vertically symmetrical, since the number of vertical hole spacing is even (not odd). However, they are not horizontally symmetrical, since the number of horizontal hole spacing is odd (not even): The origin is at a hole, and the number of holes in the right side is one hole larger than the left side.

12. Bottom side: Refer Photo 0.3.

1) Screw three Servo mount adaptors.
2) Screw three Servos to Servo mount adaptors.
3) Screw three Omniwheels to three Servos.
4) Locate two battery holders and fix with foam tow-sided tape.

13. Top side: Refer Photo 0.2.

1) Solder two 2x23 pin headers to connect Bone: Bottom center area.
2) Solder power toggle switch: Bottom left edge.
3) Solder 4-pin header for external power connection: Bottom leftmost edge.
4) Solder LED drive circuit: TR array IC and two Light LEDs: Upper left area.
5) Solder three PWM connectors and resistors: Bottom right edge.

B. Servo Control Command Example

20. Select PWM outputs from Bone.

Remember selections:

PWM0A	GPIO3_14	Mode 1	P9.31
PWM1A	GPIO1_18	Mode 6	P9.14
PWM2A	GPIO0_22	Mode 4	P8.19

Refer Startup_Shutdown_Sequence.docx for startup and shutdown sequence.

21. Acquire three PWMs with Commands

Check PWM subsystem

Check for bone_capemgr.N by checking directory /sys/devices

```
# ls /sys/devices
44e10800.pinmux  breakpoint        ocp.3     soc.1     tracepoint
ARMv7 Cortex-A8  fixedregulator.9   platform  software  virtual
bone_capemgr.8   iio_sysfs_trigger  pmu.0     system
```

In this case, we can use ***bone_capemgr.8***.

Check bone_capemgr.8

```
# ls /sys/devices/bone_capemgr.8/
baseboard  driver  modalias  power  slots  subsystem  uevent
```

Add overall PWM module

Add the am33xx_pwm module to the bone_capemgr.N/slots

```
# echo am33xx_pwm > /sys/devices/bone_capemgr.8/slots
```

Check bone_capemgr.8 again.

```
# ls /sys/devices/bone_capemgr.8
baseboard  driver  modalias  power  slot-4  slots  subsystem  uevent
```

Note that "slot-4" directory is created, which is currently a directory for all PWMs.
It seems that many capes are managed with "slot-N" directories.

Check also

```
# ls /sys/devices/ocp.3/
44e07000.gpio     48046000.timer                 48302000.epwmss    mmc.4
44e09000.serial   48048000.timer                 48304000.epwmss    modalias
44e0b000.i2c      4804a000.timer                 48310000.rng       nop-phy.5
44e10448.bandgap  4804c000.gpio                  49000000.edma      nop-phy.6
44e35000.wdt      4819c000.i2c                   4a100000.ethernet  power
44e3e000.rtc      481ac000.gpio                  53100000.sham      subsystem
47400000.usb      481ae000.gpio                  53500000.aes       uevent
48042000.timer    48200000.interrupt-controller  56000000.sgx
```

```
       48044000.timer    48300000.epwmss              gpio-leds.7
```

Notice that three epwmss's are generated.

Add individual PWM modules

PWM0A

Add the individual PWM module to the bone_capemgr. For example, EHRPWM0A is pin 31 on the P9 connector. To enable this pin for PWM, use the following command

```
# echo bone_pwm_P9_31 > /sys/devices/bone_capemgr.8/slots
```

Check bone_capemgr.8 again.

```
# ls /sys/devices/bone_capemgr.8/
baseboard  driver  modalias  power  slot-4  slot-5  slots  subsystem  uevent
```

Note that "slot-5" directory is created.

PWM1A

Add the individual PWM module for PWM1A on P9.14.

```
# echo bone_pwm_P9_14 > /sys/devices/bone_capemgr.8/slots
```

Check bone_capemgr.8 again.

```
# ls /sys/devices/bone_capemgr.8/
baseboard  modalias  slot-4  slot-6  subsystem
driver     power     slot-5  slots   uevent
```

Note that "slot-6" directory is created.

PWM2A

Add the individual PWM module for PWM2A on P8.19.

```
# echo bone_pwm_P8_19 > /sys/devices/bone_capemgr.8/slots
```

Check bone_capemgr.8 again.

```
# ls /sys/devices/bone_capemgr.8/
baseboard  modalias  slot-4  slot-6  slots      uevent
driver     power     slot-5  slot-7  subsystem
```

Note that "slot-7" directory is created.

This creates a link in /sys/devices/ocp.#/pwm_test_P9_31. [Done by sysfs?]

```
# ls /sys/devices
44e10800.pinmux  breakpoint         ocp.3     soc.1     tracepoint
ARMv7 Cortex-A8  fixedregulator.9   platform  software  virtual
bone_capemgr.8   iio_sysfs_trigger  pmu.0     system
```

In this case, we can use "ocp.3".

Check /sys/devices/ocp.3:

```
 # ls -F /sys/devices/ocp.3/
44e07000.gpio     4804c000.gpio                  gpio-leds.7
……
47400000.usb     48302000.epwmss            pwm_test_P8_19.12
48042000.timer    48304000.epwmss             pwm_test_P9_14.11
48044000.timer    49000000.edma              pwm_test_P9_31.10
48046000.timer    4a100000.ethernet            subsystem
……
```

Check individual sysfs for PWM.

```
# ls -F /sys/devices/ocp.3/pwm_test_P9_31.10
driver@  duty  modalias  period  polarity  power/  run  subsystem@  uevent

# ls -F /sys/devices/ocp.3/pwm_test_P9_14.11
driver@  duty  modalias  period  polarity  power/  run  subsystem@  uevent

# ls -F /sys/devices/ocp.3/pwm_test_P8_19.12/
driver@  duty  modalias  period  polarity  power/  run  subsystem@  uevent
```

Note. This step 21 is the function required for Acquire-Triple_PWMs.sh.

22. Release three PWMs

Note that release sequence is the opposite of acquire sequence.

Release individual PWM

Release third PWM module bone_pwm_P8_19 for PWM2A

```
# echo -7 > /sys/devices/bone_capemgr.8/slots
```

Release second PWM module bone_pwm_P9_14 for PWM1A

```
# echo -6 > /sys/devices/bone_capemgr.8/slots
```

Release first PWM module bone_pwm_P9_31 for PWM0A

```
# echo -5 > /sys/devices/bone_capemgr.8/slots
```

Confirm

```
# ls /sys/devices/bone_capemgr.8
```

Check that slot-N’s (N=5, 6, 7) are disappeared.

Release overall PWMSS

Try to release overall PWM module:

```
        #echo -4 > /sys/devices/bone_capemgr.8/slots
```

Confirm

```
        # ls /sys/devices/bone_capemgr.8
```

Note that slot-4 is NOT DISAPPEARED. It seems not working...

Cannot find any suitable method up to now.
Current working solution: Power off Bone and restart.

```
        # sudo reboot
```

Note. This step 22 is the function required for Release_Triple_PWMs.sh.
However, no complete solution now…

23. Control PWM0A on P9.31 (Front servo. Servo 0. CCW numbering)

Go to the sysfs directory

```
        # cd /sys/devices/ocp.3/pwm_test_p9_31.10
```

Disable PWM0A

```
        # echo 0 > run                                  // Sometimes it rotates a little and stops!
```

Set PWM period of 3 ms (in ns) and duty of 2 ms (in ns also).

```
        # echo 3000000 > period
        # echo 2000000 > duty
        # echo 1 > run
```

It rotates CCW (outside view).

Set PWM period of 3 ms (in ns) and duty of 1 ms (in ns also).

```
        # echo 3000000 > period                         // Can be skipped
        # echo 1000000 > duty
        # echo 1 > run                                  // Can be skipped
```

Note. This step 23 is the function required for control_PWM0A.sh.

24. Find the dead-band: duty range which stops the Servo 0.

Try various duty values. Note the minimum duty which stops servo as Dmin, and also note the maximum duty which still stops servo as Dmax.
The range [Dmin, Dmax] is called Dead-band.
Set "zero_duty" as the midpoint of duty range found: (Dmin + Dmax)/2.
It is the most safe value of duty to stop the servo afterwards.

Stop Servo 1 with "zero_duty" value (unit in ns):

```
# echo [zero_duty] > duty
```

25. Control PWM1A on P9.14 (Left servo. Sevo 1)

Do the same for PWM1A on P9.14.

26. Control PWM2A on P8.19 (Right Servo: Servo 2)

Do the same for PWM2A on P8.19.

Note. These steps 23 to 26 are the function required form Control_Triple_PWMs.sh.

Second Week

C. Servo Control Shell Script

31. Test prepared shell script to acquire PWMs

Make a working directory DesignLab/3_MobileRobot/c_PWM_Servo_Shell.

Edit the prepared shell script

```
$ gedit Acquire_Triple_PWMs.sh.
```

Change mode of Acquire_Triple_PWM.sh:

```
$ chmod a+x Acquire_Triple_PWMs.sh
```

Acquire PWMs on Bone as root

```
# sudo ./Acquire_Triple_PWMs.sh
```

Note that this script should be executed just once after power on.

If something goes wrong, you need to restart Beaglebone! (Of course reboot PC is not required).

32. Test prepared shell script to control individual PWM

Edit and test individual servo shell scripts:

```
$ gedit control_PWM0A.sh
```

Loop for duty. If neg. duty, stop and exit.

Don't forget to change modes:

```
$ chmod a+x control_PWM0A.sh
```

Test on Bone as root

```
# sudo ./control_PWM0A.sh
```

Note that individual servo shell scripts can be executed as many times as you wish.
That's why we separated acquire shell and servo shell!

33. Test prepared shell script to control three PWMs

Edit and test three servos shell scripts:

Control_Triple_PWMs.sh	Loop for duties. Neg duty stop and exit.

Don't forget to change modes:

```
# chmod a+x Control_Triple_PWMs.sh
```

Test on Bone as root

```
# sudo ./Control_Triple_PWMs.sh
```

34. Release PWMs

Instead of

```
# sudo ./Release_Triple_PWMs.sh
```

Simply reboot Bone

.

D. TMR control in C.

41. Make a working directory

```
$ mkdir -p DesignLab/3_MobileRobot/d_PWM_Servo_C
```

42. Edit C programs

Control_PWM0_Servo.c	// us input
Control_Triple_PWM_Servos.c	// us input

43. Edit Makefile

44. Compile

```
$ make
```

45. Run C programs

Use Acquire_Triple_PWMs.sh before running C program.
Test Control_PWM0_Servo with 0 %, 10 %, 20 %, 50 %, and 100 % of PWM duty.
Check also for reverse direction.

46. Check dead-band compensation

Test Control_Tripel_PWM_Servos on Bone.

Check low-speed behavior.
Is it working well for -1 %, 0%, and 1 % PWM duty input? Slowly rotating with correct direction?

E. Keyboard control for TMR.

51. Make a working directory

```
# mkdir –p 3_MobileRobot/e_TMR_Control_C
```

52. Test_Key_Process on PC

Test key inputs without 'Enter key.
ESC or Ctrl+D exits the program.

53. Test_Key_Process on Bone

It should work also.

54. Test Keyboard_Control_TMR.c

Key inputs

Four translation arrows and stop:

```
        W
A       S       D
        X
```

Rotation:

```
z               C
```

Lights:

```
Q               E
```

If successful, *demonstrate to TA!*

VII. Final Report

Discussion for the following question should be included in the report.

1) Compare mobile robots with two-wheels, three-wheels, and four wheels.

2) Discuss your dead-band compensation method and its experimental result.

3) Are there any disadvantages using continuous rotation RC servo?

4) Is it possible to drive PWM using device driver module? If possible, explain the process roughly.

5) Set your own topic to discuss related to Lab 3, and explain summary of your search result.

VIII. References

[1] Beaglebone Rev A6 System Reference Manual,
http://beagleboard.org/static/beaglebone/latest/Docs/Hardware/BONE_SRM.pdf

[2] AM3359 Datasheet, AM335x ARM Cortex-A8 Microprocessors (MPUs) (Rev. J),
http://www.ti.com/lit/ds/symlink/am3359.pdf

[3] Technical Reference Manual - AM335x ARM Cortex-A8 Microprocessors (MPUs) Technical Reference Manual (Rev. O), http://www.ti.com/lit/ug/spruh73o/spruh73o.pdf