Lab 3. Final Report

Mobile Robot

20154920 송우태

EE405 Electronic Design Lab

Professor Byung Kook Kim

29 April 2016

Table of Contents

Purpose

이번 실험의 목표는 three-wheeled mobile robot 의 회로를 구성해 하드웨어를 제작하고, PC 키보드를 통해서 바퀴의 움직임을 조종할 수 있도록 소프트웨어를 구성하는 것이다.

To add on, we also get to learn and experience pulse width modulation.

Experimental Sequence

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

B. Servo Control Command Example

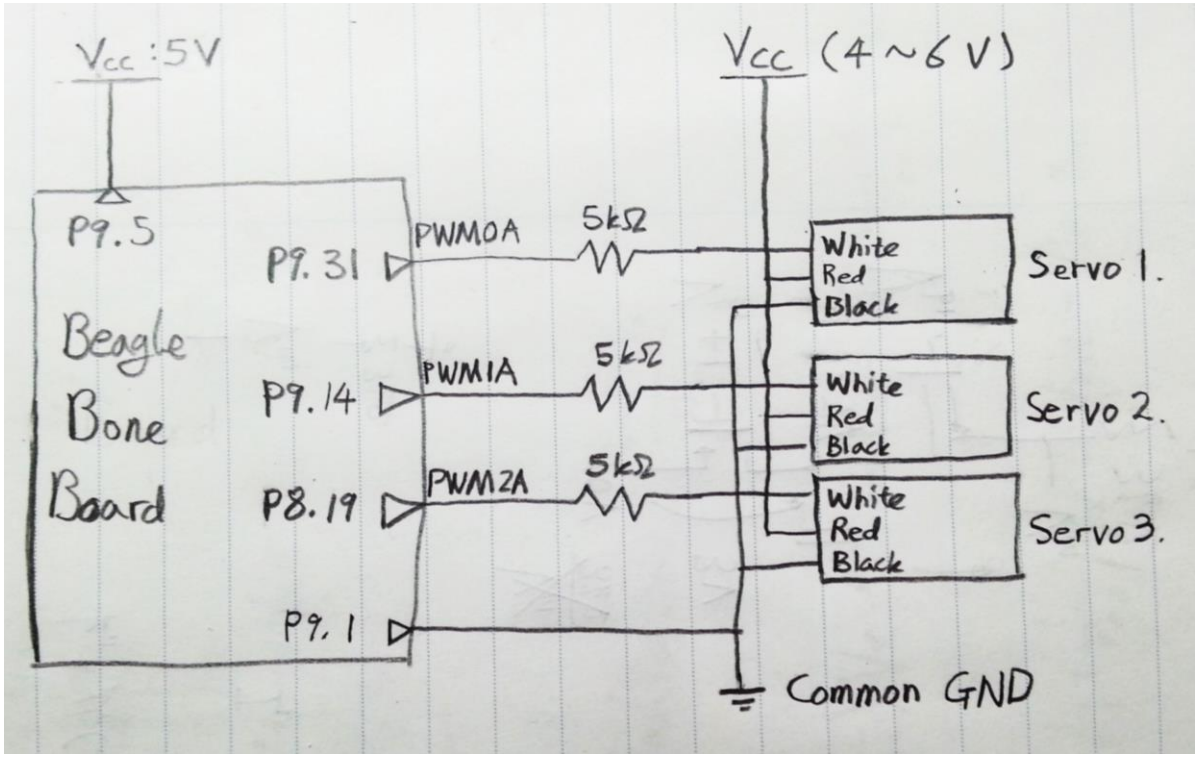
C. Servo Control Shell Script.

D. TMR control in C.

E. Keyboard Control for TMR.

Experimental Results

A. Test User LED Command Example



위의 회로는 A 번 실험에서 제작한 Three – Wheeled Mobile Robot 의 회로이다. PCB 기판에 미리 Servo Mount, Servo 와 Light Omni Wheel 을 조합해 만든 구동부를 장착하고, Pin Header 를 이용해서 기판에 Beaglebone 을 붙일 수 있도록 하였다. Beaglebone 의 P9 의 31 번, 14 번, P8 의 19 번 GPIO pin 이 각각 Servo1, 2, 3 에 신호를 (PWM) 보낼 수 있도록 5kΩ 저항을 이용해서 연결하였다. 또한 3.7V 배터리 2 개를 직렬로 연결해 7.4V 의 전압을 만들어주고, 이를 DC/DC converter 로 5V 로 만든 뒤 Servo 와 Beaglebone 에 전원을 줄 수 있도록 제작하였다. 이렇게 만들어진 회로의 경우 Beaglebone 을 통해서 회전, 전후좌우로 움직일 수 있다.

To add on, the Beaglebone's GPIO P9_31, P9_14, and P8_19 are special pins that can send PWM signals.

B. Servo Control Command Example

우리가 Servo 에 PWM 신호를 주기 위해서 선택한 GPIO pin 은 P9.31, P9.18, P8.19 이다. 이를 위해서는 PWM 모듈을 Beaglebone 에 설치해 줄 필요가 있다. echo am33xx_pwm > /sys/devices/bone_capemgr.8/slots 를 통해서 PWM 신호에 전체적으로 접속할 수 있도록 만들어주었고, echo bone_pwm_p9_31 > /sys/devices/bone_capemgr.8/slots 를 통해서 P9.31 이 PWM0A 신호를 사용할 수 있도록 만들어 주었다. 후자의 경우 p9_14, p8_19 도 반복해서 실행하였다. 이렇게 모듈을 설치한 뒤에는 sys/devices/ocp.3 디렉토리에서 pwm_test_P9_31.10, pwm_test_P9_14.11, pwm_test_P8_19.12 를 통해서 PWM 의 duty, period 등을 설정하여 run 할 수 있다. Period 를 3ms 로 고정하고, duty 값을 조절해서 dead band 를 구한 것은 다음과 같다.

GPIO pin	Start (ns)	Dead band min (ns)	Dead band max (ns)	End (ns)
P9_31 (PWM0A)	660	1429000	1533460	2999939
P9_18 (PWM1A)	720	1444000	1530440	2999999
P8_19 (PWM2A)	720	1418600	1538700	2999939

측정 과정에서 바퀴가 이미 움직이고 있을 때 측정한 dead band 값과 바퀴가 멈춰있는 상태에서 측정한 dead band 의 값이 다른 것을 알 수 있었다. 이는 dead band 가 shift 의 마찰력 때문에 일어나는 현상이기 때문이다. 정지 마찰력보다 운동 마찰력이 더 작기 때문에 바퀴가 돌아가는 상태에서는 더 넓은 범위에서 움직일 수 있다. TMR 의 경우 정지상태에서 keyboard 입력이 주어졌을 때 움직이기 시작하기 때문에 우리가 측정한 것은 바퀴가 움직이지 않는 상태에서의 dead band 이다.

C. Servo Control Shell Script.

Servo 들을 가동하기 위해 필요한 명령어들을 실행시키기 쉽도록 목적에 맞는 쉘 스크립트들을 작성하였다.

Acquire_Triple_PWM.sh

```
#!/bin/bash

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

echo bone_pwm_P9_31 > /sys/devices/bone_capemgr.8/slots
echo bone_pwm_P9_14 > /sys/devices/bone_capemgr.8/slots
echo bone_pwm_P8_19 > /sys/devices/bone_capemgr.8/slots
```

Acquire Triple shell 의 경우 앞서 진행했던 B 번 실험에서 module 을 설치하기 위해서 실행한 명령어들을 모아놓은 것이다. 이를 통해 3 개의 PWM 모듈을 한번에 설치할 수 있다.

Basically we can access PWM pins P9_31, P9_14, and P8_19 after running this code at sys/devices/ocp.3 directory.

PWM0A_Servo.sh

```
#!/bin/bash

while [ 1 ]
do
	echo "Enter period: "
	read period

	echo "Enter duty: "
	read duty

	if [ $duty -lt 0 ]
		then
			echo 0 > /sys/devices/ocp.3/pwm_test_P9_31.10/run
			exit 1;
	fi

	echo "period: $period duty: $duty"

	echo 0 > /sys/devices/ocp.3/pwm_test_P9_31.10/run

	echo $period > /sys/devices/ocp.3/pwm_test_P9_31.10/period
	echo $duty > /sys/devices/ocp.3/pwm_test_P9_31.10/duty

	echo 1 > /sys/devices/ocp.3/pwm_test_P9_31.10/run
done
```

PWM0A 에 원하는 period 와 duty 를 넣기 위한 쉘 스크립트이다. lt 비교를 사용하여 만약에 duty 값에 0 보다 작은 것이 들어온다면 구동을 멈추도록 하였고, 그 이외에는 wheel 을 일단 멈춘 후 원하는 period 와 duty 를 넣고, 다시 구동시키도록 하였다. 이 스크립트는 while 문이 있기 때문에 계속 원하는 period 와 duty 를 넣어서 테스트를 해볼 수 있게 하였다.

Release_Triple_PWMs.sh

```
#!/bin/bash

echo 0 > /sys/devices/ocp.3/pwm_test_P9_31.10/run
echo 0 > /sys/devices/ocp.3/pwm_test_P9_14.11/run
echo 0 > /sys/devices/ocp.3/pwm_test_P8_19.12/run
```

Wheel 이 구동중에 있으면 dead band 에 있는 duty 에서도 소음이 나기 때문에 PWM 신호를 한번에 차단하는 쉘 스크립트를 작성하였다.

Control_Triple_PWMs.sh

```
#!/bin/bash

while [ 1 ]
do
        echo "Select PWM: (9_31, 9_14, 8_19)"
        read PWM

        if [ $PWM = "9_31" ]; then
                path=/sys/devices/ocp.3/pwm_test_P9_31.10/
        elif [ $PWM = "9_14" ]; then
                path=/sys/devices/ocp.3/pwm_test_P9_14.11/
        else
                path=/sys/devices/ocp.3/pwm_test_P8_19.12/
        fi

        echo "Enter period: "
        read period

        echo "Enter duty: "
        read duty



        sl_run="/run"                                   (String literal for concatenation)
        sl_period="/period"
        sl_duty="/duty"

        p_run=$path$sl_run                              (concatenation)
        p_period=$path$sl_period
        p_duty=$path$sl_duty

        if [ $duty -lt 0 ]
                then
                        echo 0 > $p_run
                        exit 1;
```

```
        fi

        echo "period: $period duty: $duty"

        echo 0 > p_run

        echo $period > $p_period
        echo $duty > $p_duty
        echo 1 > $p_run
done
```

PWM0A_Servo.sh 를 세개의 wheel 모두 조절할 수 있도록 확대해서 작성한 쉘이다. 처음에 원하는 GPIO pin number 를 입력하여 원하는 PWM 을 고를 수 있도록 하였고, 그 이후에는 PWM0A_Servo.sh 과 작동이 유사하다. 각각 Servo 에 대해서 테스트를 해본 결과 잘 동작하는 것을 볼 수 있었다.

To add on, we need concatenation in this shell file because the code is dynamic. After you choose a GPIO pin (let's say you chose P9_31), you need to access the files in the directory /sys/devices/ocp.3/pwm_test_P9_31.10. To do that, I used concatenation of the directory and string literals. For run file, I concatenated the directory /sys/devices/ocp.3/pwm_test_P9_31.10 with string literal "/run", resulting in /sys/devices/ocp.3/pwm_test_P9_31.10/run. As a result, I could access all the files that I needed to control all three wheels.

D. TMR control in C

Triple_PWMs_Servo_Control.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void run(int on, int PWM) {
        FILE *file;
        if (PWM == 1) {
                file = fopen("/sys/devices/ocp.3/pwm_test_P9_31.10/run", "w");
        } else if (PWM == 2) {
                file = fopen("/sys/devices/ocp.3/pwm_test_P9_14.11/run", "w");
        } else if (PWM == 3) {
                file = fopen("/sys/devices/ocp.3/pwm_test_P8_19.12/run", "w");
        } else {
                fprintf(stderr, "Please enter a correct PWM\n");
                exit(1);
        }


        if(on) {
                fprintf(file, "%d", 1);
        } else {
                fprintf(file, "%d", 0);
        }
```

```
	fclose(file);
}

void period(int period, int PWM) {
	FILE *file;
	if (PWM == 1) {
		file = fopen("/sys/devices/ocp.3/pwm_test_P9_31.10/period", "w");
	} else if (PWM == 2) {
		file = fopen("/sys/devices/ocp.3/pwm_test_P9_14.11/period", "w");
	} else if (PWM == 3) {
		file = fopen("/sys/devices/ocp.3/pwm_test_P8_19.12/period", "w");
	} else {
		fprintf(stderr, "Please enter a correct PWM\n");
		exit(1);
	}

	if (period > 0) {
		fprintf(file, "%d", period);
	} else {
		fprintf(stderr, "Please enter a valid period\n");
	}

	fclose(file);
}

void duty(int duty, int PWM) {
	FILE *file;
	if (PWM == 1) {
		file = fopen("/sys/devices/ocp.3/pwm_test_P9_31.10/duty", "w");
	} else if (PWM == 2) {
		file = fopen("/sys/devices/ocp.3/pwm_test_P9_14.11/duty", "w");
	} else if (PWM == 3) {
		file = fopen("/sys/devices/ocp.3/pwm_test_P8_19.12/duty", "w");
	} else {
		fprintf(stderr, "Please enter a correct PWM\n");
		exit(1);
	}

	if (duty > 0) {
		fprintf(file, "%d", duty);
		fclose(file);
	} else {
		printf("done\n");
		run(0, PWM);
		fclose(file);
		exit(1);
	}
}

int main() {
	int p;
	int d;
	int PWM;


	while(1) {
		printf("Select PWM: (1: 9_31, 2: 9_14, 3: 8_19)\n");
		scanf("%d", &PWM);
```

```
                printf("Enter period: ");
                scanf("%d", &p);

                printf("Enter duty: ");
                scanf("%d", &d);

                printf("period: %d duty: %d\n", p, d);

                run(0, PWM);
                period(p, PWM);
                duty(d, PWM);
                run(1, PWM);
        }
}
```

/sys/devices/ocp.3/pwm_test_P9_31.10/과 같은 PWM test 디렉토리에서 run, period, duty 에 원하는 값을 써넣으면 작동을 한다는 것에서 착안하여 작성한 c 코드이다. We entered our duty in nanoseconds instead of percentage, which is just a design choice. 실행 루틴 자체는 C 번 실험의 Control_Triple_PWMs.sh 와 굉장히 유사하고, 다시 말해 쉘 스크립트를 C 프로그램으로 바꿔서 작성한 것으로 보아도 무방하다. However, you cannot directly access the run, duty, and period files in C program like in shell program. However, we know that everything is a file on UNIX system and Debian is a UNIX system. Therefore, I just open the files that I want to access using the fopen method with parameter "w". Parameter "w" means that we are going to write to the file that we are opening; if there is no such file, make the file; if there is such file and the file has things written in them, erase them. When we echo to a file, we are basically erasing the contents of the file and writing what we echo in to the file. For example, if we do echo 1 > run, we erase the contents in the file run and write 1 to it.

In this C program, I assume that the procedure of acquiring the triple PWMs have been done. That is, I have to run **Acquire_Triple_PWM.sh** file before I run this code. I could also implement acquiring the PWMs in this code by simply using fopen the directory /sys/devices/bone_capemgr.8/slots and writing am33xx_pwm, bone_pwm_P9_31, bone_pwm_P9_14, bone_pwm_P8_19.

So how do you actually write to the file? When you open the file using fopen, you get a file descriptor as an output. In our code, I assigned the file descriptor to the variable file. We can write to the file using fprintf method. Instead of printing on standard output stream like we normally do, we print to file with fprintf method. For example, when we do fprintf(file, "%d", 1), we are writing 1 to the file. It is exact same thing as echoing 1 to the file.

Therefore, I opened each run, duty, and period files that I need. I got the values of run, duty, period, and PWM number from standard input. And wrote the values to the appropriate file.

Let me go through a quick example. If I type 1 (P9_31) for PWM number, 3000000 for period, 2000000 for duty, I first stop the wheel by writing 0 to the run file. How to write 0 to the run file? If we look that the run function, we first check which PWM we chose. Since we chose 1 (P9_31), it opens /sys/devices/ocp.3/pwm_test_P9_31.10/run with fopen method with "w" parameter. Next, we check if we are writing 0 or 1. Since we are turning off, we are

writing 0 to the run file by using the fprintf method. For duty and period, we take the same approach.

Dead band 의 min 과 max 를 속도의 0%이라고 놓으면 속도가 linear 하게 증가한다고 가정했을 때 각각 B 번 실험에서 구한 Start 값과 End 값이 -100%와 100%가 된다. 이때 -1%와 1%에 해당하는 duty 값을 주었을 경우에도 조금씩 돌아가는 모습을 볼 수 있었다. 이는 우리가 dead band 를 측정할 때 정지 마찰력이 더 큰 것을 감안하여, 항상 Servo 를 먼저 run 0 해준 상태에서 측정을 했기 때문인데, 그렇기 때문에 1%를 해주어도 정지 마찰력보다 큰 힘이 주어지기 때문에 제대로 돌아가는 것을 볼 수 있었다. 만약에 dead band 를 Servo 를 끄지 않고 측정했다면 운동 마찰력에 의한 것으로 측정이 되기 때문에 범위가 더 좁아질 것이고, 1%, -1%구동에서 좋은 결과를 얻지 못하였을 것이다. 이때 이를 compensation 하기 위해서는 확실히 구동되는 속도로 Servo 를 움직이게 한 뒤 1% 구동을 하도록 명령을 해야 할 것이다.

E. Keyboard Control for TMR

마지막 실험은 실험의 전체적인 목표였던 Keyboard 를 이용하여 TMR 을 구동하도록 만드는 것이다. 키보드의 q, e 를 통해서 반시계와 시계방향으로 돌게하고, w, x, a, d 를 이용하여 전후좌우로 TMR 을 움직이도록 만들었으며, s 로 멈추도록하였다. 코드는 다음과 같다.

TMR_Control.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <termios.h>


/********** getche.c ***************/
static struct termios old, new;

/* Initialize new terminal i/o settings */
void initTermios(int echo)
{
   tcgetattr(0, &old); /* grab old terminal i/o settings */
   new = old; /* make new settings same as old settings */
   new.c_lflag &= ~ICANON; /* disable buffered i/o */
   new.c_lflag &= echo ? ECHO : ~ECHO; /* set echo mode */
   tcsetattr(0, TCSANOW, &new); /* use these new terminal i/o settings now */
}

/* Restore old terminal i/o settings */
void resetTermios(void)
{
   tcsetattr(0, TCSANOW, &old);
}

/* Read 1 character - echo defines echo mode */
char getch_(int echo)
{
   char ch;
```

```
    initTermios(echo);
    ch = getchar();
    resetTermios();
    return ch;
}

/* Read 1 character without echo */
char getch(void)
{
    return getch_(0);
}

/* Read 1 character with echo */
char getche(void)
{
    return getch_(1);
}

/********** getche.c ****************/

void ind_run(int on, int PWM) {
        FILE *file;
        if (PWM == 1) {
                file = fopen("/sys/devices/ocp.3/pwm_test_P9_31.10/run", "w");
        } else if (PWM == 2) {
                file = fopen("/sys/devices/ocp.3/pwm_test_P9_14.11/run", "w");
        } else if (PWM == 3) {
                file = fopen("/sys/devices/ocp.3/pwm_test_P8_19.12/run", "w");
        } else {
                fprintf(stderr, "Please enter a correct PWM\n");
                exit(1);
        }


        if(on) {
                fprintf(file, "%d", 1);
        } else {
                fprintf(file, "%d", 0);
        }
        fclose(file);
}


void period(int period, int PWM) {
        FILE *file;
        if (PWM == 1) {
                file = fopen("/sys/devices/ocp.3/pwm_test_P9_31.10/period", "w");
        } else if (PWM == 2) {
                file = fopen("/sys/devices/ocp.3/pwm_test_P9_14.11/period", "w");
        } else if (PWM == 3) {
                file = fopen("/sys/devices/ocp.3/pwm_test_P8_19.12/period", "w");
        } else {
                fprintf(stderr, "Please enter a correct PWM\n");
                exit(1);
        }

        if (period > 0) {
                fprintf(file, "%d", period);
        } else {
```

```
                fprintf(stderr, "Please enter a valid period\n");
        }

        fclose(file);
}

void duty(unsigned int duty, int PWM) {
        FILE *file;
        if (PWM == 1) {
                file = fopen("/sys/devices/ocp.3/pwm_test_P9_31.10/duty", "w");
        } else if (PWM == 2) {
                file = fopen("/sys/devices/ocp.3/pwm_test_P9_14.11/duty", "w");
        } else if (PWM == 3) {
                file = fopen("/sys/devices/ocp.3/pwm_test_P8_19.12/duty", "w");
        } else {
                fprintf(stderr, "Please enter a correct PWM\n");
                exit(1);
        }

        if (duty > 0) {
                fprintf(file, "%d", duty);
                fclose(file);
        }
}

double* vvTOwv(double* vv) {
        double *wv = (double *)malloc(3*sizeof(double));
        double r = 0.02;
        double L = 0.10;

        double T[3][3] = {{0, 1/r, L/r}, {-1.1547/r, -0.5/r, L/r}, {1.1547/r, -0.5/r, L/r}};

        double Tv = 0;
        int i, j;

        for (i = 0; i < 3; i++) {
                for (j = 0; j < 3; j++) {
                        Tv = Tv + vv[j]*T[i][j];
                }
                wv[i] = Tv;
                Tv = 0;
        }

        return wv;
}

double* stop(double* vv) {
        vv[0] = 0;
        vv[1] = 0;
        vv[2] = 0;
        return vv;
}

double* forwards(double* vv) {
        vv[0] = 1;
        vv[1] = 0;
        vv[2] = 0;
        return vv;
}
```

```
double* backwards(double* vv) {
	vv[0] = -1;
	vv[1] = 0;
	vv[2] = 0;
	return vv;
}

double* right(double* vv) {
	vv[0] = 0;
	vv[1] = 1;
	vv[2] = 0;
	return vv;
}

double* left(double* vv) {
	vv[0] = 0;
	vv[1] = -1;
	vv[2] = 0;
	return vv;
}

double* rotatecw(double* vv) {
	vv[0] = 0;
	vv[1] = 0;
	vv[2] = -1;
	return vv;
}

double* rotateccw(double* vv) {
	vv[0] = 0;
	vv[1] = 0;
	vv[2] = 1;
	return vv;
}



void quit() {
	ind_run(0, 1);
	ind_run(0, 2);
	ind_run(0, 3);
}
void run_forwards(double *wv) {
	int p=3000000;
	unsigned int duty1;
	unsigned int duty2;
	unsigned int duty3;
	unsigned int stopDuty = 1500000;

	if (wv[0] > 0) {

		duty1 = 1533400+(2999939-1533400)*wv[0]*0.01;
	} else if (wv[0] < 0) {

		duty1 = 660-(1429000-660)*wv[0]*0.01;
	} else {
		duty1 = stopDuty;
	}
```

```
		if (wv[1] > 0) {
			duty2 = 1530440+(2999999-1530440)*wv[1]*0.01;
		} else if (wv[1] < 0) {
			duty2 = 1444000+(1444000-720)*wv[1]*0.01;
		} else {
			duty2 = stopDuty;
		}

		if (wv[2] > 0) {
			duty3 = 1538700+(2999939-1538700)*wv[2]*0.01;
		} else if (wv[2] < 0) {
			duty3 = 1418600+(1418600-720)*wv[2]*0.01;
		} else {
			duty3 = stopDuty;
		}


		ind_run(0, 1);
		ind_run(0, 2);
		ind_run(0, 3);

		period(p, 1);
		period(p, 2);
		period(p, 3);

		duty(duty1, 1);
		duty(duty2, 2);
		duty(duty3, 3);

		printf("duty1: %u; ", duty1);
		printf("duty2: %u; ", duty2);
		printf("duty3: %u \n", duty3);

		ind_run(1, 1);
		ind_run(1, 2);
		ind_run(1, 3);
}
void run(double* wv) {
		int p=3000000;
		unsigned int duty1;
		unsigned int duty2;
		unsigned int duty3;
		unsigned int stopDuty = 1500000;
		int scale1 = 120;
		int scale2 = 8;
		int scale3 = 35;
		if (wv[0] > 0) {

			duty1 = 1533400+(2999939-1533400)*wv[0]*0.01-200000;
		} else if (wv[0] < 0) {

			duty1 = 660-(1429000-660)*wv[0]*0.01;
		} else {
			duty1 = stopDuty;
		}

		if (wv[1] > 0) {
			duty2 = 1530440+(2999999-1530440)*wv[1]*0.01/scale1;
```

```
		} else if (wv[1] < 0) {
			duty2 = 1444000+(1444000-720)*wv[1]*0.01/scale2-1500;
		} else {
			duty2 = stopDuty;
		}

		if (wv[2] > 0) {
			duty3 = 1538700+(2999939-1538700)*wv[2]*0.01/scale1;
		} else if (wv[2] < 0) {
			duty3 = 1418600+(1418600-720)*wv[2]*0.01/scale3;
		} else {
			duty3 = stopDuty;
		}


		ind_run(0, 1);
		ind_run(0, 2);
		ind_run(0, 3);

		period(p, 1);
		period(p, 2);
		period(p, 3);

		duty(duty1, 1);
		duty(duty2, 2);
		duty(duty3, 3);

		printf("duty1: %u; ", duty1);
		printf("duty2: %u; ", duty2);
		printf("duty3: %u \n", duty3);

		ind_run(1, 1);
		ind_run(1, 2);
		ind_run(1, 3);

}

int main() {
		char input = 0;
		double *vv;
		double *wv;

		vv = (double *)malloc(3*sizeof(double));

		FILE *file = fopen("/sys/devices/bone_capemgr.8/slots", "w");
		fprintf(file, "am33xx_pwm");
		fprintf(file, "bone_pwm_P9_31");
		fprintf(file, "bone_pwm_P9_14");
		fprintf(file, "bone_pwm_P8_19");
		fclose(file);

		while (1) {
			input = getche();

			if (input == 27 || input == 4) { //Ctrl+D = EOT (Ascii: 4)
				quit();
				printf("done\n");
				break;
			} else if (input == 'w') {
```

```
                printf("hello\n");
                vv = forwards(vv);
                wv = vvTOwv(vv);
                run_forwards(wv);
        } else if (input == 's') {
                vv = stop(vv);
                wv = vvTOwv(vv);
                run(wv);
        } else if (input == 'a') {
                vv = left(vv);
                wv = vvTOwv(vv);
                run(wv);
        } else if (input == 'd') {
                vv = right(vv);
                wv = vvTOwv(vv);
                run(wv);
        } else if (input == 'q') {
                vv = rotateccw(vv);
                wv = vvTOwv(vv);
                run_forwards(wv);
        } else if (input == 'e') {
                vv = rotatecw(vv);
                wv = vvTOwv(vv);
                run_forwards(wv);
        } else if (input == 'x') {
                vv = backwards(vv);
                wv = vvTOwv(vv);
                run_forwards(wv);
        }
    }
    free(vv);
    free(wv);

}
```

getche.c is the c program that was given in Lab 1. Bascially getche.c allows to cpature characters from standard input without waiting for enter to be pressed. This is called noncanonical input processing mode.

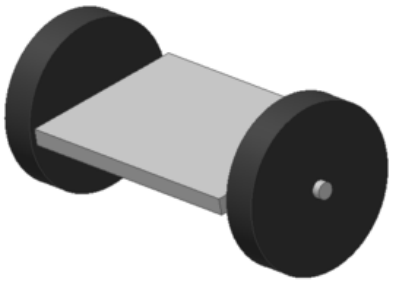
getche.c 의 경우 주어진 파일을 사용하였고, ind_run 과 period, duty 함수는 D 번에서 작성한 코드와 동일하다. vvTOwv 함수에서 vv = [vx, vy, w]'는 Cartesian linear/rotational velocity 이며 이는 각각 x 축, y 축, 회전속도를 의미한다. 여기서 x 축은 전후를 의미하고, y 축은 좌우를 의미한다. 이를 이용하여 원하는 방향에 따른 vv 값을 대입해주는 함수 stop 부터 rotateCCW 까지를 작성하였다. vv 에 각속도 전환 행렬 T 를 곱해주면 wv = [w1, w2, w3]'의 값이 나오고, 이는 Servo 1, 2, 3 의 각속도의 비를 의미한다. T 의 경우 주어진 행렬을 사용하였고, 이는 Servo 가 토크의 중심에 대해서 정삼각형을 이루고 있어서 나타나는 힘의 상관관계에서 기인한 행렬이다. 문제는 Servo 의 각속도를 계산할 때 dead band 부터 start 나 end 까지 linear 하게 증가한다고 가정했던 것이다. 만약 linear 하게 이상적으로 증가할 경우 각속도의 비를 곱했을 경우 이상적으로 전후좌우로 움직여야했으나 좌우의 움직임이 굉장히 오차가 많이 생겼다. 전후와 회전의 경우 세개의 Servo 가 (전후의 경우 두개) 동일한 속도로 움직이면 되기 때문에 linear 가정에 아무런 영향을 받지 않지만, 좌우의

경우 Servo 들의 속도비가 굉장히 중요하기 때문이다. 이를 보정하기 위해서 전후 회전은 forward 함수로, 좌우는 run 함수로 따로 작성하였고, run 함수는 wv 의 비율을 그대로 사용하지 않고 거기에 나눗셈 또는 뺄셈을 통하여 scaling 을 해주었다. Scaling 의 경우는 먼저 움직임을 보고, 토크와 힘의 방향을 확인한 후 계속해서 추가적으로 보정하였다. 이를 통해 50cm 전후좌우 이동 테스트를 한 결과, 좌우의 경우 7mm 의 오차가 났고, 전후의 경우 1mm 의 오차가 났다. Scaling 을 해주었음에도 불구하고 오차가 난 이유는 Servo 와 Beaglebone 모두 7.4V 의 배터리에 전원을 의존하다 보니 배터리를 사용하는 도중에도 계속 전압이 소모되어 낮아졌기 때문으로 생각된다.

Discussion

1) Compare mobile robots with two-wheels and three-wheels

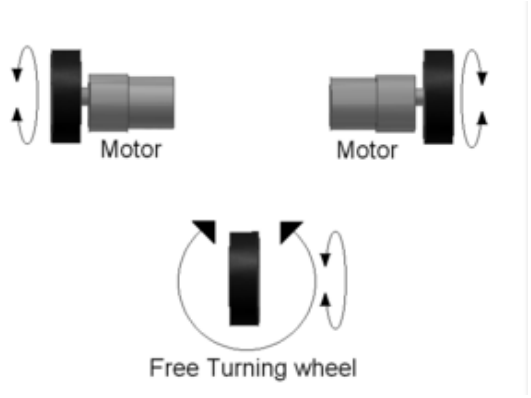
Two-wheeled robots whose wheels are parallel to each other are called dicycles and are hard to balance. Take a look at an example.



If we made our robot with two wheels, the board will swing up and down when the robot moves. A two-wheeled robot direct itself using the angular speed of each of its wheels. Therefore, when it rotates, it does not rotate with its center being the point of rotation. As a result, when it rotates, the robot does get a smooth rotation. Also, the board between the wheels keeps swinging, making the robot more unstable. Two wheeled robots can move forwards, backwards, rotate, and even move forwards and backwards when rotating. Although you do not get the precise control like three-wheeled robot, two-wheeled robot is simple and easy to control.

Examples of two-wheeled robots are robotic vacuum cleaners and 요즘유행하는 전동휠이 있다.

Three wheeled robot's wheels are positioned like this.



In this case, two wheels only rotate forwards and backwards and the front wheel can move forwards, backwards, and rotate. The front wheel could be powered. If the front wheel does not get powered. The front wheel helps balance the robot and move in any direction. The direction is determined by the angular speed of the two separately driven wheels (just like two wheeled robots). For example, if the two wheels have same the same angular speeds, the robot will go straight. However, if right wheel in the picture above has faster angular speed than the left, the robot will move to 7'o clock position.

Obviously, the three-wheeled robot is more stable than the two-wheeled robot. Also, you can get more precise movement with the three-wheeled robots. However, three-wheeled robot is more complicated and hard to control than the two-wheeled robot because you have an extra wheel to care about.

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

GPIO pin	Start (ns)	Dead band min (ns)	Dead band max (ns)	End (ns)
P9_31 (PWM0A)	660	1429000	1533460	2999939
P9_18 (PWM1A)	720	1444000	1530440	2999999
P8_19 (PWM2A)	720	1418600	1538700	2999939

Above table is the dead band that I measured for my servo motors. Servo motors have different band-bands because you cannot make servo motors exactly the same. When you make a servo, the parts that you use to make a servo aren't 100% the same.

To compensate for each wheels having different dead bands, we had to measure the dead band of the servo motors individually. Also, as I mentioned above,우리는 dead band 를 측정할 때 정지 마찰력이 더 큰 것을 감안하여, 이를 compensation 하기 위해서는 확실히 구동되는 속도로 Servo 를 움직이게 한 뒤 1% 구동을 하도록 명령을 하였다. 만약에 dead band 를 Servo 를 끄지 않고 측정했다면 운동 마찰력에 의한 것으로 측정이 되기 때문에 범위가 더 좁아질 것이고, 1%, -1%구동에서 좋은 결과를 얻지 못하였을 것이다.

Although the band-bands of the servo motors are all different, I thought that angular speed of the wheel and the pulse width of signal would have linear relationship. Therefore, I assumed

that the value between dead band max value and end value would give the highest angular speed. However, it was not the case. The relationship between the angular speed of a wheel and pulse width of a signal was not linear. As a result, we could not get the same speed for all the wheels. 만약 linear 하게 이상적으로 증가할 경우 각속도의 비를 곱했을 경우 이상적으로 전후좌우로 움직여야했으나 좌우의 움직임이 굉장히 오차가 많이 생겼다. 전후와 회전의 경우 세개의 Servo 가 (전후의 경우 두개) 동일한 속도로 움직이면 되기 때문에 linear 가정에 아무런 영향을 받지 않지만, 좌우의 경우 Servo 들의 속도비가 굉장히 중요하기 때문이다.

For this nonlinear relationship of the angular speed of the servo motors and pulse width of the signals, we had to scale our duty value for each individual servo motors. Since we do not know the exact relationship between the angular speed of the servo motors and pulse width of the signals, we had to perform trial and error to find the value for scaling the duty value. After a long trial and error, we came up with some scaling values by multiplying and adding constant values to the duties. As a result, 좌우의 경우 7mm 의 오차가 났고, 전후의 경우 1mm 의 오차가 났다. Note that we had to scale duty for left movement and right movement differently.

3) Are there any disadvantages using continuous rotation RC servo

As I explained in the previous question, it is relatively hard to control RC servo because of the differences of dead-band of RC servos and the nonlinear relationship of angular speed of the servo model's wheel and the pulse width of the signal. As a result, we have to find dead-band for each RC servo motors, which can be a quite hassle. Also, it is hard to move our robot left and right in a straight manner due to the nonlinear relationship between the angular speed of the servo model's wheel and pulse width of the signal. As a result, we had to scale our duty value to get the desired angular speed. This is a quite burden because it is very hard to find the scaling factor for the angular speed we want. Also, we have to find the scaling factor for individual RC servo motors and left movement and right movement have different scaling factors. Even if we find all the scaling factors we need, we cannot guarantee movement of left and right in a straight manner. We spent hours finding the scaling factor, but still had 7mm of observational error for the left movement. To sum up, it is pretty hard to control rotation RC servo with PWM.

When we set the value of duty of servo motors to be the value in the dead zone, we expect the servo motor to stop its activity. However, when we set the value of duty of servo motors to be the value in the dead zone, it kept making noise and consumed power. This is because when the servo motor is stopped, the motor's rotor continues to move back and forth one pulse. To sum up, another disadvantage of RC servo motor is that it consumes a lot of power. This can be detrimental to our robot because we use external batteries to run the robot. As a result, we cannot expect our robot to run for a long time using a servo motors.

Other minor disadvantages of using continuous rotation RC servo that we do not really "care" about in our robot might be that servo motor wears out; it runs away when something breaks; it is complex; it can be damaged by overload; it offers poor cooling.

References

DesignLab_RoboCam_Lab3_E.pdf

DesignLab_RoboCam_Lecture3_E.pdf

https://en.wikibooks.org/wiki/Robotics/Types_of_Robots/Wheeled

http://rtcl.kaist.ac.kr/~bkkim/lecture/designlab/Datasheet/Lab3_TMR/Parallex_Continuous-Rotation-Servo-Documentation-v2.2.pdf

http://rtcl.kaist.ac.kr/~bkkim/lecture/designlab/Datasheet/Lab3_TMR/Omniwheel_Drawing_ver2.pdf