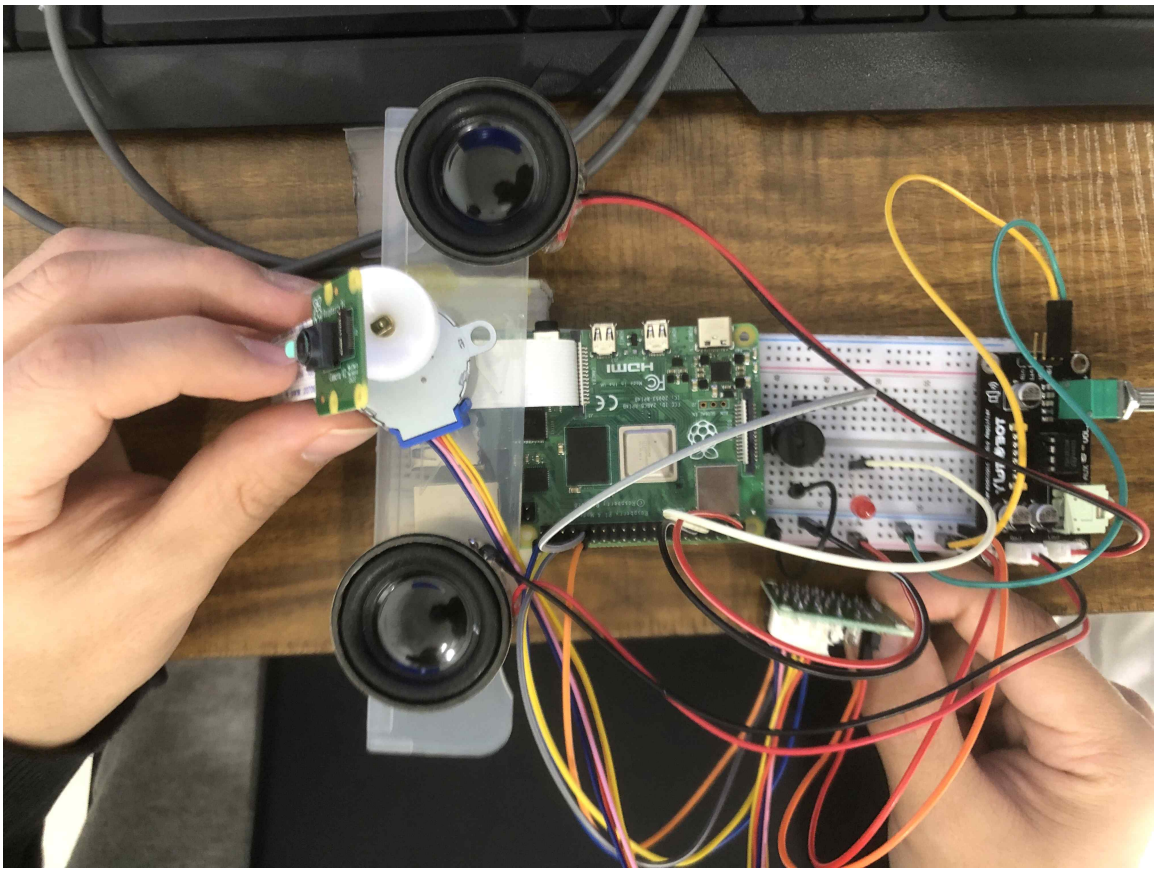
〈8조〉

# 임베디드 시스템
# 〈텀프로젝트〉

담당교수 : 조용범
실험날짜 : 2023.11.09
조 : 8 조
조원 :202114134 김준석
202114160 채민기
202114149 이기훈

# 1. 작품명

ai 지능형 카메라



# 2. 조원

202114134 김준석
202114149 이기훈
202114160 채민기

# 3. 요약

카메라로 비춰진 영상을 통해 모터 및 부저를 제어하고 특정 모션을 인식해 음악파일을 재생/멈춤, led의 on/off를 제어한다.

# 4. 배경지식

## 디바이스 드라이버 - 장치의 사용법이 적혀있는 실행파일

copy-out방식 : 커널 모듈에서 사용자 공간으로 데이터를 복사하는 메모리 전송 방식

1. 커널 모듈에서 사용자 공간에 데이터를 전송해야할 때, 데이터를 사용자 영역의 버퍼로 복사합니다.
2. 커널모듈은 'copy_to_user' 함수를 사용하여 사용자 공간의 메모리로 데이터를 복사합니다.
3. 이렇게 복사된 데이터는 사용자 프로세스에 직접 접근이 가능하며, 사용자 영역과 커널 영역의 데이터가 분리됩니다.

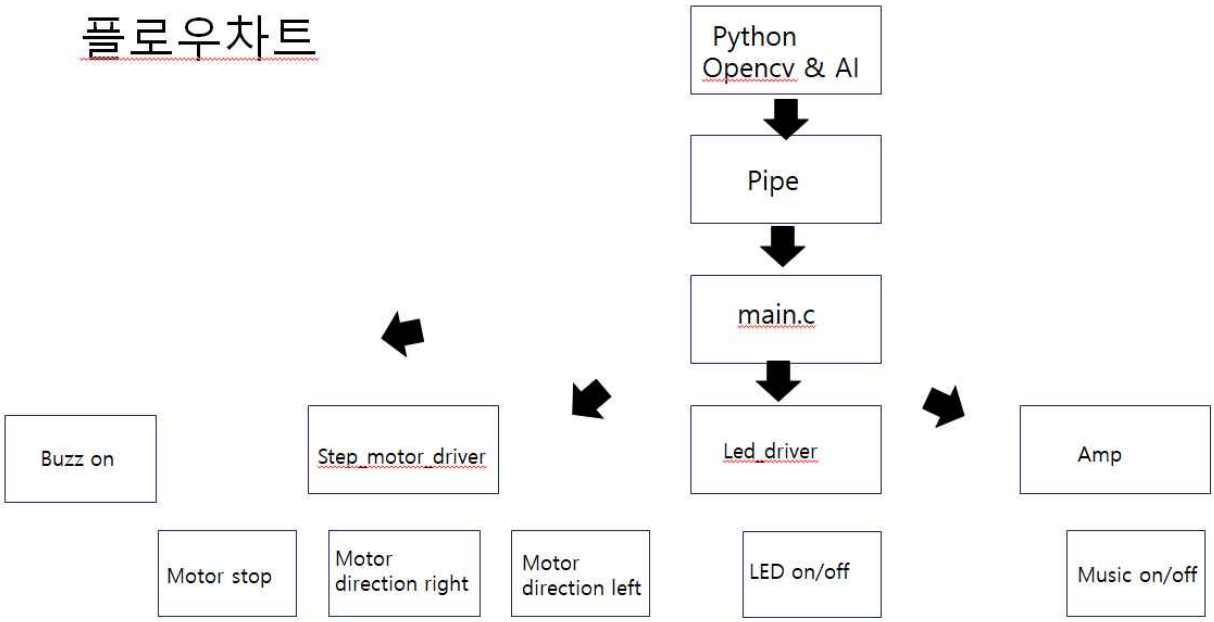
## 파이프라인 - 여러 작업이 연속적으로 이어져 처리되는 구조

이번 텀 프로젝트에서 파이프라인은 직접 찍은 사진을 데이터셋으로 한 AI 학습이 완료된 파이썬 파일을 C 언어로 되어 있는 실행파일에 적재할때에 사용하였습니다. 코드예시는 다음과 같습니다. python <파이썬 실행파일.py> | ./<c 실행파일>

## opencv - 영상 처리에 사용할 수 있는 오픈소스 라이브러리

cv2.VideoCapture : 비디오 캡처(카메라 스트림)을 위한 인터페이스를 제공
cv2.resize : 이미지 크기 조정
cv2.imshow : 이미지를 창에 표시하는 함수
cv2.waitkey : 키보드 입력을 대기하게 하여 사용자 입력처리를 가능하게함

# 5.




# 6. 소스코드

## 6-1. 사용한 TRAIN CODE

### IMAGE CLASSIFICATION

classification.ipynb - Colaboratory (google.com)

# 6-2. 실행파일

# 파이썬코드

```python
import tflite_runtime.interpreter as tflite
#openCV를 사용하기위해 import
import cv2
import numpy as np
import time
import os

# 파이프 파일 경로
pipe_name = '/tmp/my_pipe'

# 파이프가 존재하지 않으면 생성
if not os.path.exists(pipe_name):
    os.mkfifo(pipe_name)

# Disable scientific notation for clarity
np.set_printoptions(suppress=True)

# Load the TensorFlow Lite model
tflite_interpreter = tflite.Interpreter(model_path="image_classify.tflite")
tflite_interpreter.allocate_tensors()

# Get input and output tensors
input_details = tflite_interpreter.get_input_details()
output_details = tflite_interpreter.get_output_details()

# Load the labels
class_names = open("labels.txt", "r").readlines()

# CAMERA can be 0 or 1 based on default camera of your computer
camera = cv2.VideoCapture(0)

# 파이프를 쓰기 모드로 열기
with open(pipe_name, 'w') as pipe:
    while True:
        # Grab the webcamera's image
        ret, image = camera.read()

        # Resize the raw image into (224-height,224-width) pixels
        image = cv2.resize(image, (224, 224), interpolation=cv2.INTER_AREA)

        # Show the image in a window
        cv2.imshow("Webcam Image", image)

        # Make the image a numpy array and reshape it to the model's input shape
        image = np.asarray(image, dtype=np.float32).reshape(1, 224, 224, 3)

        # Normalize the image array
        image = (image / 127.5) - 1

        # Set the value of the input tensor
        tflite_interpreter.set_tensor(input_details[0]['index'], image)

        # Run the model
        tflite_interpreter.invoke()

        # Get the prediction
        prediction = tflite_interpreter.get_tensor(output_details[0]['index'])
        index = np.argmax(prediction)
        class_name = class_names[index]
        confidence_score = prediction[0][index]
        # main파일인 c실행파일에 데이터를 넘겨주기 전의 전처리 과정
        #75점 이상의 경우에서만 데이터를 전송해준다.
        if (confidence_score *100)>=75:
            pipe_data = f"{class_name[2]}"
            pipe.write(pipe_data)
            pipe.flush()

       # time.sleep(1)

        # Listen to the keyboard for presses
        keyboard_input = cv2.waitKey(1)

        # 27 is the ASCII for the esc key on your keyboard
        if keyboard_input == 27:
            break

camera.release()
cv2.destroyAllWindows()
```

# c코드

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>

#define PIPE "/tmp/my_pipe"
#define BUFFER_SIZE 100// pipeline broken error를 방지하기 위해 100정도로 사이즈 설정

int main() {
    int pipe_fd;
    char buffer[BUFFER_SIZE];
    int isPlaying = 0;  // 음악 재생 상태 추적

    //디바이스 드라이버가 열리는지 확인
    int dev1 = open("/dev/motor_driver", O_RDWR);// step motor
    int dev2 = open("/dev/my_LED", O_RDWR);// led device
    if (dev1 == -1) {
        printf("Opening was not possible!\n");
        return -1;
    }
    if (dev2 == -1) {
        printf("Opening was not possible!\n");
        return -1;
    }
    printf("device opening was successful!\n");

    pipe_fd = open(PIPE, O_RDONLY);
    if (pipe_fd == -1) {
        perror("open");
        return EXIT_FAILURE;
    }
    // 파이프 라인을 통해 넘어온 데이터를 읽어옴
    while (1) {
        ssize_t bytes_read = read(pipe_fd, &buffer, BUFFER_SIZE);  // 단일 문자 읽기
        if (bytes_read > 0) {
            printf("Received: %c\n", buffer[0]);  // 단일 문자 출력

             if(buffer[0]=='S'){//motor stop and buzz on
               printf("Stop\n");
               write(dev1,&buffer[0],1);
               usleep(100);
            }
            else if(buffer[0]=='R'){
                printf("Right\n");
                write(dev1,&buffer[0],1);
                usleep(100);
            }
            else if(buffer[0]== 'L'){
                printf("Left\n");
                write(dev1,&buffer[0],1);
                usleep(100);
            }
            else if (buffer[0] == 'A' && !isPlaying){
                // 'A' 입력 시, 음악이 정지 상태이면 재생 시작
                system("aplay taeyeon.wav &");  // 음악 재생
                isPlaying = 1;
            }
            else if (buffer[0] == 'F' && isPlaying){
                // 'F' 입력 시, 음악이 재생 상태이면 정지
                system("pkill aplay");  // 음악 정지
                isPlaying = 0;
            }
            else if(buffer[0]== 'B'){//for controll led light bulb
                printf("led toggle\n");
                write(dev2,&buffer[0],1);
            }
            else
                //예외 처리된 값이 넘어오면 아무동작도 하지 않는다.
        } else {
            perror("read");
            close(pipe_fd);
            return EXIT_FAILURE;
        }
    }
    close(dev2);// 디바이스 해제
    close(dev1);
    close(pipe_fd);
    return EXIT_SUCCESS;
}
```

# 6-3. 스텝모터 드라이버

## <소스코드>

```c
#include <linux/module.h>
#include <linux/init.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/uaccess.h>
#include <linux/gpio.h>
#include <linux/delay.h>

/* Meta Information */
MODULE_LICENSE("GPL");
MODULE_AUTHOR("team8");
MODULE_DESCRIPTION("STEP_Motor Driver");

/* Variables for device and device class */
static dev_t my_device_nr;
static struct class *my_class;
static struct cdev my_device;
static int delay_time = 1;// 모터를 제어하기 위한 변수
static int default_time = 10;//모터를 제어하기 위한 변수

#define DRIVER_NAME "motor_driver"
#define DRIVER_CLASS "MotorModuleClass"
//gpio 핀의 값을 정해주는 함수
void set_value(int p1, int p2, int p3, int p4) {
    gpio_set_value(12, p1);
    gpio_set_value(16, p2);
    gpio_set_value(20, p3);
    gpio_set_value(21, p4);
}

static ssize_t driver_write(struct file *File, const char *user_buffer, size_t count, loff_t *offs) {
   int to_copy, not_copied, delta, i;
   char value;

   /* Get amount of data to copy */
   to_copy = min(count, sizeof(value));

  /* Copy data to user */
  not_copied = copy_from_user(&value, user_buffer, to_copy);


   switch(value){///사용자영역에서 S,R,L을 받아와서 모터의 방향을 제어
   //turn the motor right direction
       case 'R':
           for(i = 0; i < default_time; i++) {
               set_value(1, 1, 0, 0);
               msleep(delay_time);
               set_value(0, 1, 1, 0);
               msleep(delay_time);
               set_value(0, 0, 1, 1);
               msleep(delay_time);
               set_value(1, 0, 0, 1);
               msleep(delay_time);
           }
           break;
   //turn the motor left direction
       case 'L':
           for(i = default_time; i > 0; i--) {
               set_value(1, 0, 0, 1);
               msleep(delay_time);
               set_value(0, 0, 1, 1);
               msleep(delay_time);
               set_value(0, 1, 1, 0);
               msleep(delay_time);
               set_value(1, 1, 0, 0);
               msleep(delay_time);
           }
           break;
       case 'S':
           set_value(0, 0, 0, 0);
           gpio_set_value(6,1);// 멈춤을 알리는 부저on
           msleep(100);
           gpio_set_value(6,0);
```

```
            break;
        default:
            set_value(0, 0, 0, 0);
            break;
    }
    delta = to_copy - not_copied;
    return delta;
}

/**
 * @brief This function is called, when the device file is opened
 */
static int driver_open(struct inode *device_file, struct file *instance) {
   printk("step_motor - open was called!\n");
   return 0;
}

/**
 * @brief This function is called, when the device file is opened
 */
static int driver_close(struct inode *device_file, struct file *instance) {
   printk("step_motor - close was called!\n");
   return 0;
}

static struct file_operations fops = {
   .owner = THIS_MODULE,
   .open = driver_open,
   .release = driver_close,
   .write = driver_write
};

/**
 * @brief This function is called, when the module is loaded into the kernel
 */
 static struct file_operations fops = {
    .owner = THIS_MODULE,
    .open = driver_open,
    .release = driver_close,
    .write = driver_write
 };

 /**
  * @brief This function is called, when the module is loaded into the kernel
  */
 static int __init ModuleInit(void) {
    printk("Hello, Motor_Driver!\n");

    /* Allocate a device nr */
    if( alloc_chrdev_region(&my_device_nr, 0, 1, DRIVER_NAME) < 0) {
       printk("Device Nr. could not be allocated!\n");
       return -1;
    }
    printk("read_write - Device Nr. Major: %d, Minor: %d was registered!\n", my_device_nr >> 20, my_device_nr && 0xfffff);

    /* Create device class */
    if((my_class = class_create(THIS_MODULE, DRIVER_CLASS)) == NULL) {
       printk("Device class can not e created!\n");
       goto ClassError;
    }

    /* create device file */
    if(device_create(my_class, NULL, my_device_nr, NULL, DRIVER_NAME) == NULL) {
       printk("Can not create device file!\n");
       goto FileError;
    }

    /* Initialize device file */
    cdev_init(&my_device, &fops);

    /* Regisering device to kernel */
    if(cdev_add(&my_device, my_device_nr, 1) == -1) {
       printk("Registering of device to kernel failed!\n");
       goto AddError;
    }

     /* Set GPIO 12 */
     if(gpio_request(12, "rpi-gpio-12")) {
        printk("Can not allocate GPIO 12\n");
        goto AddError;
     }

     if(gpio_direction_output(12, 0)) {
        printk("Can not allocate GPIO 12\n");
        goto Gpio12Error;
     }

```

```
    /* Set GPIO 16 */
    if(gpio_request(16, "rpi-gpio-16")) {
        printk("Can not allocate GPIO 16\n");
      goto AddError;
    }

    if(gpio_direction_output(16, 0)) {
        printk("Can not allocate GPIO 16\n");
        goto Gpio16Error;
    }

    /* Set GPIO 20 */
    if(gpio_request(20, "rpi-gpio-20")) {
        printk("Can not allocate GPIO 20\n");
      goto AddError;
    }

    if(gpio_direction_output(20, 0)) {
        printk("Can not allocate GPIO 20\n");
        goto Gpio20Error;
    }

    /* Set GPIO 21 */
    if(gpio_request(21, "rpi-gpio-21")) {
        printk("Can not allocate GPIO 21\n");
      goto AddError;
    }

    if(gpio_direction_output(21, 0)) {
        printk("Can not allocate GPIO 21\n");
        goto Gpio21Error;
    }
     /* Set GPIO 6 */
     if(gpio_request(6, "rpi-gpio-6")) {
         printk("Can not allocate GPIO 21\n");
       goto AddError;
     }

     if(gpio_direction_output(6, 0)) {
         printk("Can not allocate GPIO 6\n");
         goto Gpio6Error;
     }
     return 0;

Gpio12Error:
    gpio_free(12);
Gpio16Error:
    gpio_free(16);
Gpio20Error:
    gpio_free(20);
Gpio21Error:
    gpio_free(21);
Gpio6Error:
    gpio_free(6);
AddError:
    device_destroy(my_class, my_device_nr);
FileError:
   class_destroy(my_class);
ClassError:
   unregister_chrdev_region(my_device_nr, 1);
   return -1;
}

/**
 * @brief This function is called, when the module is removed from the kernel
 */
static void __exit ModuleExit(void) {
   gpio_set_value(12, 0);
   gpio_set_value(16, 0);
   gpio_set_value(20, 0);
   gpio_set_value(21, 0);
   gpio_set_value(6, 0);
   gpio_free(6);
   gpio_free(12);
   gpio_free(16);
   gpio_free(20);
   gpio_free(21);

    cdev_del(&my_device);
   device_destroy(my_class, my_device_nr);
   class_destroy(my_class);
   unregister_chrdev_region(my_device_nr, 1);
   printk("Goodbye, Motor_driver\n");
}

module_init(ModuleInit);
module_exit(ModuleExit);
```

# 6-4. led 드라이버

## <소스코드>

```c
#include <linux/module.h>
#include <linux/init.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/uaccess.h>
#include <linux/gpio.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("team8");
MODULE_DESCRIPTION("setting a LED and reading a motion");

static dev_t my_device_nr;
static struct class *my_class;
static struct cdev my_device;

#define DRIVER_NAME "my_LED"//THIS is LED DRIVER
#define DRIVER_CLASS "MyModuleClass"


static ssize_t driver_write(struct file *File, const char *user_buffer, size_t count, loff_t *offs) {
    int to_copy, not_copied, delta;
    static int last_state = 0;

    char value;

    to_copy = min(count, sizeof(value));

    not_copied = copy_from_user(&value, user_buffer, to_copy);

    if (value == 'B') {// copy out을 통해 유저공간에서 B를 입력받으면  LED 제어
        if (last_state == 0) {
            gpio_set_value(4, 1);//LED ON
            last_state = 1;
        } else if (last_state == 1) {//LED OFF
            gpio_set_value(4, 0);
            last_state = 0; //LED toggle스위치로 사용
        }
    } else {
        printk("Invalid Input!\n");
    }

    delta = to_copy - not_copied;
    return delta;
}


static int driver_open(struct inode *device_file, struct file *instance) {
    printk("led_button - open was called!\n");
    return 0;
}

static int driver_close(struct inode *device_file, struct file *instance) {
    printk("led_button - close was called!\n");
    return 0;
}
// read기능은 미사용
static struct file_operations fops = {
    .owner = THIS_MODULE,
    .open = driver_open,
    .release = driver_close,
    //.read = driver_read,
    .write = driver_write
};
// 모듈 초기화 과정
static int __init ModuleInit(void) {
    printk("Hello, LED_Kernel!\n");

    if(alloc_chrdev_region(&my_device_nr, 0, 1, DRIVER_NAME) < 0) {
        printk("Device Nr. could not be allocated!\n");
        return -1;
    }
    printk("read_write - Device Nr. Major: %d, Minor: %d was registered!\n", my_device_nr >> 20, my_device_nr && 0xfffff);

```

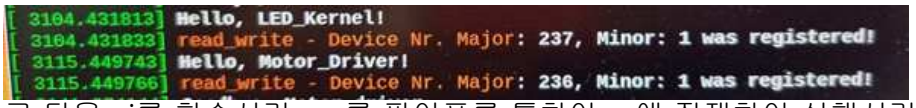
```
74      if((my_class = class_create(THIS_MODULE, DRIVER_CLASS)) == NULL) {
75          printk("Device class can not e created!\n");
76          goto ClassError;
77      }
78
79      if(device_create(my_class, NULL, my_device_nr, NULL, DRIVER_NAME) == NULL) {
80          printk("Can not create device file!\n");
81          goto FileError;
82      }
83
84      cdev_init(&my_device, &fops);
85
86      if(cdev_add(&my_device, my_device_nr, 1) == -1) {
87          printk("Registering of device to kernel failed!\n");
88          goto AddError;
89      }
90
91      if(gpio_request(4, "rpi-gpio-4")) {
92          printk("Can not allocate GPIO 4\n");
93          goto AddError;
94      }
95
96      if(gpio_direction_output(4, 0)) {
97          printk("Can not set GPIO 4 to output!\n");
98          goto Gpio4Error;
99      }
100
101
102     return 0;
103 Gpio4Error:
104     gpio_free(4);
105 AddError:
106     device_destroy(my_class, my_device_nr);
107 FileError:
108     class_destroy(my_class);
109 ClassError:
110     unregister_chrdev_region(my_device_nr, 1);
111     return -1;
112 }
113 //모듈 제거시 할당된 gpio 해제
114 static void __exit ModuleExit(void) {
115     gpio_set_value(4, 0);
116     gpio_free(4);
117     cdev_del(&my_device);
118     device_destroy(my_class, my_device_nr);
119     class_destroy(my_class);
120     unregister_chrdev_region(my_device_nr, 1);
121     printk("Goodbye,LED_Kernel\n");
122 }
123
124 module_init(ModuleInit);
125 module_exit(ModuleExit);
```

# 6-5. label map

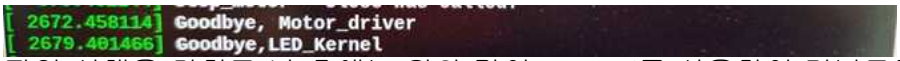
```
0 S
1 L
2 R
3 A
4 F
5 B
6 D
```

# 7. 분석 및 결론

실행파일을 실행하기에 앞서 insmod로 커널에 모듈을 적재한다.




그 다음 ai를 학습시킨 .py를 파이프를 통하여 .c에 적재하여 실행시키고




파일 실행을 마치고 난 후에는 위와 같이 rmmod를 사용하여 커널모듈을 제거한다.

모터 디바이스 드라이버를 만들때에 스텝모터의 원리에 입각하여 오른쪽/왼쪽으로 회전하는 코드를 구성하였다.

| | | | | |
|---|---|---|---|---|
| A | 1 | 0 | 0 | 0 |
| B | 0 | 1 | 0 | 0 |
| A- | 0 | 0 | 1 | 0 |
| B- | 0 | 0 | 0 | 1 |

<오른쪽 회전>

led 디바이스 드라이버에서는 copy out을 통해 유저공간에서 B를 입력받으면 last_state를 인수로 설정하여 0과 1일 때 각각 LED를 on/off 하는 toggle 스위치로 사용하였다.
부저와 앰프는 입출력 장치로 사용하였고 이를 실행파일에 적재하였다.

# 8. 주석

https://studium-anywhere.tistory.com/22

https://coding-potato.tistory.com/20

https://hoohaha.tistory.com/80