로봇, 모두의 곁에 a robot for all

ROBOT

INVESTOR RELATIONS

INDEX

https://github.com/arirang2067/drone_tutorial
비행로봇 수업자료와 예제코드를 받을 수 있습니다.

pigpio 라이브러리

pigpio는 GPIO pin들을 제어할 수 있는 라즈베리파이(RPI)용 라이브러리이다.

pigpiod라는 데몬 형태로 사용할 수 있으며 로우레벨 엑세스를 지원함.
// 데몬(daemon) : 사용자가 직접적으로 제어하지 않고 백그라운드에서 돌면서 여러 작업을 하는 프로그램.
// 로우레벨 : 기계어 및 하드웨어에 가까운 단계를 의미함.

로우레벨 제어로 GPIO 자체에 대한 정교한 조작으로 Software PWM을 쉽게 이용 가능함.
// PWM은 펄스 폭 변조로 추후 설명.

pigpio 설치하기

http://abyz.me.uk/rpi/pigpio/download.html 홈페이지 참조.

$ wget https://github.com/joan2937/pigpio/archive/master.zip
// wget : 웹에서 파일을 다운로드 할수 있는 리눅스 명령어.

$ unzip master.zip
// unzip : zip파일 압축을 푸는 리눅스 명령어.

$ cd pigpio-master
// pigpio-master로 이동한다.

$ make
// makefile에 기술된 Shell 명령어들을 순차적으로 실행하여 컴파일을 수행하는 리눅스 명령어.

$ sudo make install
// make로 만들어진 설치파일을 설치하는 리눅스 명령어

pigpio 실행 및 테스트하기

$ sudo pigpiod
// pigpio 데몬(pigpiod)을 실행한다. → 모터 노드를 실행하기 전에 1회 실행해 줄 것.

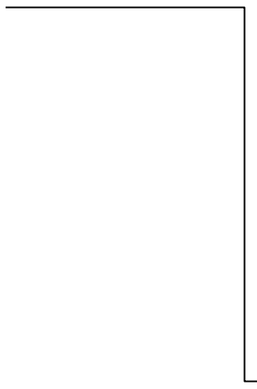
$ sudo ./x_pigpiod_if2
// pigpio 설치시 제대로 설치되었는지 확인한다.

$ sudo killall pigpiod
//pigpiod가 비정상일시, 멈추고 다시 시작할 때 사용한다.

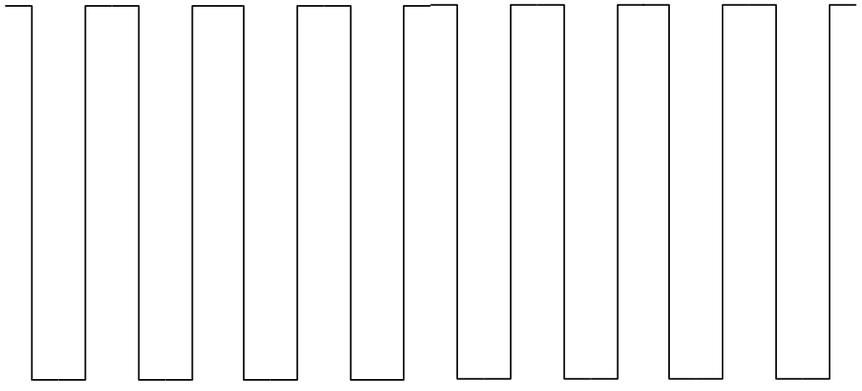
디지털 신호로 모터의 출력을 제어하는 방법

0과 1로 구성된 디지털 신호로 어떻게 모터의 출력을 제어할 수 있을까?

0은 모터가 꺼짐. 1은 모터가 켜짐.



이것을 매우 짧은 시간동안 계속 반복한다면 어떻게 될까?

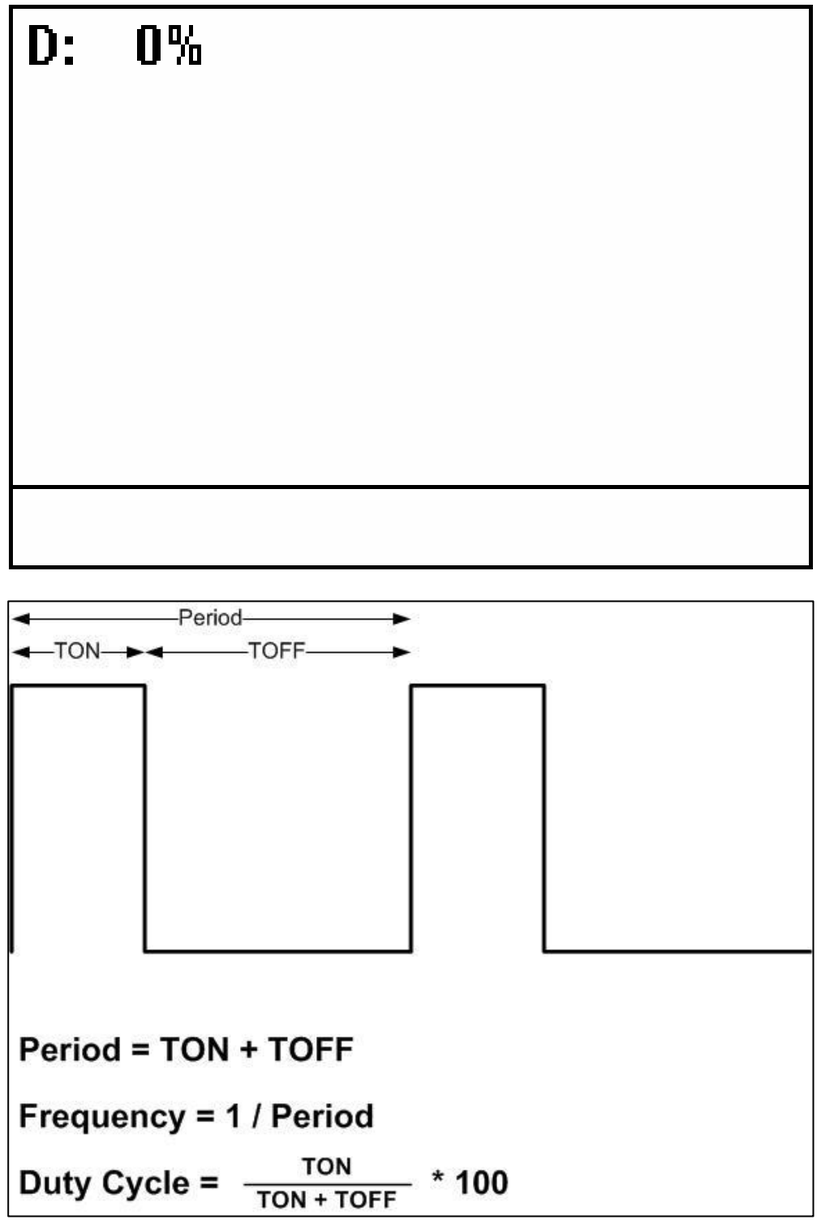


PWM (펄스 폭 변조, Pulse Width Modulation)

그리고 그 신호의 폭을 조절한다면?
→ 이 제어 방법을 PWM이라고 한다.

PWM의 구성은 아래와 같다.

TON : 제어 대상의 출력이 1인 구간.
TOFF : 제어 대상의 출력이 0인 구간.
Period : TON과 TOFF를 나누는 주기.
Frequency : 단위 시간 동안의 주기 빈도. (주파수)
Duty Cycle : 주기 중 TON의 비율. (출력의 퍼센트)



pwm_test.cpp 둘러보기

모터가 PWM 출력이 제대로 되는지 확인해보는 코드

CheckPigpio 함수 : pigpio 데몬과 연결을 시도하고 연결 여부를 점검하는 함수

InitPwm 함수 : GPIO 핀 모드와 PWM의 dutycycle, frequency를 초기화하는 함수

SetPwmDutycycle 함수 : 모든 모터에 PWM 출력을 활성화하는 함수

main 함수 : pigpio 데몬과 연결하고 모든 모터를 10% dutycycle로 출력함

pwm_test.cpp 둘러보기

모터가 PWM 출력이 제대로 되는지 확인해보는 코드

CheckPigpio 함수 : pigpio 데몬과 연결을 시도하고 연결 여부를 점검하는 함수

InitPwm 함수 : GPIO 핀 모드와 PWM의 dutycycle, frequency를 초기화하는 함수

SetPwmDutycycle 함수 : 모든 모터에 PWM 출력을 활성화하는 함수

main 함수 : pigpio 데몬과 연결하고 모든 모터를 10% dutycycle로 출력함

라이브러리 및 상수, 변수 설정

```
#include <ros/ros.h>
#include <pigpiod_if2.h>

const int k_control_cycle = 10;
const int k_pins = 8;
const int k_pwm_range = 1000;
const int k_pwn_frequency = 20000;
int pin_nums[k_pins] = {17,4,22,27,6,5,19,13};
int pi_num;
```

CheckPigpio 함수

```
bool CheckPigpio()
{
  pi_num = pigpio_start(NULL, NULL);

  if (pi_num < 0)
  {
    ROS_ERROR("PI number is %d",pi_num);
    ROS_ERROR("PIGPIO connection failed.");
    return false;
  }

  ROS_INFO("Setup Finished.");
  return true;
}
```

InitPwm 함수

```
void InitPwm()
{
  for (int i = 0; i < k_pins; i++)
  {
    set_mode(pi_num, pin_nums[i], PI_OUTPUT);
    set_PWM_range(pi_num, pin_nums[i], k_pwm_range);
    set_PWM_frequency(pi_num, pin_nums[i], k_pwn_frequency);
  }
}
```

SetPwmDutycycle 함수

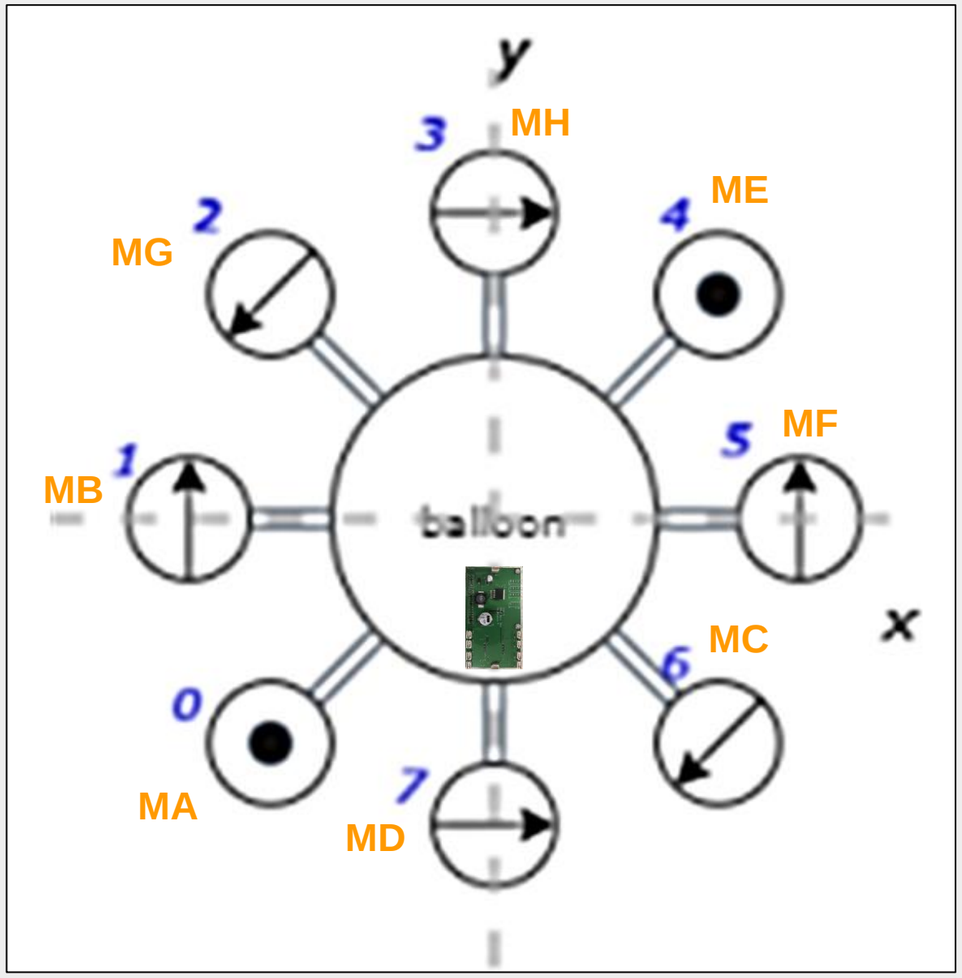
```
void SetPwmDutycycle(int rate)
{
  if(rate < 0 || rate > k_pwm_range)
  {
    ROS_WARN("Invalid Dutycycle.");
    for (int i = 0; i < k_pins; i++) set_PWM_dutycycle(pi_num, pin_nums[i], 0);
  }

  for (int i = 0; i < k_pins; i++) set_PWM_dutycycle(pi_num, pin_nums[i], rate);
}
```

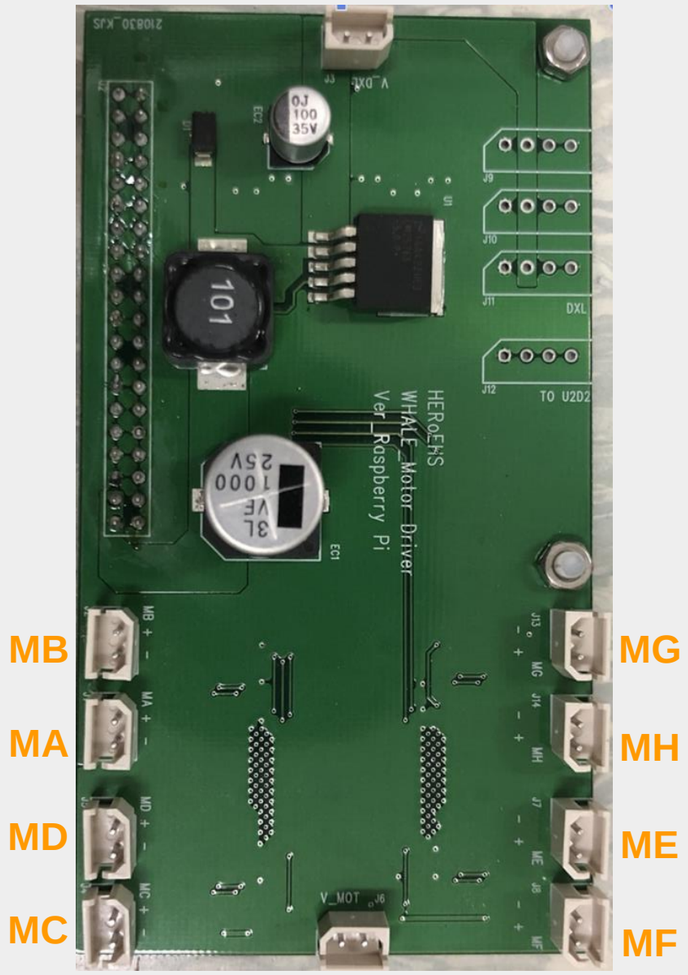
main 함수

```
int main(int argc, char **argv)
{
  ros::init(argc, argv, "pwm_test");
  ros::NodeHandle nh;
  ros::Rate loop_rate(k_control_cycle);

  if(!CheckPigpio()) return -1;
  InitPwm();

  while (ros::ok())
  {
    SetPwmDutycycle(100);
    loop_rate.sleep();
    ros::spinOnce();
  }

  SetPwmDutycycle(0);
  return 0;
}
```

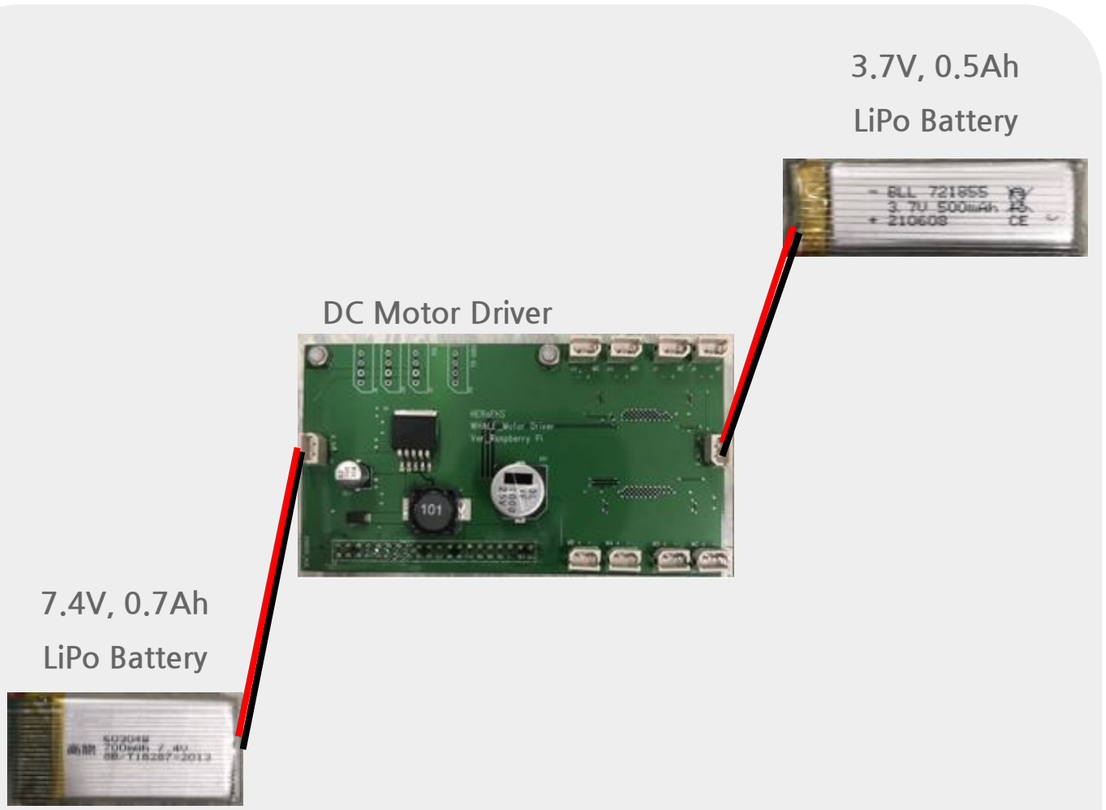
비행로봇 모터 배선

프로펠러의 방향을 잘 보고 모터선을 모터 드라이버에 연결하자.



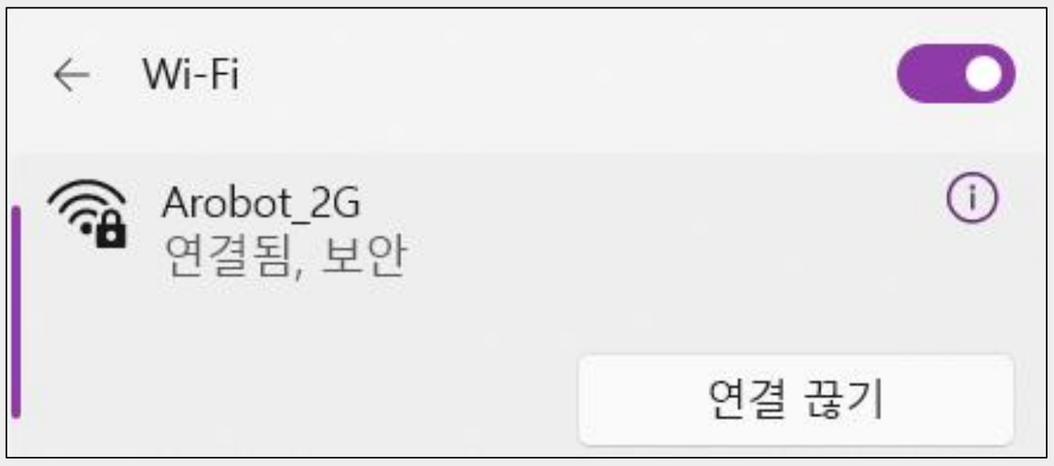


〈풍선형 비행 로봇의 프로펠러 방향(좌), 비행 로봇용 모터 드라이버 보드(우)〉



비행로봇의 배터리 연결

- 7.4V, 0.7Ah 배터리는 RPI 전원
- 3.7V 0.5Ah 배터리는 DC모터 전원
- 안전상 RPI 배터리를 먼저 꽂고 몇 초 뒤에 DC모터 배터리를 꽂을 것.
- RPI가 부팅하면서 초기화되지 않은 GPIO 핀의 영향으로 모터가 움직일 수 있음.



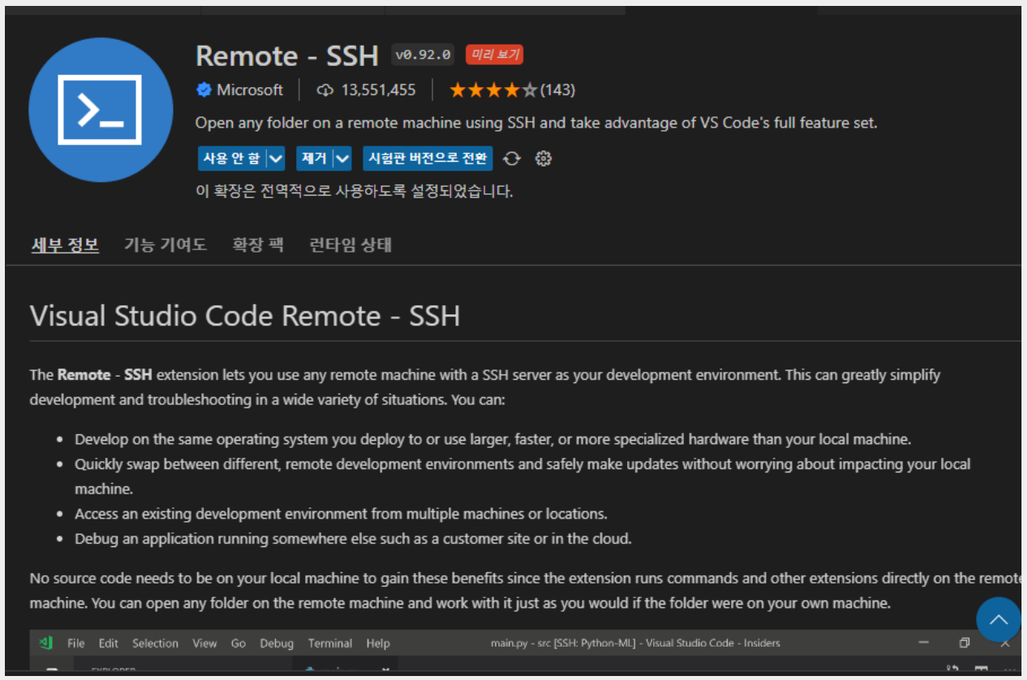
내부 네트워크로 연결

- 로컬 PC와 RPI를 같은 네트워크에 연결하기 위한 작업
- 로컬 PC에서 Arobot_2G / humansociety1!로 Wi-Fi를 잡을 것. (인터넷은 지원하지 않음.)
- RPI는 자동으로 연결되게 끔 설정되어 있음.
- RPI가 연결되지 않으면 RPI에 직접 접속하여 설정을 수정해야 함.

로컬 PC 코드 이동하기

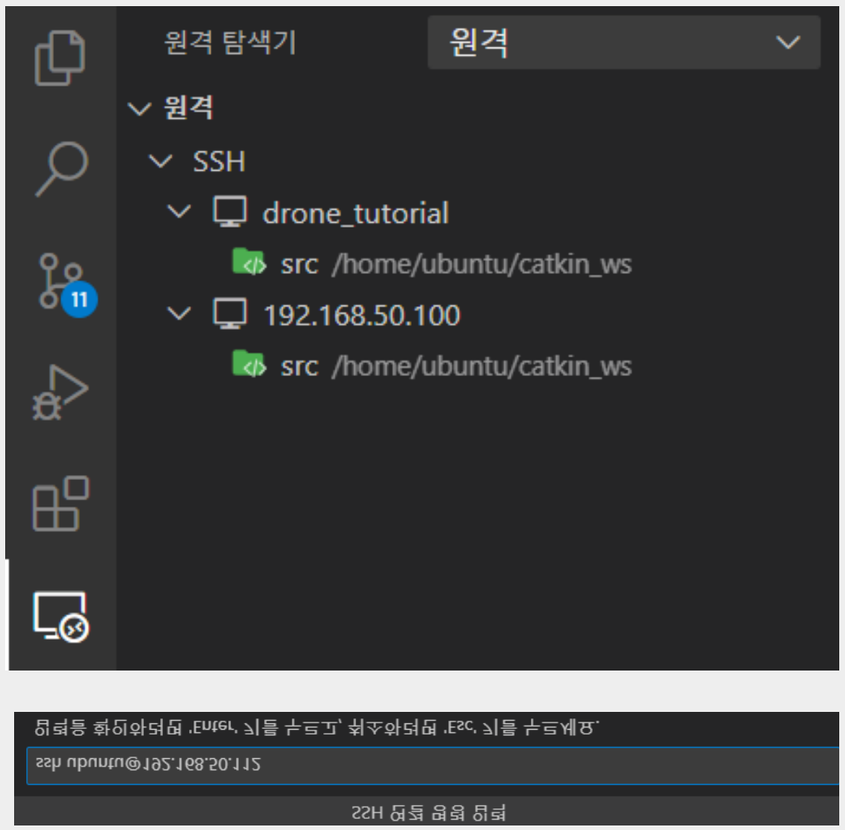
$ scp -r ~/catkin_ws/src/drone_tutorial/ ubuntu@192.168.50.XXX:~/catkin_ws/src

// scp : 현재 PC의 파일을 원격 PC로 복사한다. - r은 디렉토리 째 보내는 경우 사용하는 옵션



Remote - SSH의 설치

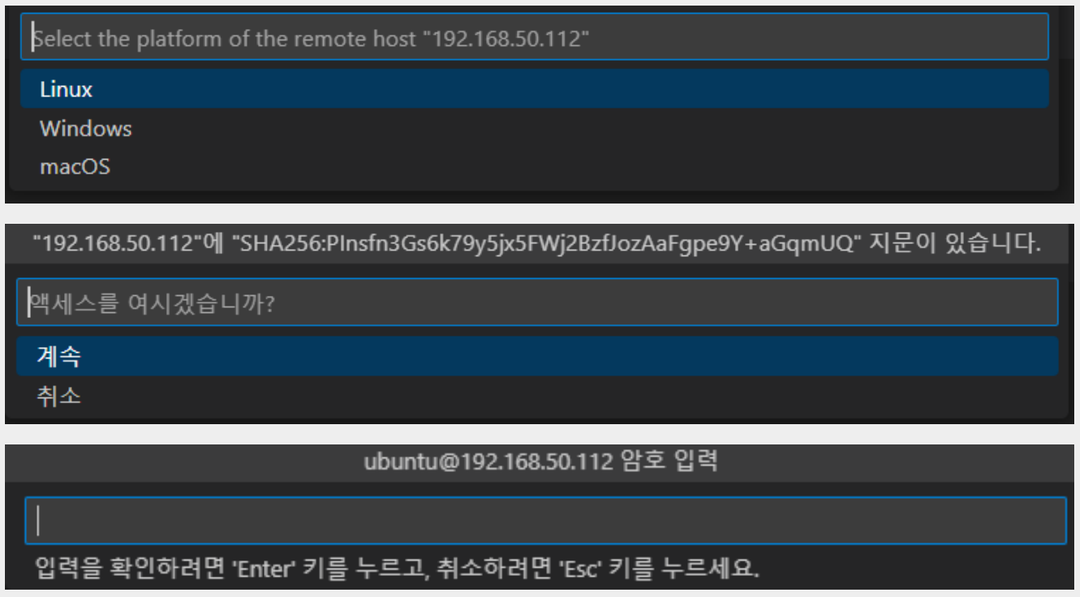
- Visual Studio Code(VS Code)에서 같은 네트워크 상의 외부기기에 원격 접속할 수 있는 도구
- VS Code의 확장(Ctrl + Shift + X)에서 ‘Remote - SSH’를 검색하여 설치



Remote - SSH를 통한 RPI 접속(1)

- VS Code 왼쪽 탭에 원격탐색기를 클릭
- 상단 탭을 '원격'으로 변경
- SSH 우측의 '+ 새 원격' 버튼을 눌러서 새 원격 추가
- 연결 명령 'ssh ubuntu@192.168.50.XXX' 입력
- 원격 탭 새로 고침
- SSH 탭에서 아이피를 찾아 '새 창에서 연결' 클릭

Remote - SSH를 통한 RPI 접속(2)



- 새 VS Code 창에서 ‘Linux’ 선택
- ‘계속’ 클릭
- RPI 암호 입력 (암호 : 12341234)
- VS Code에서 RPI 접속 완료

만약에 접속에 실패한다면 로컬 PC 터미널에서
$ ssh ubuntu@192.168.50.XXX로 접속할 것

RPI에서 pwm_test 노드 실행

SSH로 접속한 RPI VS Code 창에서 새 터미널 실행(Ctrl + Shift + `)

```
$ cm
//catkin_make의 단축키. 작업공간 내의 소스코드들을 빌드한다.
//만약 clock skew 에러로 빌드에 실패할 경우, sudo date -s “2022-12-10 tt:mm”을 입력.
$ roscore
$ rosrun drone_tutorial pwm_test
// 비행로봇의 8개 모터가 10% dutycycle로 움직인다.
```

drone_teleop.cpp 둘러보기

키보드 키를 통해 비행 로봇을 원격조종 할 수 있도록 토픽을 보내는 코드

KeyboardReader 클래스 : 키보드의 입력에 관한 함수들이 모여 있는 클래스

DroneTeleop 클래스 : 비행로봇의 원격조종 키입력과 토픽 발행에 대한 클래스

main 함수 : DroneTeleop 클래스로 객체를 생성하여 원격조종을 실행하는 함수

라이브러리 및 상수, 변수 설정

```
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
#include <signal.h>
#include <stdio.h>
#include <termios.h>
#include <unistd.h>

#define KEYCODE_RIGHT 0x43
#define KEYCODE_LEFT 0x44
#define KEYCODE_UP 0x41
#define KEYCODE_DOWN 0x42
#define KEYCODE_Q 0x71
#define KEYCODE_S 0x73
#define KEYCODE_Z 0x7a
#define KEYCODE_X 0x78
#define KEYCODE_SPACE 0x20
const int k_control_cycle = 100;
```

KeyboardReader 클래스

```
class KeyboardReader
{
public:
  KeyboardReader(): kfd(0)
  {
    // get the console in raw mode
    tcgetattr(kfd, &cooked);
    struct termios raw;
    memcpy(&raw, &cooked, sizeof(struct termios));
    raw.c_lflag &=~ (ICANON | ECHO);
    // Setting a new line, then end of file
    raw.c_cc[VEOL] = 1;
    raw.c_cc[VEOF] = 2;
    tcsetattr(kfd, TCSANOW, &raw);
  }
```

KeyboardReader 클래스(2)

```
  void readOne(char * c)
  {
    int rc = read(kfd, c, 1);
    if (rc < 0)
    {
      throw std::runtime_error("read failed");
    }
  }

  void shutdown()
  {
    tcsetattr(kfd, TCSANOW, &cooked);
  }

private:
  int kfd;
  struct termios cooked;
};

KeyboardReader input;
```

DronePWM 구조체

```
struct DronePWM
{
  double l_x;
  double l_y;
  double l_z;
  double a_w;
  double l_scale;
  double a_scale;
};
```

DroneTeleop 클래스(1)

```
class DroneTeleop
{
public:
  DroneTeleop():
  velocity_x(0),
  velocity_y(0),
  velocity_z(0),
  velocity_w(0),
  l_scale_(1000.0),
  a_scale_(1000.0)
  {
    nh_.param("scale_angular", a_scale_, a_scale_);
    nh_.param("scale_linear", l_scale_, l_scale_);

    twist_pub_ = nh_.advertise<geometry_msgs::Twist>("drone_teleop_pwm", 1);
  }
```

DroneTeleop 클래스(2)

```
void keyLoop()
{
  char c;
  bool dirty=false;
  ros::Rate loop_rate(k_control_cycle);

  puts("Reading from keyboard");
  puts("---------------------------");
  puts("Use keys to move the drone.");
  puts("↑ : foward       ↓ : backward");
  puts("← : left         → : right");
  puts("z : turn left    x : turn right");
  puts("s : stop         b : all");
  puts("q : quit");
```

DroneTeleop 클래스(3)

```
    while (ros::ok())
    {
      // get the next event from the keyboard
      try
      {
        input.readOne(&c);
      }
      catch (const std::runtime_error &)
      {
        perror("read():");
        return;
      }
      ROS_DEBUG("value: 0x%02X\n", c);
```

DroneTeleop 클래스(4)

```
    switch(c)
    {
      case KEYCODE_LEFT:
        ROS_DEBUG("LEFT");
        velocity_x = 0.0;
        velocity_y += 0.01;
        velocity_z = 0.0;
        velocity_w = 0.0;
        dirty = true;
        break;
      case KEYCODE_RIGHT:
        ROS_DEBUG("RIGHT");
        velocity_x = 0.0;
        velocity_y -= 0.01;
        velocity_z = 0.0;
        velocity_w = 0.0;
        dirty = true;
        break;
           ...
      case KEYCODE_Q:
        ROS_DEBUG("quit");
        return;
    }
```

DroneTeleop 클래스(5)

```
    if(velocity_x > 1.0)velocity_x = 1.0;
    else if(velocity_x < -1.0)velocity_x = -1.0;
    if(velocity_y > 1.0)velocity_y = 1.0;
    else if(velocity_y < -1.0)velocity_y = -1.0;
    if(velocity_z > 1.0)velocity_z = 1.0;
    else if(velocity_z < -1.0)velocity_z = -1.0;
    if(velocity_w > 1.0)velocity_w = 1.0;
    else if(velocity_w < -1.0)velocity_w = -1.0;

    geometry_msgs::Twist twist;
    twist.linear.x = l_scale_*velocity_x;
    twist.linear.y = l_scale_*velocity_y;
    twist.linear.z = l_scale_*velocity_z;
    twist.angular.x = a_scale_*velocity_w;
```

DroneTeleop 클래스(6)

```
      if(dirty ==true)
      {
        dirty=false;
        twist_pub_.publish(twist);
      }

      loop_rate.sleep();
      ros::spinOnce();
    }
    return;
  }

private:

  ros::NodeHandle nh_;
  double velocity_x, velocity_y, velocity_z, velocity_w, l_scale_, a_scale_;
  ros::Publisher twist_pub_;
};
```

quit 함수

```
void quit(int sig)
{
  (void)sig;
  input.shutdown();
  ros::shutdown();
  exit(0);
}
```

main 함수

```
int main(int argc, char** argv)
{
  ros::init(argc, argv, "drone_teleop");
  DroneTeleop drone_teleop;

  signal(SIGINT,quit);

  drone_teleop.keyLoop();
  quit(0);

  return 0;
}
```

로컬 PC에서 drone_teleop 노드 실행

로컬 PC의 VS Code 창에서 새 터미널 실행(Ctrl + Shift + `)

$ cm

$ roscore

$ rosrun drone_tutorial drone_teleop_key

```
chris20@DESKTOP-6906KK0:~/tutorial_ws/src$ rosrun drone_tutorial drone_teleop
Reading from keyboard
---------------------------
Use keys to move the drone.
↑ : foward      ↓ : backward
← : left        → : right
z : turn left   x : turn right
s : stop        q : quit
```

drone_controller.cpp 둘러보기

키보드 키를 통해 비행 로봇을 원격조종 할 수 있도록 토픽을 보내는 코드

TeleopCallback 함수 : 키보드 명령을 받고 해당하는 PWM 출력을 결정하는 함수

SetPwmDutycycle : 정해진 수치대로 PWM값을 조절하는 함수.

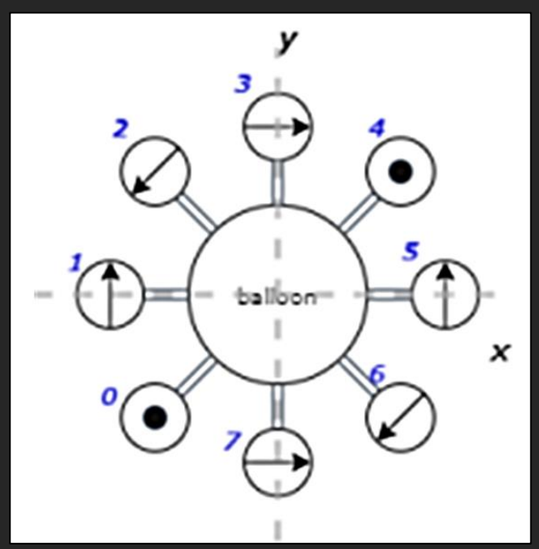
main 함수 : 원격조종 명령대로 비행로봇을 움직이는 함수

라이브러리 및 상수, 변수 설정

```
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
#include <pigpiod_if2.h>

#define constrain(amt, low, high) ((amt) < (low) ? (low) :
((amt) > (high) ? (high) : (amt)))
const int k_control_cycle = 10;
const int k_pins = 8;
const int k_pwm_range = 1000;
const int k_pwn_frequency = 20000;
int pin_nums[k_pins] = {17,4,22,27,6,5,19,13};
int pi_num;
int target_pwm[k_pins] = {};
```

TeleopCallback 함수

```
void TeleopCallback(const geometry_msgs::Twist &msg)
{
  double X_pos = (msg.linear.x >= 0) ? msg.linear.x : 0;
  double Y_pos = (msg.linear.y >= 0) ? msg.linear.y : 0;
  double Z_pos = (msg.linear.z >= 0) ? msg.linear.z : 0;
  double W_pos = (msg.angular.x > 0) ? constrain(msg.angular.x,15,250) : 0;
  double X_neg = (msg.linear.x < 0) ? msg.linear.x : 0;
  double Y_neg = (msg.linear.y < 0) ? msg.linear.y : 0;
  double W_neg = (msg.angular.x < 0) ? constrain(msg.angular.x,-250,-15) : 0;

  double vh = sqrt(2) * 0.5;

  target_pwm[0] = 0 + 0 + Z_pos + 0;
  target_pwm[1] = -X_neg * vh + Y_pos + 0 + W_pos;
  target_pwm[2] = -X_neg - Y_neg + 0 - W_neg;
  target_pwm[3] = X_pos - Y_neg * vh + 0 + W_pos;
  target_pwm[4] = 0 + 0 + Z_pos + 0;
  target_pwm[5] = -X_neg * vh + Y_pos + 0 - W_neg;
  target_pwm[6] = -X_neg - Y_neg + 0 + W_pos;
  target_pwm[7] = X_pos - Y_neg * vh + 0 - W_neg;
}
```



CheckRpi 함수

```
bool CheckPigpio()
{
  pi_num = pigpio_start(NULL, NULL);

  if (pi_num < 0)
  {
    ROS_ERROR("PI number is %d",pi_num);
    ROS_ERROR("PIGPIO connection failed.");
    return false;
  }

  ROS_INFO("Setup Finished.");
  return true;
}
```

InitPwm 함수

```
void InitPwm()
{
  for (int i = 0; i < k_pins; i++)
  {
    set_mode(pi_num, pin_nums[i], PI_OUTPUT);
    set_PWM_range(pi_num, pin_nums[i], k_pwm_range);
    set_PWM_frequency(pi_num, pin_nums[i], k_pwn_frequency);
  }
}
```

SetPwmDutycycle 함수(1)

```
void SetPwmDutycycle(int rate)
{
  if(rate < 0 || rate > k_pwm_range)
  {
    ROS_WARN("Invalid Dutycycle.");
    for (int i = 0; i < k_pins; i++) set_PWM_dutycycle(pi_num, pin_nums[i], 0);
  }

  for (int i = 0; i < k_pins; i++) set_PWM_dutycycle(pi_num, pin_nums[i], rate);
}
```

SetPwmDutycycle 함수(2)

```
void SetPwmDutycycle()
{
  for (int i = 0; i < k_pins; i++)
  {
    if(target_pwm[i] < 0 || target_pwm[i] > k_pwm_range)
    {
      ROS_WARN("Invalid Dutycycle.");
      set_PWM_dutycycle(pi_num, pin_nums[i], 0);
    }
    set_PWM_dutycycle(pi_num, pin_nums[i], target_pwm[i]);
  }
}
```

main 함수

```
int main(int argc, char **argv)
{
  ros::init(argc, argv, "drone_controller");
  ros::NodeHandle nh;
  ros::Rate loop_rate(k_control_cycle);
  ros::Subscriber sub_teleop = nh.subscribe("/drone_teleop_pwm", 5, TeleopCallback);

  if(!CheckPigpio()) return -1;
  InitPwm();

  while (ros::ok())
  {
    SetPwmDutycycle();
    loop_rate.sleep();
    ros::spinOnce();
  }

  SetPwmDutycycle(0);
  return 0;
}
```

SSH에서 drone_controller 노드 실행

SSH로 접속한 RPI VS Code 창에서 새 터미널 실행(Ctrl + Shift + `)

```
$ cm

$ roscore

$ rosrun drone_tutorial drone_teleop

$ rosrun drone_tutorial drone_controller

// 키보드를 통해 비행로봇을 원격 조종할 수 있다.
```

비행로봇 최종 데모

비행로봇을 통해 다음 미션을 수행하자.

출발지에서 띄워서 목적지에 도달하기

목적지에 도착하여 완전히 정지하면 성공!

목적지 중심으로부터의 거리를 재서 등수를 매김

1,2,3 등에게는 상품이 있습니다!

감 사 합 니 다