Bachelor Thesis

Controlling Quadrotor using Wearable Device

웨어러블 기기를 이용한 소형 로봇(쿼드롭터) 제어

By

Howoong Jun

June, 2016

Department of Electrical and Computer Engineering

Seoul National University

웨어러블 기기를 이용한 소형 로봇(쿼드롭터) 제어

지도교수 Songhwai Oh

이 논문을 공학 학사 학위 논문으로 제출함

서울대학교 공과대학

전기정보공학부

전 호 웅

전 호 웅의 학사 학위 논문을 인준함.

2016 년 11 월 11 일

지 도 교 수 오 성회 (인)

Abstract

This paper proposes new system for controlling quadrotor. The paper uses wearable device to control quadrotor. Wearable device is more intuitive and comfortable controlling method than the conventional one which uses two joysticks. In this new controlling system, wearable device sends the data to server and the server computes the data. Finally, the server sends computed instructions to the quadrotor. In order to make this system, sensor calibration of the wearable device is required first. After that, sensor fusion technique is needed to estimate the attitude (Roll, Pitch, Yaw) of the wearable device. Based on the attitude estimation, the wearable device can send data to the server. After receiving data from the wearable device, server should compute instructions for quadrotor. Also server should receive and compute data from the quadrotor sensor which include magnetic sensor, gyroscope, accelerometer, barometer and battery status.

Keywords: Quadrotor, Wearable Device, Bitcraze Crazyflie 2.0, Samsung Galaxy Gear S

Contents

List of Figures

List of tables

Chapter 1
Introduction

A quadrotor is a flying robot which can move in the 3D space. It can move x, y and z directions without limitations. Conventionally, to control the quadrotor, two joysticks are used, one is for x and y directions and the other is for z direction. However, joysticks are operated in the 2D plane, so it takes long time to get used to controlling quadrotor with joystick. Also this method requires users to use two hands because of two joysticks. Due to this counterintuitive and uncomfortable features of the conventional system, new controller which is intuitive and can control with one hand is needed.

This paper proposes new controlling system named Wearable Controlling System for Quadrotor, which uses wearable device to control quadrotor. The system uses 9-Degree of Freedom (DOF) Inertial Measurement Unit (IMU) sensors of wearable device to recognize the command from the user. After that, wearable device sends data to server via web socket. The server computes the received data and sends instructions to quadrotor. Also the server uses sensor data from the quadrotor to operate feedback control.

For the wearable device, Samsung Galaxy Gear S is used in this paper. It is because Galaxy Gear S has all the 9-DOF IMU sensors which include magnetic sensor, gyroscope and accelerometer. Also it has Wi-Fi module which is practical for building a web socket system. Moreover, Galaxy Gear S is app-based device, so it is easy to manipulate sensor data and build a web socket system. Tizen OS is used for the operating system of Galaxy Gear S and Javascript is used for the application program language.

For the server, Robot Operating System (ROS) is used. ROS can communicate with Crazyflie 2.0. Also it has web socket library called ROS-Bridge so it is easy to build web socket system by ROS. The data which is sent from wearable device to server via web socket is based on the JavaScript Object Notation (JSON) type. By JSON type, it is easy to manipulate data because ROS-Bridge automatically parses the data. The main server program is built with c++ language.

For the quadrotor, Bitcraze Crazyflie 2.0 is used. This quadrotor is micro quadrotor which weighs 27 grams and 9 cm^2 size. It includes Bluetooth module and 'Crazyradio' module which is for wireless communication. Also it has accelerometer, gyroscope, magnetic sensor, barometer and battery status so that it does not need extra sensors to control. Moreover, it is open source device so it is very easy to modify the controlling system. For the firmware, Crazyflie2-2014.12.0 version is used in this paper.

Chapter 2
Wearable Controlling System for Quadrotor

2.1 Wearable Device (Galaxy Gear S)

2.1.1 IMU Calibration

In order to modify the distortion by the misalignment, scaling and bias factor of sensor axis, IMU sensor data calibration is required. For accelerometer calibration, parameters are obtained by optimizing the acceleration of gravity into reference g through Levenberg-Marquardt algorithm. The cost function is the difference between g value of static interval and reference g value.

$$\theta^{acc} = [\alpha_{yz}, \alpha_{zy}, \alpha_{zx}, s_x^a, s_y^a, s_z^a, b_x^a, b_y^a, b_z^a] \tag{2.1}$$

$$a^O = h(a^S, \theta^{acc}) = T^a K^a (a^S + b^a) \tag{2.2}$$

$$L(\theta^{acc}) = \sum_{k=1}^{M} \left(\left\| g \right\|^2 - \left\| h(a_k^S, \theta^{acc}) \right\|^2 \right)^2 \tag{2.3}$$

In case of gyroscope, cost function is the sum of difference between gravity vector $u_{a,k}$ and $u_{g,k}$, which is obtained by multiplying rotation matrix to $u_{a,k-1}$ at each interval. The optimal parameters are obtained by Levenberg-Marquardt algorithm.

$$\theta^{gyro} = [\gamma_{yz}, \gamma_{zy}, \gamma_{xz}, \gamma_{zx}, \gamma_{xy}, \gamma_{yx}, s_x^g, s_y^g, s_z^g] \tag{2.4}$$

$$\omega^O = T^g K^g \omega^s \tag{2.5}$$

$$L(\theta^{gyro}) = \sum_{k=2}^{M} \left\| u_{a,k} - u_{g,k} \right\|^2 \quad (u_{g,k} = \psi[\omega_i^S, u_{a,k-1}]) \tag{2.6}$$

2.1.2 Sensor Fusion & Attitude Estimation

To estimate the roll and pitch values, Euler angle is required. To achieve this process, the angle should be calculated first by acceleration values.

$$\varphi = \operatorname{atan}\left(\frac{Acc_y}{Acc_z}\right) \tag{2.7}$$

$$\theta = \operatorname{atan}\left(\frac{Acc_x}{Acc_y}\right) \tag{2.8}$$

The calculated angle above should be transformed into Euler angle.

$$\varphi = \operatorname{atan}\left(\frac{Acc_y}{\sqrt{Acc_x^2+Acc_z^2}}\right) \tag{2.9}$$

$$\theta = \operatorname{atan}\left(\frac{Acc_x}{\sqrt{Acc_y^2+Acc_z^2}}\right) \tag{2.10}$$

After this process, it should be converged with gyroscope values by complementary filter.

$$Angle_{accurate} =$$

$$(GyroPercentage) * \ Angle_{Gyro} + (1 - GyroPercentage) * Angle_{Accel} \tag{2.11}$$

Yaw value is estimated by magnetic sensor and roll, pitch values. The tilt compensated magnetic vectors are as follows.

$$X' = X\cos(\phi) + Y\sin(\rho)\sin(\phi) - \ Z\cos(\rho)\sin(\phi) \tag{2.12}$$

$$Y' = Ycos(\rho) + Zsin(\phi) \tag{2.13}$$

The final compass heading (azimuth α) can be calculated using the equation as follows.

$$\alpha = \arctan\left(\frac{Y'}{X'}\right) \tag{2.14}$$

2.2 Server

There are two main roles for server in the system, communicate with wearable device and quadrotor. The system uses web socket to communicate with wearable device and for quadrotor, it uses crazyradio. By communicating with quadrotor, server computes feedback control. With feedback control, server controls two things, hovering and battery compensation.

2.2.1 Communicate with Wearable Device

For communicating with wearable device, web socket is used. In order to use web socket, wearable device and the server should be connected to the same Wi-Fi. Wearable device sends the messages in JSON format as follows.

```
{"msg": {"data": "Mode^Roll^Pitch^Thrust"},
"op": "publish",
"topic": "/topic_name"}
```

If server receives the messages, ROS-Bridge of server automatically publishes it to the server program and the program reads “data” object of the message. Data is one string format and it is separated with splitter ‘^’. The server program parses the data with the splitter into four parts, mode, roll, pitch and thrust. If mode equals 1, flying mode 1 of quadrotor is activated and if mode equals 2, flying mode 2 of quadrotor is activated. Received roll, pitch and thrust values are the estimated attitudes of wearable device so it should be adjusted for the quadrotor. The adjusting values are different based on the mode.

2.2.2 Communicate with Quadrotor

A. Hovering

For communicating with quadrotor, 'crazyradio' which is provided from bitcraze is used. The server sends messages by Crazyflie-ROS library function named sendSetpoint(). Also, the server receives the sensor data of quadrotor. The sensor data include accelerometer, gyroscope, magnetic sensor, barometer and battery status. Accelerometer, gyroscope, magnetic sensor and barometer are used for the hovering system and battery status is used for battery compensation system. All feedback controls in the system is processed with PD control which makes the system fast.

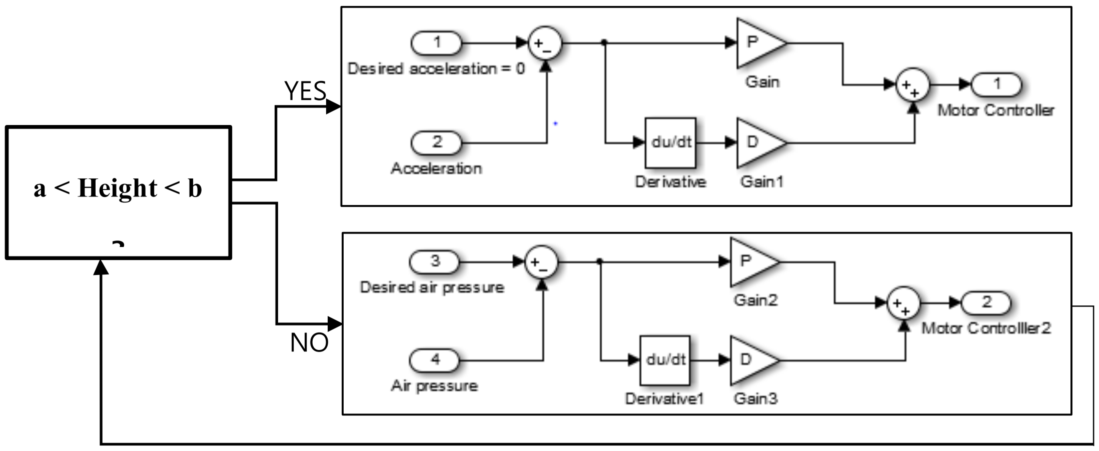

Fig 2.1 The feedback system for hovering – height control

Hovering system needs three feedback controls in order to maintain the position, height, roll and pitch control. Height control system is operated based on the barometer and z-axis accelerometer. The system checks the barometer value first and then if the barometer value is in the proper range, it checks the z-axis accelerometer value. The target value of the barometer is calculated as follows

$$Baro = initBaro + Offset \tag{2.15}$$

The variable ‘Baro’ indicates the target value of the barometer and ‘initBaro’ means initial barometer value before takeoff. Offset parameter is set to 0.55 in the experiment and the range parameter is set to $Baro \pm 0.52$. After this process, if the barometer value passes the condition, the system checks z-axis accelerometer value. Theoretically, target value of z-axis accelerometer which is at stable condition is 9.8 m/s^2. However, the actual value is not exactly 9.8 m/s^2 and it varies in every flight. Due to this feature, target value of z-axis accelerometer is set to the initial value before takeoff. The range parameter is set to $zAccel \pm 0.1\ m/s^2$.

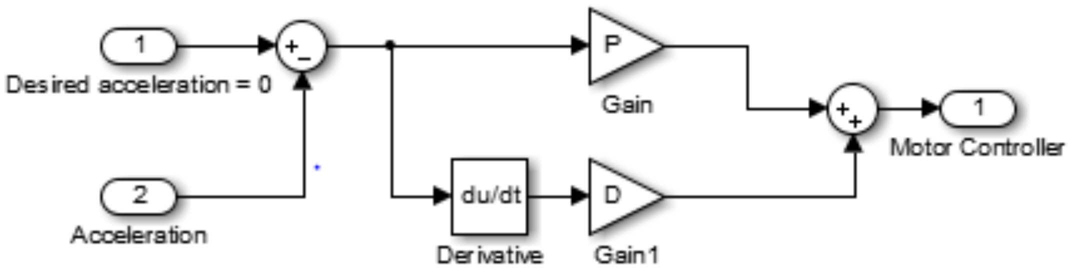

Fig 2.2 The feedback system for hovering – roll & pitch control

Roll and pitch control system are based on the same feedback system described in Fig 2.2. Target value of roll and pitch are $0\ m/s^2$ and the range parameter is set to $roll \pm 0.3\ m/s^2$ and $pitch \pm 0.3\ m/s^2$.

B. Battery Compensation

Crazyflie 2.0 does not have voltage regulator so the motor speed of quadrotor is decreased as the battery is consumed. In order to maintain the stable output motor speed, battery compensation is required. Crazyflie 2.0 provides battery status data so that it can be used as a battery compensation. The equation of current applied motor voltage that does not have battery compensation is as follows.

$$Motor\ Voltage = k_1 * Battery\ Voltage * Input\ Thrust \qquad (2.16)$$

Due to ‘Battery Voltage’ variable in the equation, motor voltage is dependent to current battery voltage. In order to delete the ‘Battery Voltage’ variable, the equation can be modified as follows.

$$Motor\ Voltage = \ k_2 * Battery\ Voltage * Input\ Thrust * \frac{\boldsymbol{Maximum\ Voltage}}{\boldsymbol{Battery\ Voltage}} \qquad (2.17)$$

Variable ‘Maximum Voltage’ value is 3.7 V which is referenced by Crazyflie 2.0 manual. With this battery compensation, the quadrotor flies constantly until the battery status reaches 3.7 V. If the battery voltage dropped under 3.7 V, Crazyflie2.0 will stop flying.

2.3 Quadrotor (Crazyflie 2.0)

The proposed system includes two modes, mode 1 and mode 2. Mode 1 is controlling quadrotor by mapping attitudes of wearable device to the instructions of quadrotor. Mode 2 is controlling quadrotor using motion vector of the wearable device.

2.3.1 Flying Mode 1

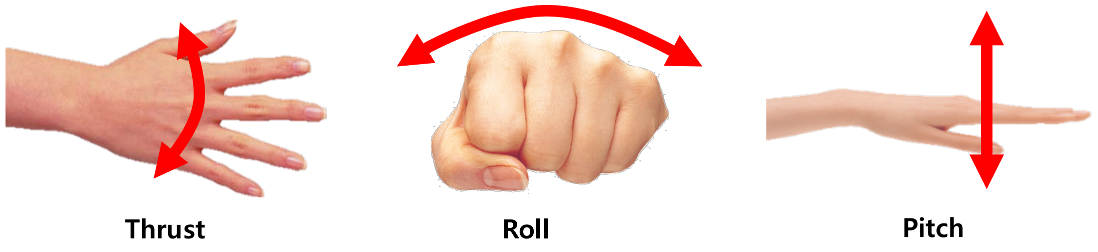

Fig 2.3 Mode 1 controlling motions

Fig 2.3 indicates the controlling motions for mode 1. Wearable device has three attitudes, yaw, roll and pitch. The method of mode 1 is to map these attitudes to the instructions of quadrotor. Yaw attitude of wearable device is mapped to thrust instruction of quadrotor and roll and pitch attitudes of wearable device is mapped to roll and pitch instructions of quadrotor each. Also in order to match the units between attitudes of wearable device and instructions of quadrotor, unit matching parameter is applied. For yaw attitude 400/3 is applied and for roll and pitch attitudes 1/3 is applied each. Also yaw attitude is angle variable, so it ranges from 0-degree to 360-degree. However, for controlling with wearable device, only 0-degree to 90-degree range is needed so yaw range between 90-degree to 360-degree is used for emergency stop. If yaw angle reaches this yaw range, the quadrotor stops moving.

Table 2.1 Mapping attitudes to the instructions of quadrotor

Wearable Device Attitudes	Quadrotor Instructions	Unit Matching Parameter
Yaw	Thrust	400/3
Roll	Roll	1/3
Pitch	Pitch	1/3

2.3.2 Flying Mode 2

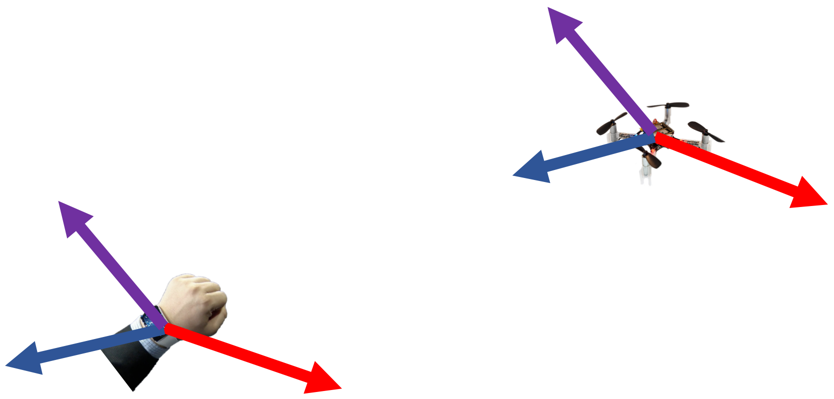
Fig 2.4 Mode 2 controlling motions

Mode 2 of the system is based on the motion vector of wearable device. As the wearable device moves, the server calculates the direction and magnitude of the motion vector and send quadrotor proper instructions based on the motion vector. Specifically, the server integrates the acceleration values and calibrate it for the quadrotor. Also, in order to avoid fine movements of the wearable device to be applied to the quadrotor, the system has ‘stable zone’. If the acceleration values are in the ‘stable zone’ for three times successively, the system recognize it as ‘stable status’ and make quadrotor hovers. If the acceleration values reach the ‘stable zone’ under 3 times it does not recognize it as ‘stable status’. This regulation is required in the acceleration transition state. Even though the acceleration value reaches the stable zone, it is not stable in the acceleration transition state so the system has to distinguish this difference.

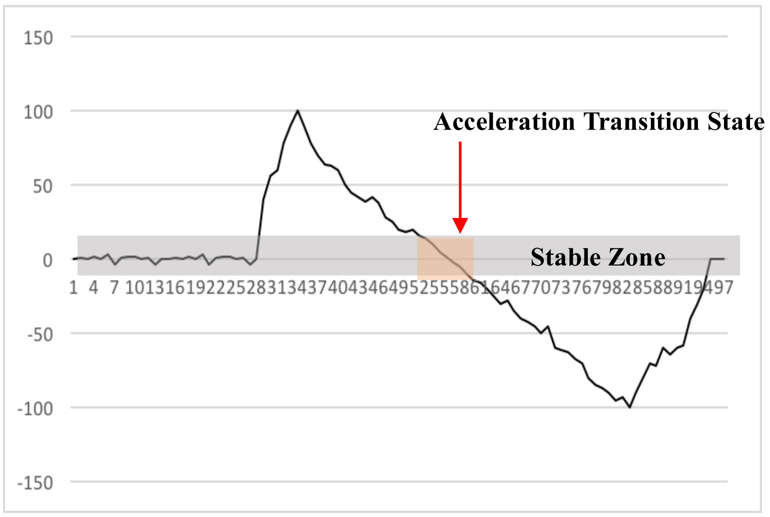

Fig 2.5 Graph of acceleration example and stable zone

Chapter 3
Conclusion

The paper proposes new controlling system for quadrotor which uses wearable device. The system includes wearable device, server and quadrotor. With sensor values from wearable device and quadrotor, server computes instructions and send it to quadrotor. The proposed system is intuitive because wearable device can move in a 3D space, not like joystick which can only move in 2D plane. Also, it is convenient method because the system needs only one hand contrary to joystick method which uses two hands. However, due to uncalibrated sensor values of quadrotor, feedback system was not stable enough to control it smoothly. In order to make the system stable, sensor values of quadrotor should be calibrated. Also, in order to make motor speed completely independent from battery status, voltage regulating circuits should be applied. If these problems are solved, the new quadrotor controlling system will be convenient enough to replace the conventional methods.

Bibliography

[1] Hanna, William, “Modelling and Control of an Unmanned Aerial Vehicle” *Charles Darwin University*, May. 2014.

[2] Phlivantürk, Can, “Lossless convexification of quadrotor motion planning with experiments”, *University of Texas at Austin*, Sep. 2014.

[3] Jihoon Lee, “Web Applications for Robots using rosbridge”, *Brown University*, 2012.

[4] D. Tedaldi, A. Pretto and E. Menegatti, “A robust and easy to implement method for IMU calibration without external equipments”, *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pp.3042-3049, 2014.

[5] Guerrero-Castellanos, J.F., et al. “Design and implementation of an Attitude and Heading Reference System (AHRS).” *Electrical Engineering Computing Science and Automatic Control (CCE), 2011 8th International Conference on.* IEEE, 2011.

[6] Watson, Matthew, “The Design and Implementation of a Robust AHRS for Integration into a Quadrotor Platform.” *Meng electronic engineering, Department of electronic & electrical engineering*, May. 2013.

[7] Madgwick, Sebastian Oh, “An efficient orientation filter for inertial and inertial/magnetic sensor arrays.” *Report x-io and University of Bristol (UK)*, 2010.

Appendix

Server Program Code – C++

```
#include <iostream>
#include <unistd.h>
#include <thread>
#include <sstream>
#include <vector>
#include <string>

#include "ros/ros.h"
#include "std_msgs/Float32.h"
#include "std_msgs/String.h"

#include "crazyflie_driver/LogBlock.h"
#include "crazyflie_driver/AddCrazyflie.h"
#include "crazyflie_driver/GenericLogData.h"
#include "crazyflie_driver/UpdateParams.h"
#include "sensor_msgs/Imu.h"
#include "sensor_msgs/Temperature.h"
#include "sensor_msgs/MagneticField.h"
#include "std_srvs/Empty.h"
#include "geometry_msgs/Twist.h"

#include <mutex>
#include <sstream>
#include <boost/program_options.hpp>
#include <crazyflie_cpp/Crazyradio.h>
#include <crazyflie_cpp/Crazyflie.h>
#include <crazyflie_cpp/crtp.h>

int instruct = 0;

const float rollTrim = -7;
const float pitchTrim = -2;

void getThrustPoint();
std::vector<std::string> splitMsg(std::string str, char delimiter);
void thrustCallback(const std_msgs::String::ConstPtr& msg);

//Fundamental Functions
constexpr double pi() { return std::atan(1)*4; }

double degToRad(double deg) {
   return deg / 180.0 * pi();
}

double radToDeg(double rad) {
   return rad * 180.0 / pi();
}

//Hover Mode Class
class HoverMode{
public:
   int HoverModeThrust(float baro, float accel);
   float HoverModeRollPitch(float accel, bool RPflag);
   float HoverModeYaw(float gyroZ);

private:
   float PDthrustControl(float err, float& err_, float offset, float standard);
   float errBaro1 = 0.0;
   float errBaro2 = 0.0;
   float errAccelz1 = 0.0;
   float errAccelz2 = 0.0;
   float errRoll = 0.0;
   float errPitch = 0.0;
   float errYaw = 0.0;
   ros::Time m_previousTime;
};

class crazyflieValues{
public:
   crazyflieValues(){
      thrust = 0;
      roll = 0.0;
      pitch = 0.0;
      yawrate = 0.0;
   }

   void setValues(uint16_t inThrust, float inRoll, float inPitch, float inYaw) {
      thrust = inThrust;
      roll = inRoll;
      pitch = inPitch;
      yawrate = inYaw;
   }

   uint16_t motorSetThrust(float batStatus);
   float getRoll() const { return roll; }
   float getPitch() const { return pitch; }
   float getYaw() const { return yawrate; }

   float calcPeak(float inAcc, int rpt){
      std::cout<<"inACC "<<inAcc<<std::endl;
      if(rpt == 1){
         sumAccRoll += inAcc;
         std::cout<<"Roll SUM: "<<sumAccRoll<<std::endl;
         return sumAccRoll;
      }
      else if(rpt == 2) {
         sumAccPitch += inAcc;
```

```
            std::cout<<"Pitch SUM: "<<sumAccPitch<<std::endl;
            return sumAccPitch;
        }
        else if(rpt == 3) {
            sumAccThrust += inAcc;
            std::cout<<"Thrust SUM: "<<sumAccThrust<<std::endl;
            return sumAccThrust;
        }
    }

    void sumAccZero(int rpt){
        if(rpt == 1) sumAccRoll = 0;
        else if(rpt == 2) sumAccPitch = 0;
        else if(rpt == 3) sumAccThrust = 0;
    }

    void printValues(){
        std::cout<<"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"<<roll<<std::endl;
    }

    float baroTemp, linAccTemp;

private:
    static float sumAccRoll, sumAccPitch, sumAccThrust ;
    static float maxAcc, minAcc;
    static uint16_t thrust;
    static float roll;
    static float pitch;
    static float yawrate;
};

uint16_t crazyflieValues::thrust = 0;
float crazyflieValues::roll = 0.0;
float crazyflieValues::pitch = 0.0;
float crazyflieValues::yawrate = 0.0;
float crazyflieValues::maxAcc = 0.0;
float crazyflieValues::minAcc = 0.0;
float crazyflieValues::sumAccRoll = 0.0;
float crazyflieValues::sumAccPitch = 0.0;
float crazyflieValues::sumAccThrust = 0.0;

//Crazyflie ROS Class
class CrazyflieROS
{
public:
    //CrazyflieROS Constructor
    CrazyflieROS(const std::string& link_uri,bool enable_logging)
    : m_cf(link_uri)
    , m_isEmergency(false)
    , m_enableLogging(enable_logging)
    , m_sentSetpoint(false)
    {
        std::thread t(&CrazyflieROS::run, this);
        t.detach();
    }

private:
  struct logImu {
    float acc_x;
    float acc_y;
    float acc_z;
    float gyro_x;
    float gyro_y;
    float gyro_z;
  } __attribute__((packed));

  struct log2 {
    float mag_x;
    float mag_y;
    float mag_z;
    float baro_temp;
    float baro_pressure;
    float pm_vbat;
  } __attribute__((packed));

public:
  bool emergency(
    std_srvs::Empty::Request& req,
    std_srvs::Empty::Response& res)
  {
    ROS_FATAL("Emergency requested!");
    m_isEmergency = true;

    return true;
  }

  template<class T, class U>

  void trashFunc() { /*Don't know the reason why this function is needed, but it IS neccesary */  }

  void run()
  {

    auto start = std::chrono::system_clock::now();

    std::unique_ptr<LogBlock<logImu> > logBlockImu;
    std::unique_ptr<LogBlock<log2> > logBlock2;
    if (m_enableLogging) {
      ROS_INFO("Requesting Logging variables...");
      m_cf.requestLogToc();

      std::function<void(logImu*)> cb = std::bind(&CrazyflieROS::onImuData, this, std::placeholders::_1);

      logBlockImu.reset(new LogBlock<logImu>(
        &m_cf,{
          {"acc", "x"},
          {"acc", "y"},
          {"acc", "z"},
          {"gyro", "x"},
          {"gyro", "y"},
          {"gyro", "z"},
        }, cb));
      logBlockImu->start(1); // 10ms
      std::function<void(log2*)> cb2 = std::bind(&CrazyflieROS::onLog2Data, this, std::placeholders::_1);

      logBlock2.reset(new LogBlock<log2>(
        &m_cf,{
          {"mag", "x"},
          {"mag", "y"},
          {"mag", "z"},
          {"baro", "temp"},
          {"baro", "pressure"},
          {"pm", "vbat"},
        }, cb2));
      logBlock2->start(10); // 100ms
    }
    auto end = std::chrono::system_clock::now();

    // Send 0 thrust initially for thrust-lock
    for (int i = 0; i < 100; ++i) {
       m_cf.sendSetpoint(0, 0, 0, 0);
    }

    while(!m_isEmergency) {
      // make sure we ping often enough to stream data out
      if (m_enableLogging && !m_sentSetpoint) {
        m_cf.sendPing();
      }
      m_sentSetpoint = false;
      std::this_thread::sleep_for(std::chrono::milliseconds(1));
    }

    // Make sure we turn the engines off
    for (int i = 0; i < 100; ++i) {
       m_cf.sendSetpoint(0, 0, 0, 0);
    }

  }

  void onImuData(logImu* data) {

    msgI.orientation_covariance[0] = -1;

    // measured in deg/s; need to convert to rad/s
    msgI.angular_velocity.x = degToRad(data->gyro_x);
    msgI.angular_velocity.y = degToRad(data->gyro_y);
    msgI.angular_velocity.z = degToRad(data->gyro_z);

    // measured in mG; need to convert to m/s^2
    msgI.linear_acceleration.x = data->acc_x * 9.81;
    msgI.linear_acceleration.y = data->acc_y * 9.81;
    msgI.linear_acceleration.z = data->acc_z * 9.81;

  }

  void onLog2Data(log2* data) {

    {
      // measured in degC
      msgTemp.temperature = data->baro_temp;
    }

    {
      // measured in Tesla
      msgMag.magnetic_field.x = data->mag_x;
      msgMag.magnetic_field.y = data->mag_y;
      msgMag.magnetic_field.z = data->mag_z;
    }

    {
      // hPa (=mbar)
      msgBaro.data = data->baro_pressure;
    }

    {
      // V
      msgBat.data = data->pm_vbat;
    }
  }

public:
  Crazyflie m_cf;
  sensor_msgs::Temperature msgTemp;
  sensor_msgs::MagneticField msgMag;
  std_msgs::Float32 msgBaro;
  std_msgs::Float32 msgBat;
  sensor_msgs::Imu msgI;

private:
  bool m_isEmergency;
  bool m_enableLogging;
  bool m_sentSetpoint;
};

int main(int argc, char **argv)
{
    try
  {
    //Create Crazyflie ROS constructor
    CrazyflieROS* cf = new CrazyflieROS("radio://0/80/250K",1);
    HoverMode crazyflieHover;
    crazyflieValues cv;

    //Initialize ROS
    ros::init(argc, argv, "scan");
    ros::NodeHandle n;

    //Create ROS Publisher & Subscriber
    ros::Publisher pub = n.advertise<std_msgs::String>("pThrust", 1000);
    ros::Subscriber sub  = n.subscribe("pThrust", 1000, thrustCallback);

    std::cout<<"start"<<std::endl;
    std::cout<<"thread start\n \n ****************\n <Instructions> \n 1: Emergency Stop \n 2: Terminate Program \n 3: Resume E.Stop \n 4: Hover
Mode \n 5: Crazyflie Status \n **************** \n"<<std::endl;
```

```
    std::thread t1(&getThrustPoint);

    int j = 0;

    //Main Loop
    while(ros::ok()){
        cv.baroTemp = cf->msgBaro.data;
        cv.linAccTemp = cf->msgI.linear_acceleration.z;

        //Emergency Stop(intstruction = 1) and Terminate Program
        if(instruct == 1) cv.setValues(0, 0.0, 0.0, 0.0);
        else if(instruct == 2) break;

        else if(instruct == 4) {
            cv.setValues(
                    crazyflieHover.HoverModeThrust(cf->msgBaro.data, cf->msgI.linear_acceleration.z),
                    crazyflieHover.HoverModeRollPitch(cf->msgI.linear_acceleration.y, true),
                    crazyflieHover.HoverModeRollPitch(cf->msgI.linear_acceleration.x, false),
                        0);
        }
        else if(instruct == 5) {
            std::cout<<"\nBattery Check      : "<<cf->msgBat.data<<std::endl;
            std::cout<<"Linear Acceleration : ("<<cf->msgI.linear_acceleration.x<<","<<cf->msgI.linear_acceleration.y<<","<<cf-
>msgI.linear_acceleration.z<<")"<<std::endl;
            std::cout<<"Barometer          : "<<cf->msgBaro.data<<std::endl;
            std::cout<<"Angular Velocity    : ("<<cf->msgI.angular_velocity.x<<","<<cf->msgI.angular_velocity.y<<","<<cf-
>msgI.angular_velocity.z<<")"<<std::endl;
            std::cout<<"Magnetic Sensor     : ("<<cf->msgMag.magnetic_field.x<<","<<cf->msgMag.magnetic_field.y<<","<<cf-
>msgMag.magnetic_field.z<<")\n"<<std::endl;
            instruct = 1;
        }

        //Compensate Thrust with Battery Status
        uint16_t thrustInput = cv.motorSetThrust(cf->msgBat.data);

        //Lowest Thrust Value
        if(thrustInput < 10001 && 0 < thrustInput){
            thrustInput = 10001;
        }

        //Highest Thrust Value
        else if(thrustInput > 65535){
            thrustInput = 65534;
        }

        //Send Final data to Crazyflie
        cf->m_cf.sendSetpoint(cv.getRoll() + rollTrim, cv.getPitch() + pitchTrim, cv.getYaw(), thrustInput);

        //Receive data from websocket - ROS subscriber
        ros::spinOnce();

    }

    //Console Control Thread
    t1.join();

    std::cout<<"Program Terminated"<<std::endl;
    return 0;
    }

    catch(std::exception& e)
    {
        std::cerr << e.what() << std::endl;
        return 1;
    }
}

//For Console Input Thread
void getThrustPoint(){
    while(1){
        if(instruct != 5){
            std::cout<<"<INPUT(1 to stop)>"<<std::endl;
            std::cout<<"instructions: ";
            std::cin>>instruct;

            if(instruct == 2){
                break;
            }
        }
    }
}

//For compensating thrust value by battery status
uint16_t crazyflieValues::motorSetThrust(float batStatus){
    if(thrust == 0) return 0;
    float percentage = 3.7 / batStatus;
    if(batStatus == 0) percentage = 0;
    uint16_t outThrust = percentage * thrust;
    return outThrust;
}

//For parsing message which is from websocket
std::vector<std::string> splitMsg(std::string str, char delimiter){
    std::vector<std::string> internal;
    std::stringstream ss(str);
    std::string tok;

    while(getline(ss, tok, delimiter)){
        internal.push_back(tok);
    }

    return internal;
}

int zeroFlagRoll = 0;
int zeroFlagPitch = 0;
int zeroFlagThrust = 0;
//Get message from websocket - ROS subscriber
void thrustCallback(const std_msgs::String::ConstPtr& msg){
    crazyflieValues cvTemp;
    HoverMode hoverTemp;
```

```cpp
    float intValue[4];
    uint16_t thrustInput = 0;
    float rollInput = 0.0, pitchInput = 0.0;
    std::vector<std::string> sep = splitMsg(msg->data.c_str(), '^');
    for(int i = 0; i<4; i++){
        intValue[i] = atof(sep[i].c_str());
    }

    if(intValue[0] == 1){
        if(intValue[3] > 10 && intValue[3] < 300) thrustInput = hoverTemp.HoverModeThrust(cvTemp.baroTemp, cvTemp.linAccTemp);
        else thrustInput = 0;
        if((intValue[1] > -1 && intValue[1] < 1 )|| (intValue[2] > -1 && intValue[2] < 1)) {
            thrustInput += 1000;
            std::cout<<"!!"<<std::endl;
        }

        cvTemp.setValues(
            thrustInput + intValue[1] * 18.5 + intValue[2] * 18.5,
            -2.5 + intValue[1]/3,
            -5 + intValue[2]/3 + 10,
            0
        );
    }
    else if(intValue[0] == 2){
        if(instruct != 1 && instruct != 2){
            if(intValue[1] < 3 && intValue[1] > -3 && intValue[2] < 3 && intValue[2] > -3 && intValue[3] > 7 && intValue[3] < 12){
                instruct = 4;
                std::cout<<"Stable"<<std::endl;
                zeroFlagRoll++;
                zeroFlagPitch++;
                zeroFlagThrust++;
                if(zeroFlagRoll > 3){
                    cvTemp.sumAccZero(1);
                }
                if(zeroFlagPitch > 3){
                    cvTemp.sumAccZero(2);
                }
                if(zeroFlagThrust > 3){
                    cvTemp.sumAccZero(3);
                }
           }

            else{
                instruct = 3;
                zeroFlagRoll = 0;
                zeroFlagPitch = 0;
                zeroFlagThrust = 0;
                rollInput = cvTemp.calcPeak(intValue[1], 1) * 2;
                pitchInput = cvTemp.calcPeak(intValue[2], 2) * 2.5;
                if(pitchInput > 40) pitchInput = 40;
                else if(pitchInput < -38) pitchInput = -38;
                if(rollInput > 40) rollInput = 40;
                else if(rollInput < -50) rollInput = -50;
                if(intValue[3] - 9.7 < 0) thrustInput = cvTemp.calcPeak(intValue[3] - 9.7, 3) * 300 + 32767 + 8000;
                else thrustInput = cvTemp.calcPeak(intValue[3] - 9.7, 3) * 30 + 32767;

                std::cout<<intValue[3]<<","<<-intValue[1]<<","<<-intValue[2]<<std::endl;
                std::cout<<thrustInput<<","<<-rollInput<<","<<-pitchInput<<std::endl;
                cvTemp.setValues(thrustInput + 3000, -rollInput - 3, -pitchInput - 3,0);
            }
        }
        else if(intValue[0] == 0){
            instruct = 1;
            cvTemp.setValues(0, 0, 0, 0);
        }
    }
}

bool flagBaro = false;
float tmpBaro = 0;
float tmpAccel = 0;

int HoverMode::HoverModeThrust(float baro, float accel){
    const int HoverThrustConst = 32767 + 5500;
    const int HoverThrustOffset = 3000;
    const float BaroConst = 0.55;
    const float AccelConst = 0;//0.1;
    const float baroInit = 0.6;

    int HoverThrust = HoverThrustConst;

    //First Barometer Value
    if(baro != 0 && !flagBaro) {
        flagBaro = true;
        tmpBaro = baro;
        tmpAccel = accel;
        std::cout<<"Hover Mode Test"<<std::endl;
        std::cout<<"Standard - Baro: "<<tmpBaro<<", Accel: "<<tmpAccel<<std::endl;
    }

    if(baro != 0 && flagBaro){
        //if too high
        if(tmpBaro - baroInit - BaroConst > baro) {
            HoverThrust = PDthrustControl(tmpBaro - baroInit - BaroConst - baro, errBaro1, HoverThrustOffset - 2000, HoverThrustConst);
        }
        //if too low
        else if(tmpBaro - baroInit + BaroConst < baro) {
            HoverThrust = PDthrustControl(baro - tmpBaro + baroInit - BaroConst, errBaro2, HoverThrustOffset + 5000, HoverThrustConst);
            HoverThrust = HoverThrust + 1000;
        }
        else{
            //if too high
            if(tmpAccel - AccelConst > accel){
                HoverThrust = PDthrustControl(tmpAccel - AccelConst - accel, errAccelz1, - HoverThrustOffset, HoverThrustConst + 4000);
            }
            //if too low
            else if(tmpAccel + AccelConst < accel) {
                HoverThrust = PDthrustControl(accel - tmpAccel - AccelConst, errAccelz2, HoverThrustOffset - 1000, HoverThrustConst);
            }
            else HoverThrust = HoverThrustConst;
        }
    }
    return HoverThrust;
}


float HoverMode::HoverModeRollPitch(float accel, bool RPflag){
    float HoverRollPitch;
    const float accelStd = 0.0;

    if(accel > accelStd) {
        if(RPflag){
            HoverRollPitch = PDthrustControl(accel - accelStd, errRoll, rollTrim - 3, 0);
}
        else{
            HoverRollPitch = PDthrustControl(accel - accelStd, errPitch, -10, 0);
        }
    }
    else if(accel < -accelStd) {
        if(RPflag) {
            HoverRollPitch = PDthrustControl(accelStd - accel, errRoll, 1, 0);
        }
        else {
            HoverRollPitch = PDthrustControl(accelStd - accel, errPitch, -1, 0);
        }
    }

    else HoverRollPitch = 0;

    return HoverRollPitch;
}

float HoverMode::HoverModeYaw(float gyroZ){
    float HoverYaw;
    const float accelStd = 1;

    if(gyroZ > accelStd){
        HoverYaw = PDthrustControl(gyroZ - accelStd, errYaw, 0.5, 0);
        std::cout<<"Yaw Accel ++:"<<gyroZ<<","<<HoverYaw<<std::endl;
    }
    else if(gyroZ < -accelStd){
        HoverYaw = PDthrustControl(-accelStd - gyroZ, errYaw, -0.5, 0);
        std::cout<<"Yaw Accel --:"<<gyroZ<<","<<HoverYaw<<std::endl;
    }

    //For safety
    if(HoverYaw > 10 || HoverYaw < -10) HoverYaw = 0;

    return HoverYaw;
}

float HoverMode::PDthrustControl(float err, float& err_, float offset, float standard){
    ros::Time time = ros::Time::now();
    float errDiff = (err - err_);
    float outputControl = standard + offset * (0.8 * err + 0.2 * errDiff);

    m_previousTime = time;
    err_ = err;
    return outputControl;
}
```

Galaxy Gear S Application main code – Javascript

```
         var webSocketUrl = "ws://172.20.10.4:9090/"; //8-4
         var webSocket;
         webSocket = new WebSocket(webSocketUrl);

         // If the connection is established

         webSocket.onopen = function(e)
         {
           console.log('connection open, readyState: ' + e.target.readyState);
         };

          //If the connection fails or is closed with prejudice
         webSocket.onerror = function(e)
         {
           /* Error handling */
             console.log('connection closed, readyState: ' + e.target.readyState);
         };



 $(window).load(function(){


         var onSave = false, docuDir, newFile, fileStream, startTime, SService, UVSensor, LSensor, MagSensor, PSensor, HAM, data2Write =
 [],mode0Btn,mode1Btn,mode2Btn, bgBtn, saveBtn, endSaveBtn;

         var pitch=0, roll=0, yaw=0;

         //for GPS
         var loc = navigator.geolocation;

         //end save. close stream
         function offSave(){
             console.log("Save off");
             onSave = false;
             fileStream.close();
             document.getElementById("isSave").innerHTML = "Save OFF";
         }

         //This listens for the back button press
         document.addEventListener('tizenhwkey', function(e) {
             if(e.keyName === "back"){
                           tizen.application.getCurrentApplication().exit();
                           offSave();
             }
         });

         //callback if it succeeds to start sensor (except Heart Rate, GPS)
         function onStartSensor(sensor){
             console.log("Sensor" + sensor + "start");
         }

         //callback if it fails (sensor start, getData)
         function onFailSensor(error){
             console.log("Fail : " + error);
         }

         //for name of file
         startTime = tizen.time.getCurrentDateTime();

         //time to String
         function timePrint(time){
             var hour, minute, second, milSec;

             hour = time.getHours();
             minute = time.getMinutes();
             second = time.getSeconds();
             milSec = time.getMilliseconds();

             return hour.toString()+"_" + minute.toString() + "_" + second.toString() + "_" + milSec.toString();
         }

         //create txt file
         tizen.filesystem.resolve("documents",
             function(result){
                           docuDir = result;
                           newFile = docuDir.createFile(startTime.getMonth().toString() + "_"
 +startTime.getDate().toString()+"_"+timePrint(startTime)+".txt");
                           console.log("file open : " + newFile);
             }, function(error){
                           console.log("error : " + error.message);
             }, 'rw'
         );

         //for convenience
         document.getElementById("time").innerHTML = "Start at " + timePrint(startTime);

         //Sensor service is require to start each sensor
         SService = window.webapis&&window.webapis.sensorservice;  //tizen.sensorservice is not worked
         UVSensor = SService.getDefaultSensor("ULTRAVIOLET");
         LSensor = SService.getDefaultSensor("LIGHT");
         MagSensor = SService.getDefaultSensor("MAGNETIC");
         PSensor = SService.getDefaultSensor("PRESSURE");
         HAM = (tizen&&tizen.humanactivitymonitor)||(window.webapis&&window.webapis.motion); //Heart Rate is another application, not sensor
 service

         //start each sensor
         UVSensor.start(onStartSensor, onFailSensor);
         LSensor.start(onStartSensor, onFailSensor);
         MagSensor.start(onStartSensor, onFailSensor);
         PSensor.start(onStartSensor, onFailSensor);


         //add listener. listener is called when sensor value has changed
         UVSensor.setChangeListener(function(data){
             document.getElementById("UV").innerHTML = 'UV : ' + data.ultravioletLevel;
          data2Write[10] = data.ultravioletLevel;
     });


     MagSensor.setChangeListener(function(data){
          document.getElementById("xmag").innerHTML = 'Mag X : ' + -data.x;
          document.getElementById("ymag").innerHTML = 'Mag Y : ' + data.y;
          document.getElementById("zmag").innerHTML = 'Mag Z : ' + data.z;

          data2Write[7] = data.x;
          data2Write[8] = data.y;
          data2Write[9] = data.z;
     });

     LSensor.setChangeListener(function(data){
          document.getElementById("light").innerHTML = 'Light : ' + data.lightLevel;

          data2Write[11] = data.lightLevel;
     });

     PSensor.setChangeListener(function(data){
          document.getElementById("press").innerHTML = 'Pressure : ' + data.pressure;

          data2Write[12] = data.pressure;
     });

     //start Heart Rate sensor
     HAM.start("HRM",function(data){
          document.getElementById("heart").innerHTML = 'HeartRate : ' + data.heartRate;

          data2Write[13] = data.heartRate;
     });

     //start GPS
     HAM.start("GPS",function(data){
          var GPSdata = data.gpsInfo[0];

          document.getElementById("latitude").innerHTML = 'Latitude : ' + GPSdata.latitude;
          document.getElementById("longitude").innerHTML = 'Longitude : ' + GPSdata.longitude;
          document.getElementById("altitude").innerHTML = 'Altitude : ' + GPSdata.altitude;
          document.getElementById("speed").innerHTML = 'Speed : ' + GPSdata.speed;

          data2Write[14] = GPSdata.latitude;
          data2Write[15] = GPSdata.longitude;
          data2Write[16] = GPSdata.altitude;
          data2Write[17] = GPSdata.speed;
     });

     var currTime = tizen.time.getCurrentDateTime();

     window.addEventListener('deviceorientation', function(e){
          data2Write[18] = -e.beta;
          data2Write[19] = e.gamma;
          data2Write[20] = e.alpha;

          document.getElementById("pitch").innerHTML = 'Pitch : ' + (e.gamma)  ;
          document.getElementById("roll").innerHTML = 'Roll : ' + (-e.beta) ;
          document.getElementById("yaw").innerHTML = 'Yaw : ' + (e.alpha) ;
     },true);


     var ax_=0, ay_=0, az_=0, mode=0,ax,ay,az;
     //get acceleration and angular velocity (reference : Mozilla Web API)
     window.addEventListener('devicemotion', function(e) {
          var axo,ayo,azo, axi,ayi,azi,rotxi,rotyi,rotzi,rotx,roty,rotz,interval, magyaw;

          axi = -e.accelerationIncludingGravity.x;
          ayi = e.accelerationIncludingGravity.y; //- attach (-)
          azi = e.accelerationIncludingGravity.z;
          rotxi =  e.rotationRate.alpha ;
          rotyi = -e.rotationRate.beta ;
          rotzi = -e.rotationRate.gamma ;

          //apply calibration
          ax= (9005086496992679*axi)/9007199254740992 - (970847136413691*ayi)/1152921504606846976 + (4676546001099593*azi)/4611686018427387904
- 8343091890828650998003174283065555/20769187434139310514121985316880384;
          ay= (9003026989503915*ayi)/9007199254740992 - (4684841617841737*azi)/2305843009213693952 -
1273430245934866820054511086505921/2596148429267413814265248164610048;
          az= (2252626011982959*azi)/2251799813685248 + 2563386722619340963857975513927/2535301200456458802993406410752;


          axo = (-
Math.sin(data2Write[20]*Math.PI/180)*Math.sin(data2Write[18]*Math.PI/180)*Math.sin(data2Write[19]*Math.PI/180)+Math.cos(data2Write[20]*Math.PI/180)
*Math.cos(data2Write[19]*Math.PI/180))*ax + (-Math.sin(data2Write[20]*Math.PI/180)

*Math.cos(data2Write[18]*Math.PI/180))*ay +
(Math.sin(data2Write[20]*Math.PI/180)*Math.sin(data2Write[18]*Math.PI/180)*Math.cos(data2Write[19]*Math.PI/180)+Math.cos(data2Write[20]*Math.PI/180
)*Math.sin(data2Write[19]*Math.PI/180))*az;
          ayo =
(Math.cos(data2Write[20]*Math.PI/180)*Math.sin(data2Write[18]*Math.PI/180)*Math.sin(data2Write[19]*Math.PI/180)+Math.sin(data2Write[20]*Math.PI/180
)*Math.cos(data2Write[19]*Math.PI/180))*ax + (Math.cos(data2Write[20]*Math.PI/180)

*Math.cos(data2Write[18]*Math.PI/180))*ay + (-
Math.cos(data2Write[20]*Math.PI/180)*Math.sin(data2Write[18]*Math.PI/180)*Math.cos(data2Write[19]*Math.PI/180)+Math.sin(data2Write[20]*Math.PI/180)
*Math.sin(data2Write[19]*Math.PI/180))*az;
          azo = (-Math.cos(data2Write[18]*Math.PI/180)*Math.sin(data2Write[19]*Math.PI/180))*ax + (Math.sin(data2Write[18]*Math.PI/180))*ay +
(Math.cos(data2Write[18]*Math.PI/180)*Math.cos(data2Write[19]*Math.PI/180))*az;

          ax = axo-ax_;
          ay = ayo-ay_;
          az = azo-az_;


          ax_=axo;
          ay_=ayo;
          az_=azo;


          ax=parseInt(axo*100)/100;
          ay=parseInt(ayo*100)/100;
          az=parseInt(azo*100)/100;
```

```
             rotx= (4532074390415255*rotxi)/4503599627370496 + (2883896438722165*rotyi)/72057594037927936 -
(8721539557351861*rotzi)/288230376151711744;
             roty=  (2432704819743611*rotxi)/72057594037927936 + (554278758788703*rotyi)/562949953421312 +
(7048490289617383*rotzi)/72057594037927936;
             rotz=  (4157351154835707*rotzi)/4503599627370496 - (3328889871184437*rotyi)/144115188075855872 -
(5800857746681135*rotxi)/1152921504606846976;

             var tmp = tizen.time.getCurrentDateTime();

             if(tmp.getSeconds()==currTime.getSeconds()){
             interval = tmp.getMilliseconds() - currTime.getMilliseconds();
             }else{
                           interval = 1000 + tmp.getMilliseconds() - currTime.getMilliseconds();
             }
             currTime= tmp;

             magyaw = Math.atan2(data2Write[8]*Math.cos(roll*Math.PI/180)-data2Write[9]*Math.sin(roll*Math.PI/180),

         data2Write[7]*Math.cos(pitch*Math.PI/180)+data2Write[8]*Math.sin(pitch*Math.PI/180)*Math.sin(roll*Math.PI/180)+data2Write[9]*Math.cos(roll
*Math.PI/180)*Math.sin(pitch*Math.PI/180));

             yaw = 0.95*(yaw+rotz*interval/1000)+0.05*magyaw*180/Math.PI;


             data2Write[0] = interval;
             data2Write[1] = ax;
             data2Write[2] = ay;
             data2Write[3] = az;
             data2Write[4] = rotx;
             data2Write[5] = roty;
             data2Write[6] = rotz;

             roll = data2Write[18];
             pitch = data2Write[19];
             yaw = data2Write[20];

             document.getElementById("interv").innerHTML =  'Interval : ' +  interval;
             document.getElementById("xaccel").innerHTML =  'AccX : ' +  ax;
             document.getElementById("yaccel").innerHTML = 'AccY : ' + ay;
             document.getElementById("zaccel").innerHTML = 'AccZ : ' + az;

             document.getElementById("rotx").innerHTML = 'Rot X : ' + rotx ;
             document.getElementById("roty").innerHTML = 'Rot Y : ' + roty ;
             document.getElementById("rotz").innerHTML = 'Rot Z : ' + rotz ;

             //save all data to txt
             if(onSave === true){
                           fileStream.write(timePrint(currTime)+","+data2Write.toString()+"\n");
                           console.log(timePrint(currTime)+","+data2Write.toString());
             }
             if(mode === 0){
                           var msg = {"msg": {"data": 0+"^"+0+"^"+0+"^"+0},
                                                      "op": "publish",
                                                      "topic": "/pThrust"
                                                     };
                           webSocket.send(JSON.stringify(msg));
             }
             if(mode === 1){
                           roll=parseInt(roll*100)/100;
                           pitch=parseInt(pitch*100)/100;
                           yaw=parseInt(yaw*100)/100;
                           var msg = {"msg": {"data": mode+"^"+roll+"^"+pitch+"^"+yaw},
                                                      "op": "publish",
                                                      "topic": "/pThrust"
                                                     };
                           webSocket.send(JSON.stringify(msg));
             }
             if(mode === 2){
                           var msg = {"msg": {"data": mode+"^"+ax+"^"+ay+"^"+az},
                                                      "op": "publish",
                                                      "topic": "/pThrust"
                                                     };
                           webSocket.send(JSON.stringify(msg));

             }

         });

         mode0Btn = document.getElementById("mode0-btn");
         mode0Btn.addEventListener("click",function(){
             mode=0;
         });

         mode1Btn = document.getElementById("mode1-btn");
         mode1Btn.addEventListener("click",function(){
             mode =1;

         });

         mode2Btn = document.getElementById("mode2-btn");
         mode2Btn.addEventListener("click",function(){
             mode=2;

         });

         //app hide. Need to enable background-support in config.xml
         bgBtn = document.getElementById("bg-btn");
         bgBtn.addEventListener("click", function(){
             var app = tizen.application.getCurrentApplication();
             app.hide();
         });//add event listener to app hide button

         //start save. open stream to save
         saveBtn = document.getElementById("save-btn");
         saveBtn.addEventListener("click", function(){
             console.log("Save on");
             newFile.openStream("w",
                           function(fs){
                                        fileStream = fs;
                                        console.log("Hello gear!");
                           }, function(e){
```

```
                                        console.log("error : " + e.message);
                              });
              onSave = true;
              document.getElementById("isSave").innerHTML = "Save ON";
          });//add event listener to save button

          endSaveBtn = document.getElementById("saveEnd-btn");
          endSaveBtn.addEventListener("click", offSave);//add event listener to save end button

  } );
```

초 록

본 논문은 쿼드롭터를 제어하는 방법으로 웨어러블 기기를 사용하는 새로운 시스템을 제안한다. 기존의 조이스틱을 이용한 컨트롤은 직관성이 떨어지고, 두 손을 모두 사용해야 한다는 단점이 있다. 반면 웨어러블 기기는 그것에 비해 매우 직관적이고, 한 손만을 사용하여 제어가 가능하다. 본 논문에서 제안하는 새로운 시스템에서는 웨어러블 기기의 센서 데이터를 서버에 전송하고, 서버가 받은 데이터를 처리한다. 그리고 서버는 처리한 데이터를 바탕으로 쿼드롭터에 전달할 명령을 계산하여 쿼드롭터에 보낸다. 본 시스템을 구축하기 위해서는 웨어러블 기기의 센서 값의 Calibration 이 먼저 필요하다. 이후에 Sensor Fusion 기술을 통해 Roll, Pitch, Yaw 값을 계산하여 서버에 전송한다. 서버는 웨어러블 기기로부터 받은 데이터를 바탕으로 쿼드롭터에 명령을 내린다. 또한 서버는 웨어러블 기기 뿐만 아니라 쿼드롭터로부터 지자기 센서, 자이로스코프, 가속도 센서, 기압계, 배터리 상태와 같은 값을 받아 피드백 제어에 사용한다.

주요어: Quadrotor, Wearable Device, Bitcraze Crazyflie 2.0, Samsung Galaxy Gear S