RINGS-SVGS

How to Use It for EMFF tests

Written by: Nasir Hariri

Fall 2018
# Table of Contents

1. **INTRODUCTION** .................................................................................................................................................................................. 3

1.1 Mechanical/Electrical setups ................................................................................................................................................................. 4

1.2 Software setup: Uploading code to the microcontroller .................................................................................................................... 6

2. **A PROCEDURE TO RUN AN EMFF EXPERIMENT** .................................................................................................................................. 10

2.1 Procedure for the stationary vehicle ...................................................................................................................................................... 10

2.2 Procedure for moving the vehicle ......................................................................................................................................................... 12

2.3 Setup of external IR triggering circuitry ............................................................................................................................................. 14

2.4 Setup of SVGS sensor .............................................................................................................................................................................. 14

3. **APPENDIX: ARDUINO CODES FOR EMFF TESTS** ...................................................................................................................................... 16

3.1 Arduino Code for stationary vehicle: ..................................................................................................................................................... 16

3.2 Code for moving vehicle ......................................................................................................................................................................... 18

3.2.1 CL_main_NH.............................................................................................................................................................................................. 18

3.2.2 CMD........................................................................................................................................................................................................... 21

3.2.3 DataLogger ........................................................................................................................................................................................... 23

3.2.4 Fan........................................................................................................................................................................................................ 24

3.2.5 IMU_BNO055 ............................................................................................................................................................................................ 26

3.2.6 LQ_servo ................................................................................................................................................................................................ 29

3.2.7 RINGS_EM .................................................................................................................................................................................................. 30

3.2.8 SMC ....................................................................................................................................................................................................... 32

3.2.9 SVGS ......................................................................................................................................................................................................... 34

3.3 Code for IR triggering circuitry ................................................................................................................................................................. 38
1 INTRODUCTION

The goal of this manual is to provide sufficient information for future researchers in how to operate the RINGS-SVGS system (shown in the figure below).

The RINGS-SVGS system consists of three main subsystems, which are:

- Ducted-fan motion platform
- RINGS hardware
- SVGS sensor

All three subsystems are connected and capable of communicating with each other using a developed Arduino code. This allows the main microcontroller (Arduino Due) to control the operation of the ducted fans, and EMFF mode of RINGS. In addition, it receives data from different sensors such as SVGS and gyroscope. This manual will highlight the software development, which allows for the control, and data collection operations of the SVGS-RINGS system.

The ducted-fan motion platform, seen in FIG XX, was designed and developed to allow for navigation of the RINGS hardware in a flat floor facility at the FIT-ASAP lab. The system consists of the following experimental setups:

- Mechanical/Electrical setups
- Software setup
1.1 Mechanical/Electrical setups

The design of the ducted-fan motion platform can be seen in the figure below, where eight Electric Ducted Fans (EDFs) are used.

![Figure 1-1. ducted-fan motion platform.](image)

There is a number of electric devices that are used in the developed motion platform (as can be seen in the figure below). Note that:

- The Arduino DUE is the main microcontroller that runs the operation of EMFF tests and sends commands to the ducted fans as well as the RINGS hardware, so the developed code, that will be shown later, must be uploaded to this Arduino board (through the *programming* port).
- The microSD board allows inserting microSD card to save experimental data.
- The master switch provides manual control to the main DeWalt 20VDC battery that energizes and powers the ducted-fans actuators, LED targets, and Solenoid valve (to control the operation of the air bearings).

![Figure 1-2. Electric devices and their locations on the developed motion platform.](image)
All hardware connections between all electric devices are shown in the following electric diagram (Just in case a connection needed to be replaced or verified for future researchers):
1.2 Software setup: Uploading code to the microcontroller

To write a program and send it to the main microcontroller of the motion platform (Arduino DUE), an Arduino IDE software is used.

Uploading code to the microcontroller:

1. Select the correct board:
   - in the Arduino IDE:
     ▪ Tools ➔ Board ➔ Arduino Due (Programming Port)
   - If you don’t see the Arduino Due board, install the driver from:
     ▪ Tools ➔ Board ➔ Boards Manager (Then search for “Arduino Due” and install its driver).

2. Select the serial port of the Arduino Due board (as seen below)
   - Make sure you first connect the PC to the programming micro-USB port of the Arduino Due (not the native port!)
   - Then, in the Arduino IDE:
     ▪ Tools ➔ Port ➔ COMxx (Arduino Due (Programming Port))

3. Upload the code to the Arduino board:
   - Make sure the code was uploaded successfully,
     ▪ Some Arduino Due boards exhibit errors when uploading a code, which can be solved by disconnecting and then reconnecting the USB port.
The main software that was developed to control the system (can be seen in the figure below) is called:

- **CL_main_NH**

However, different supporting files were also developed to control different elements in the system:

- CMD, DataLogger, Fan, IMU_BNO055, LQ-servo, RINGS_EM, SMC, SVGS.
- These files will appear as tabs in the Arduino IDE as shown below:

![Table of developed Arduino code]

**Figure 1-3. Main folder of the developed Arduino code.**

![Arduino IDE window with code]

**Figure 1-4. Developed Arduino code to control SVGS-RINGS system.**
Different tabs were created to control different parts of the system. A summary of the main objectives for each tab can be highlighted as follows:

1. **CL_main_NH** tab:
   a. It defines global variables,
   b. Initialize electric devices and their communications in the `void setup()` function, and
   c. Runs the main loop of the controller in the `void loop()` function
      i. Then, the system operates in a sequence of operations that are defined by varies `Flag` values (This will be illustrated further next)

2. **CMD** tab:
   a. Defines the desired command variables ($x_d = x_x, x_y$ and $x_\theta$) for the tracking controller to follow.

3. **Data_logger** tab:
   a. Initialize and configure the settings to write on a microSD card.

4. **Fan** tab:
   a. Attach the proper PWM pins to the corresponding EDFs,
   b. Arms the Electronic Speed Controllers (ESC), and
   c. Send the control input magnitudes (computed by the LQ-servo, or the SMC controllers) to the corresponding fans to run them and generate thrust forces/torques.

5. **IMU_BNO055** tab:
   a. Setup the I2C communication with the IMU sensor,
   b. Collect the gyroscope data of the z-axis ($\dot{\theta}$), and
   c. Computes the relative angle ($\theta$)

6. **LQ-servo** tab:
   a. Apply the algorithm for the LQ-servo controller for the 3-DOF system

7. **RINGS_EM** tab:
   a. Defines the command packets for the RINGS modes and EMFF operations, and
   b. Sends a current command to RINGS EMFF mode:
      i. Desired duty cycle
      ii. Desired phase delay (0°, or 180°)

8. **SMC** tab:
   a. Apply the algorithm for the Sliding mode controller for the EMFF controller (Only applied to control the axial axis between aligned coils)

9. **SVGS** tab:
   a. Acquire the packet-based serial data of the SVGS,
   b. Perform data manipulations
      i. SLIP protocol
      ii. IEEE-754 single-precision floating-point format
   c. Define position variables of SVGS sensor, and
   d. Apply Kalman filter algorithm.
As mentioned, the **void loop** function in the `CL_main_NH` tab runs the main loop of the code by defining different **Flags**. Therefore, it is important to understand the way it was designed to operate the system. The following flowchart highlights the operation of the system:
2 A PROCEDURE TO RUN AN EMFF EXPERIMENT

The following procedures describe the processes needed to perform an experiment using the EMFF tests, where the experimental setup consists of stationary and moving SVGS-RINGS systems (As seen in the figure below).

![Experimental setup of an EMFF test.](image)

Since this experimental setup consists of two systems, different procedures are being utilized to each vehicle. Therefore, the following actions were used to describe the complete EMFF tests:

- Procedure for the stationary vehicle.
- Procedure for the moving vehicle.
- Setup of external IR triggering circuitry.
- Setup of SVGS sensor.

2.1 Procedure for the stationary vehicle

Since the stationary vehicle maintained its position at a fixed location, there is no need to run the ducted-fan subsystem. Only the operation of the EMFF mode of this RINGS hardware is needed to be established for the aimed EMFF test. Note that, this RINGS hardware will run a constant RMS current of about 50% duty cycle with zero phase delay during the EMFF mode.

Therefore, a new Arduino code was written to manually control the operation of the RINGS and thus send a fixed value of RMS current command using the microcontroller of the developed motion platform. The following steps describe the procedures:
1. The initialization phase of a stationary vehicle
   a. The master switch is turned OFF (No need to power the ducted-fans).
   b. Connect the PC to the microcontroller (Arduino Due) of the stationary vehicle through a serial port (USB) and ensure that you uploaded the “RINGS_Modes” code
   c. Note: Baud rate for the serial monitor is 115200
      i. To check if the connection is established between the PC and the Arduino:
         1. Open the “RINGS_Modes” code in Arduino IDE
         2. Open the Serial monitor
         3. Type in “1”, and then click on Send
         4. You should see a note in the serial monitor as:
            User Input is: 1
            IDLE mode packet was sent

2. Controlling RINGS modes manually
   a. Turn ON the RINGS hardware.
      i. You should see the LCD of the RINGS hardware turns ON
   b. In the Arduino Serial monitor, type “1” and send it to activate the IDLE mode.
      i. You should see the LCD of RINGS showing IDLE mode
   c. Thereafter, send “2” to activate the EMFF mode.
      i. Once the EMFF mode was activated, you might send “5” to activate the EMFF mode with a constant 50% duty cycle (This step can also be made once the moving vehicle enters the EMFF mode as well)
   d. Done!
      i. The system will now run a constant RMS current through its coil.
      ii. Upon completing the EMFF test, deactivate the system by:
         1. Send “1” to enter again the IDLE mode
         2. Once the system operates in an IDLE mode, you can switch OFF the RINGS hardware

A detailed view of the code, that used to manually control the operation of RINGS hardware, can be seen in the figure below:

A list of user Input commands:
1 ➔ IDLE mode
2 ➔ EMFF mode
3 ➔ EMFF mode: 0% duty cycle
4 ➔ EMFF mode: 10% duty cycle
5 ➔ EMFF mode: 50% duty cycle
6 ➔ EMFF mode: 60% duty cycle
7 ➔ EMFF mode: 70% duty cycle
2.2 Procedure for moving the vehicle

Another developed Arduino software operates the moving vehicle, which consists of three main phases to complete a single EMFF test, which are:

- An Initialization phase of moving vehicle.
- A 3-DOF ducted-fan based control phase.
- An EMFF control phase.

NOTE THAT: The eight ducted fans on the system operate at fast speed, and it becomes VERY DANGEROUS to place your hand/fingers in close proximity. ALWAYS KEEP A SAFE DISTANCE FROM THE FANS WHEN THE MAIN POWER IS TURNED ON! And also at any time during the experiments to avoid an accident.

1. The initialization phase of a moving vehicle. First, make sure the system is initially turned OFF
   a. The master switch is turned OFF.
   b. Batteries (for both the Ducted fan and the RINGS systems) are fully charged.
   c. SVGS smartphone is charged and placed properly.
   d. Supplied compressed air is available (@ ~100 psi) and main valve is turned on.
   e. Micro-SD card is inserted for data collection.
   f. IR timed-circuitry is powered and placed properly to trigger IR sensors of both RINGS.

2. Turn ON the system
   a. Turn ON the main switch, and simultaneously reset the Arduino Due board vies the reset button on its attached shield.
      i. You should hear multiple beeps to indicates that Electronic Speed Controller (Talon25) are energized, armed, and ready to operate.
   b. Then, Turn ON the RINGS system.
      i. You should see the LCD of the RINGS system turns ON and state an IDLE mode.
   c. Turn ON the SVGS sensor
      i. Open the SVGS app in the smartphone:
         1. Choose the 3rd option (IOIO-FIT mode) ➔ Start the SVGS
         2. You should see data are being computed/calculated for all the 6 states on the SVGS GUI.
         3. If not, Make sure the SVGS detects the LED target (successful calculation on the SVGS GUI).
      ii. Be careful: the fans will turn ON as soon as the SVGS detects the target!
         1. If not, make sure the Bluetooth dongle is connected to the IOIO board and solid blue light is highlighted.

3. A 3-DOF ducted-fan based control phase is ACTIVE
   a. Once the SVGS sensor is turned on, the control system will be ACTIVE automatically
      i. The control system will move the SVGS-RINGS system in 3-DOF to the home/desired position:
1. Zero coaxial distance and relative angle.
2. A constant separation distance of 0.7 m.
   b. As soon as the system reaches the desired position, the EMFF mode will start automatically.

4. An EMFF control mode is ACTIVE
   a. EMFF mode command is sent to the RINGS hardware
      i. Current command.
      ii. Phase delay command (0°, or 180°).
   b. The system will be controlled in 3-DOF
      i. The co-axial axis between coils is controlled by the developed 1-DOF EMFF controller.
      ii. The other 2-DOF are controlled by the ducted fans subsystem to ensure both RINGS are maintained aligned during this EMFF test.

5. Upon completing the designed maneuver, the system will turn OFF
   a. The main relay will switch off the main power line:
      i. Turn OFF the ducted fans
      ii. Turn OFF the LED target
      iii. Turn OFF the air bearing operation
   b. RINGS will enter the IDLE mode

6. Manually, turn OFF the main switch
   a. To ensure the fans are not powered
   b. Then, take off the microSD card and save the experimental data to a PC.

7. DONE!
2.3 Setup of external IR triggering circuitry

In order to maintain coils of both RINGS hardware synchronized, an IR pulse must be generated by an external IR triggering circuitry (as seen in the figure below). This circuitry consists of two IrDA2 click boards (made by MikroElektronika) that are placed next to each other and pointing away to allow triggering both RINGS systems at the same moment (at 2 Hz). For electric connections, simply connect the Tx pin of the Arduino to the Rx pin of both boards. Also, powering the board by 5VDC from the Arduino board as well as connecting the grounds of both devices.

Note that, this IR triggering circuitry must be placed in a location where it allows to trigger at least one of the IR sensors in both RINGS hardware. Also, this additional Arduino board can be powered by external power bank.

![External IR triggering circuitry](image)

*Figure 2-2. The external IR triggering circuitry.*

2.4 Setup of SVGS sensor

The following parameters were adjusted in the settings menu of the SVGS app:

- Image Height: 1920
- Image Height: 1080
- Focal length:
  - S8 smartphone: 1510
  - Nexus S smartphone: 1850
- Threshold:
  - LED target: 250
  - Retro-reflective target: 140-170

Also, ensure the correct selection of the SVGS target:

- Select the type of target (LED, or retro-reflective) in the setting menu
- Insert the proper dimensions of the SVGS target
**Other Observations/Notes:**

- Be extra careful when working/operating close to the ducted fans.
- Batteries of RINGS last about 5-10 minutes.
- Don’t forget to insert the microSD card for data collection prior to the start of the test.
- When switching the RINGS hardware ON, on some occasions, the system won’t start up. Thus, turn the system OFF for a few seconds and then turn it ON again (only if you are sure that you have fully charged batteries).
- To save experimental data of other variables:
  - In the SVGS tab ➔ Modify the “DataString” by adding additional variables that you want to save.
- Operation of the EMFF tests are sensitive to external disturbance as well as friction, so try to keep the glass surface as clean as possible and place the tube of the compressed air in a location with minimum effect on the system.
- For the operation of EMFF tests, make sure that the IR triggering circuitry is active, powered, and placed correctly prior to the start of the EMFF test. Future researchers might also integrate this circuitry to the RINGS-SVGS motion platform if needed to easily be powered, and controlled.
- For future development of the different design of controllers:
  - Add additional tab for each developed controller to the main Arduino code (CL_main_NH) to keep it organized.
  - To deactivate a ducted-fan subsystem **during an EMFF test**, in the **void loop** of CL_main_NH and under the operation of Flag == 2:
    1. Comment out the function that activates the corresponding fan:
       - Fans_Effort(); // Turn ON attitude controller
       - Fans_Effort_pos_x(); // Turn ON x axes controller
       - Fans_Effort_pos_z(); // Turn ON z axes controller
    2. Define, or activate, the function corresponding to the designed EMFF controller:
       - i.e.: to activate the SMC for the EMFF for axial configuration:
         - SMC_u(); // Activate SMC for EMFF
3.1 Arduino Code for stationary vehicle:

This code was uploaded to the microcontroller (Arduino DUE), and was used to control the operation of the EMFF mode of the stationary RINGS.

```c
#define SERIAL_SPEED 115200
byte IDLE_Packet[] = {0xFF, 0x02, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00};
byte EMFF_Packet[] = {0xFF, 0x02, 0x02, 0x00, 0x00, 0x00, 0x53, 0x0D, 0x0A};
byte EMFF_180_Packet[] = {0xFF, 0x02, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0A};
int DutyCycle = 0;
byte DutyCycle_Hex = 0;
byte EMFF_DutyCycle_Packet_part1[] = {0xFF, 0x03, 0x00};
byte EMFF_DutyCycle_Packet_part2[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x0D, 0x0A};
byte EMFF_DutyCycle_Packet_part2_180phase[] = {0xB4, 0x00, 0x00, 0x00, 0x00, 0x0D, 0x0A};

void setup() {
  Serial.begin (SERIAL_SPEED); //Serial.println(" 1) Serial setup >> done ");
  Serial2.begin (SERIAL_SPEED); //Serial.println(" 1) Serial2 setup >> done ");
}

void loop() {
  if (Serial.available() > 0) {
    String s = Serial.readString();
    int User_Input = s.toInt();
    Serial.print (" User Input is: "); Serial.println (User_Input);
    if (User_Input == 1) {
      IDLE_mode();
      Serial.println (" IDLE mode packet was sent ");
    } else if (User_Input == 2) {
      EMFF_mode();
      Serial.println (" EMFF mode was sent ");
    } else if (User_Input == 3) {
      EMFF_DutyCycle();
      Serial.println (" EMFF-0: 0% DutyCycle-active mode was sent ");
    } else if (User_Input == 4) {
      EMFF_DutyCycle_10();
      Serial.println (" EMFF-0: 10% DutyCycle-active mode was sent ");
    } else if (User_Input == 5) {
      EMFF_DutyCycle_50();
      Serial.println (" EMFF-0: 50% DutyCycle-active mode was sent ");
    } else if (User_Input == 6) {
      EMFF_DutyCycle_60();
      Serial.println (" EMFF-0: 60% DutyCycle-active mode was sent ");
    } else if (User_Input == 7) {
      EMFF_DutyCycle_70();
      Serial.println (" EMFF-0: 70% DutyCycle-active mode was sent ");
    } else if (User_Input == 22) {
      EMFF_180_mode();
      Serial.println (" EMFF-180 mode was sent ");
    } else if (User_Input == 44) {
      EMFF_DutyCycle_10_180Phase();
      Serial.println (" EMFF-180: 10% DutyCycle-active mode was sent ");
    } else if (User_Input == 55) {
      EMFF_DutyCycle_50_180Phase();
      Serial.println (" EMFF-180: 50% DutyCycle-active mode was sent ");
    }}}

void IDLE_mode() {
  Serial2.write(IDLE_Packet, sizeof(IDLE_Packet));
}

void EMFF_mode() {
  Serial2.write(EMFF_Packet, sizeof(EMFF_Packet));
}

void EMFF_180_mode() {
  Serial2.write(EMFF_180_Packet, sizeof(EMFF_180_Packet));
}
```

APPENDIX: ARDUINO CODES FOR EMFF TESTS

This code was uploaded to the microcontroller (Arduino DUE), and was used to control the operation of the EMFF mode of the stationary RINGS.
void EMFF_DutyCycle() {
    DutyCycle = 0;  // Update the dutycycle variable based on the optimal control signal
    DutyCycle = constrain(DutyCycle,0,99);  
    DutyCycle = DutyCycle * 2;  // conversion factor (2 * dutycycle = byte 4)
    DutyCycle_Hex = (DutyCycle, HEX);
    Serial2.write(EMFF_DutyCycle_Packet_part1,sizeof(EMFF_DutyCycle_Packet_part1));
    Serial2.write(DutyCycle);
    Serial2.write(EMFF_DutyCycle_Packet_part2,sizeof(EMFF_DutyCycle_Packet_part2));
}

void EMFF_DutyCycle_10() {
    DutyCycle = 10;  // Update the dutycycle variable based on the optimal control signal
    DutyCycle = constrain(DutyCycle,0,99);  
    DutyCycle = DutyCycle * 2;  // conversion factor (2 * dutycycle = byte 4)
    DutyCycle_Hex = (DutyCycle, HEX);
    Serial2.write(EMFF_DutyCycle_Packet_part1,sizeof(EMFF_DutyCycle_Packet_part1));
    Serial2.write(DutyCycle);
    Serial2.write(EMFF_DutyCycle_Packet_part2,sizeof(EMFF_DutyCycle_Packet_part2));
}

void EMFF_DutyCycle_50() {
    DutyCycle = 50;  // Update the dutycycle variable based on the optimal control signal
    DutyCycle = constrain(DutyCycle,0,99);  
    DutyCycle = DutyCycle * 2;  // conversion factor (2 * dutycycle = byte 4)
    DutyCycle_Hex = (DutyCycle, HEX);
    Serial2.write(EMFF_DutyCycle_Packet_part1,sizeof(EMFF_DutyCycle_Packet_part1));
    Serial2.write(DutyCycle);
    Serial2.write(EMFF_DutyCycle_Packet_part2,sizeof(EMFF_DutyCycle_Packet_part2));
}

void EMFF_DutyCycle_10_180Phase() {
    DutyCycle = 10;  // Update the dutycycle variable based on the optimal control signal
    DutyCycle = constrain(DutyCycle,0,99);  
    DutyCycle = DutyCycle * 2;  // conversion factor (2 * dutycycle = byte 4)
    DutyCycle_Hex = (DutyCycle, HEX);
    Serial2.write(EMFF_DutyCycle_Packet_part1,sizeof(EMFF_DutyCycle_Packet_part1));
    Serial2.write(DutyCycle);
    Serial2.write(EMFF_DutyCycle_Packet_part2_180phase,sizeof(EMFF_DutyCycle_Packet_part2_180phase));
}

void EMFF_DutyCycle_50_180Phase() {
    DutyCycle = 50;  // Update the dutycycle variable based on the optimal control signal
    DutyCycle = constrain(DutyCycle,0,99);  
    DutyCycle = DutyCycle * 2;  // conversion factor (2 * dutycycle = byte 4)
    DutyCycle_Hex = (DutyCycle, HEX);
    Serial2.write(EMFF_DutyCycle_Packet_part1,sizeof(EMFF_DutyCycle_Packet_part1));
    Serial2.write(DutyCycle);
    Serial2.write(EMFF_DutyCycle_Packet_part2_180phase,sizeof(EMFF_DutyCycle_Packet_part2_180phase));
}
3.2 Code for moving vehicle

This code was uploaded to the microcontroller (Arduino DUE), and thereafter was used to control the operation of the moving SVGS-RINGS system during an EMFF test. Note that, this Arduino code consists of nine main parts as mentioned previously (CL_main_NH, CMD, DataLogger, Fan, IMU_BNO055, LQ_servo, RINGS_EM, SMC, SVGS).

3.2.1 CL_main_NH

```cpp
#include <math.h>
#define SERIAL_SPEED 57600
#define SERIAL_SPEED_RINGS_PIC32 115200
int SVGS_Started = 0;
double SVGS_Started_TimeStamp = 0;
double SVGS_Started_TimeStamp_Previous = 0;
int SVGS_i = 0;
double Data_Time = 0;
double Previous_Time = 0;
String DataString = ""
boolean SensorFlag = false;
float Error = 0;
float Error_pos_x = 0;
float Error_pos_z = 0;
float Effort = 0;
float Effort_pos_x = 0;
float Effort_pos_z = 0;
float Effort_and_ESC_offset = 0;
float CMD = 0;
float CMD_pos_x = 0;
float CMD_pos_z = 0.7;
float FBK_IMU_yaw = 0; // From IMU: BNO055
float FBK_IMU_yaw_dot = 0; // From IMU: BNO055
float FBK_IMU_Acc_x = 0;
float FBK_IMU_Acc_z = 0;
float FBK_IMU_Velocity_x = 0;
float FBK_IMU_Velocity_z = 0;
fractional float FBK_SVGS_roll = 0; // From SVGS
float derivative_pos_x = 0;
int mag_Effort_pos_sin = 0;
int mag_Effort_pos_z = 0;
int mag_Effort_Fan_Front_1 = 0;
int mag_Effort_Fan_Front_2 = 0;
int mag_Effort_Fan_Middle_1 = 0;
int mag_Effort_Fan_Middle_2 = 0;
int mag_Effort_Fan_Back = 0;
int mag_Effort_Fan_Back_2 = 0;
float FBK_SVGS_x = 0; //CMD_pos_x; //0; // -0.25; // From SVGS
float derivative_pos_x = 0;
float FBK_SVGS_z = 0; //CMD_pos_z; // From SVGS
float derivative_pos_x = 0;
float derivative_pos_z = 0;
float derivative_pos_x_Filtered = 0;
float derivative_pos_z_Filtered = 0;
double previous_error_pos_x = 0;
double previous_FBK_SVGS_x = 0;
double previous_error_pos_z = 0;
double previous_FBK_SVGS_z = 0;
double previous_FBK_SVGS_theta = 0;
unsigned long previous_millis_Pos = 0;
double Time_EMFF_CMD = 0;
double Time_EMFF_mode = 0;
double PrevTime_EMFF_mode = 0;
int flag_IDLE = 0;
int flag_EMFF = 0;
const float Pi = 3.141593;
int flag = 0;
```

// For S8 phone (35ms (> 50ms average for LED)
float A[3][3] = {{1.0, 0.0499, 0}, {0, 0.9984, 0}, {0, 1, 0.04996}, {0, 0, 0.9984, 0}, {0, 0, 0, 0.9984}};
```
```c
float B[6][6] = {{0.0003, 0, 0},
                {0.0019, 0, 0},
                {0, 0.0001, 0},
                {0, 0.0003, 0},
                {0, 0, 0.0001},
                {0, 0, 0.00029}};

float C[6][6] = {{0.5683, 0, 0, 0, 0, 0},
                {1.0, 0, 0, 0, 0, 0},
                {0, 2.14, 0, 0, 0, 0},
                {0, 0, 1.0, 0, 0, 0},
                {0, 0, 0, 2.145, 0, 0},
                {0, 0, 0, 0, 1.0}};

float Xn_hat[6] = {0, 0, 0, 0, 0, 0};
float Xn_hat_previous[6] = {0, 0, 0, 0, 0, 0};
float un[1] = {0, 0};

// Adjust this matrix to the proper KF gain matrix (From Matlab: H_d)
float H[6][6] = {{0.0381, 0.05, 0, 0, 0, 0},
                {0.1088, 0.998, 0, 0, 0, 0},
                {0, 0, 0.1233, 0.0372, 0, 0},
                {0, 0, 0.553, 0.3409, 0, 0},
                {0, 0, 0.1206, 0.0348, 0, 0},
                {0, 0, 0, 0.2733, 0.3656}};

float H[6][6] = {{0.0381, 0.05, 0, 0, 0, 0},
                {0.1088, 0.998, 0, 0, 0, 0},
                {0, 0, 0.1233, 0.0372, 0, 0},
                {0, 0, 0.553, 0.3409, 0, 0},
                {0, 0, 0.1206, 0.0348, 0, 0},
                {0, 0, 0, 0.2733, 0.3656}};

float theta_hat = Xn_hat[1];
float theta_dot_hat = Xn_hat[2];
float x_hat = Xn_hat[3];
float x_dot_hat = Xn_hat[4];
float y_hat = Xn_hat[5];
float y_dot_hat = Xn_hat[6];
float Xn[6] = {0, 0, 0, 0, 0, 0};
float BH_term[1] = {0, 0, 0, 0, 0};

void setup() {
  Serial.begin(57600);
  Serial.println("Code started!");
  pinMode(8, OUTPUT); digitalWrite(8, HIGH); // Set up DO pin, AND, turn OFF power (master relay is OFF)
  pinMode(13, OUTPUT); digitalWrite(13, HIGH); // Set up DO pin, AND, turn OFF air in airbearing (2nd relay is OFF)
  Serial.println("Main power & air are: OFF");
  IMU_BNO055_setup();
  Data_Timer_setup();
  Fans_setup();
  SVGS_setup();
  RINGS_EM_setup();
  delay(1000);
  digitalWrite(8, LOW); // turn ON the main relay (main power is ON)
  ESC_arm();
  delay(10000);
  Flag = 0;
  SVGS_i = 0;
  SensorFlag = 0;
  Time_EMFF_mode = 0;
  Prev_Time_EMFF_mode = 0;
  Previous_Time = micros();
  SVGS_Saved_TimeStamp_Previous = micros();
}

void loop() {
  IMU_Gyro();
  // IMU_Angle_from_Gyro();
  svgs_data(); // Start acquiring feedback signals from SVGS (X: x_pos | Y: y_pos | Z: z_pos | Yaw: )
  if (SVGS_Saved == 1) {
    Data_Time = (micros() - SVGS_Saved_TimeStamp_Previous) / 1000000;
  }
  
  if (SensorFlag == 1) {
    IMU_Angle_from_Gyro();
  }
  
}  
```

if (Flag == 0) {
  IDLE_mode();
}
else if (Flag == 1) {
  CMD_function();
  LQ_servo(); // Compute LQ-servo optimal control input for all 3-DOF
  Fans_Effort(); // Turn ON attitude controller
  Fans_Effort_pos_x(); // Turn ON x axes controller
  Fans_Effort_pos_z(); // Turn ON z axes controller
  // Test_Period(); // End the test experiments after a T period
  IDLE_mode();
  IC_EMFF();
  SVGS_i = 1;
}
else if (Flag == 2) { // Activate EMFF (for coaxial maneuver + 2-DOF controller)
  CMD_function();
  LQ_servo(); // Compute LQ-servo optimal control input for all 3-DOF
  Fans_Effort(); // Turn ON attitude controller
  Fans_Effort_pos_x(); // Turn ON x axes controller
  // Fans_Effort_pos_z(); // Turn ON z axes controller
  Fan_Middle_Stop();
  // LQ_servo_pos_z_EMFF();
  SMC_u(); // Activate SMC for EMFF
  EMFF_CMD();
  Test_Period(); // End the test experiments after a T period
  SVGS_i = 1;
  Time_EMFF_mode = Data_Time - Prev_Time_EMFF_mode;
}
else if (Flag == 3) {
  Fan_Stop_Fans();
  flag_IDLE = 0;
  IDLE_mode();
  Flag = 4;
}
else if (Flag == 4) {
  IDLE_mode();
  delay(500);
  digitalWrite(13, HIGH); // turn OFF the 2nd relay (air is OFF)
  delay(1000);
  digitalWrite(3, HIGH); // turn OFF the main relay (main power is OFF)
  IDLE_mode();
}

void Test_Period() {
  double T = 5.0 * 60.0; // Test period, sec.
  // if (Time_EMFF_mode > T){
  //   Flag = 3;
  // }
  if (Data_Time > T) {
    Flag = 3;
  }
}

void IC_EMFF() {
  if (abs(Error) <= 4 * Pi / 180.0 && abs(Error_pos_x) <= 0.04 && abs(Error_pos_z) <= 0.04 &&
      abs(derivative_pos_x_Filtered) <= 0.01 && abs(derivative_pos_z_Filtered) <= 0.01 &&
      abs(derivative_pos_theta) <= 0.1) {
    Flag = 2;
    EMFF_mode();
    Fan_Middle_Stop();
    Time_EMFF_CMD = Data_Time;
    Prev_Time_EMFF_mode = Data_Time;
  }
}

void EMFF_CMD() {
  if (Data_Time - Time_EMFF_CMD >= 0.08) { // Change the time to desire freq. of EMF CMD, in seconds
      Send_EMFF_DutyCycle_CMD(); // Send EMF current CMD every a specific period
      Time_EMFF_CMD = Data_Time; // reset the timer (Freq. of EMFF CMD)
  }
}
3.2.2 CMD

```c
void CMD_function()
{
    // 1- Constant:
    CMD = 0;
    CMD_d = 0;
    CMD_dd = 0;

    // 2- SINE WAVE:
    float Amp = 45;
    float Offset = 45;
    float Freq = 10; // in Hz
    CMD = Amp * sin(Freq*(Pi/180)*t+ Offset);
    CMD_d = Amp * Freq*(Pi/180) * cos(Freq*(Pi/180)*t + Offset);
    CMD_dd = Amp * (Freq*(Pi/180))^2 * sin(Freq*(Pi/180)*t + Offset);

    // 2- SINE WAVE_modified:
    float Amp = 45;
    float Offset = 0;
    float Freq = 10; // in Hz
    CMD = Amp * (1 - cos(Freq * (Pi/180) * Data_Time + Offset));
    CMD_d = Amp * Freq*(Pi/180) * sin(Freq*(Pi/180)* Data_Time + Offset);
    CMD_dd = Amp * (Freq*(Pi/180))*(Freq*(Pi/180)) * cos(Freq*(Pi/180)*Data_Time+ Offset);

    // 3- Square WAVE
    if (Data_Time >= 20 && Data_Time < 60){
        CMD = 90;
        CMD_d = 0;
        CMD_dd = 0;
    } else if (Data_Time >= 60){
        CMD = 0;
        CMD_d = 0;
        CMD_dd = 0;
    }

    // 4- Triangular WAVE

    // 5- Step signal
    if (Data_Time >= 20){
        CMD = 90;
        CMD_d = 0;
        CMD_dd = 0;
    } else if (Data_Time >= 60){
        CMD = 0;
        CMD_d = 0;
        CMD_dd = 0;
    }
}

void CMD_function_Position()
{
    // 1- Constant:
    CMD_pos_x = 0;
    CMD_pos_z = 0.7; //0.8; //1.0;

    // For Square wave CMD :
    if (Time_EMFF_mode < 40){
        CMD_pos_z = 0.7;
    } else if (Time_EMFF_mode >= 40 && Time_EMFF_mode < 100){
        CMD_pos_z = 0.9;
    } else if (Time_EMFF_mode >= 100){
        CMD_pos_z = 0.7;
    } else if (Time_EMFF_mode >= 120 && Time_EMFF_mode < 160){
        CMD_pos_z = 1.0;
    }
}```
```c
else if (Time_EMFF_mode >= 160 && Time_EMFF_mode < 200){
    CMD_pos_z = 0.9;
}
else if (Time_EMFF_mode >= 200 && Time_EMFF_mode < 240){
    CMD_pos_z = 0.8;
}
else if (Time_EMFF_mode >= 240){
    CMD_pos_z = 0.7;
}

// 2nd Staircase CMD from 0.6 to 1m:
//        if (Time_EMFF_mode < 60){
//          CMD_pos_z = 0.6;
//        } else if (Time_EMFF_mode >= 60 && Time_EMFF_mode < 120){
//          CMD_pos_z = 0.8;
//        } else if (Time_EMFF_mode >= 120 && Time_EMFF_mode < 180){
//          CMD_pos_z = 1.0;
//        } else if (Time_EMFF_mode >= 180 && Time_EMFF_mode < 240){
//          CMD_pos_z = 0.8;
//        } else if (Time_EMFF_mode >= 240){
//          CMD_pos_z = 0.6;
//        }

////////////////////////////////////////////////////////////////////////
// 3- Square WAVE (x-axis ONLY)
//        if (Data_Time < 20){
//          CMD_pos_x = -.25; CMD_pos_z = 1.5;
//        } else if (Data_Time >= 20 && Data_Time < 60){
//          CMD_pos_x = .25; // in mm CMD_pos_z = 1.5;
//        } else if (Data_Time >= 60){
//          CMD_pos_x = -.25;
//          CMD_pos_z = 1.5;
//        }

////////////////////////////////////////////////////////////////////////
// 4- Square WAVE (x and z axes)
//        if (Data_Time < 20){
//          CMD_pos_x = -.25; CMD_pos_z = 1.35;
//        } else if (Data_Time >= 20 && Data_Time < 60){
//          CMD_pos_x = .25; // in mm CMD_pos_z = 1.65;
//        } else if (Data_Time >= 60){
//          CMD_pos_x = -.25;
//          CMD_pos_z = 1.35;
//        }
```
3.2.3 DataLogger

```cpp
#include <SD.h> // includes the SD card library (in Arduino Folder)
#define chipSelect 4 // Set Pin 4 as chip select pin for the SD card
String File_name = "data"; // Name of the saved file.. or use: "dat.csv";
Note, Short names only
String File_format = ".txt"; // Excel format ".csv" text format
".txt"
String File_NAMEdotFORMAT = File_name + File_format; // Combining the file name and
format into single string

void Data_LOGGER_setup() {
    Serial.println("Initializing the SD card >> "); // set chip select PIN as OUTPUT. Required also for:
    pinMode(chipSelect, OUTPUT);
    SD.h library.
    if (SD.begin(chipSelect)) {
        Serial.println("NO SD CARD FOUND! check if card is installed, Then >> (reset) button ");
        return; // exit the setup function. This quits the void setup()
    }
    and the program will jump to the void loop().
    if (SD.exists(File_NAMEdotFORMAT)) {
        // Check if the datalog.txt file is already on
        the disk
        SD.remove(File_NAMEdotFORMAT); // delete it. This prevents that the data is
        appended to an already existing file.
        // Set Headers for the data in the text file.
        Set_Headers();
        Serial.println("SD card found >> "); // otherwise, card was successfully initialized.
        Set_Headers();
    }

    void Set_Headers (){
        String HeaderString = "";
        HeaderString = String("Data_Time") + String(\"\t\") + String("CMD_theta") + String(\"\t\") +
            String("FBK_SVGS_roll") + String(\"\t\") + String("derivative_pos_theta") + String(\"\t\") + String("Effort_theta")
            + String(\"\t\") + String("FBK_SVGS_x") + String(\"\t\") + String("FBK_SVGS_z") + String(\"\t\") + String("derivative_pos_x")
                + String(\"\t\") + String("derivative_pos_z") + String(\"\t\") + String("FBK_SVGS_z") + String(\"\t\") + String("FBK_SVGS_x")
                    + String(\"\t\") + String("derivative_pos_z") + String(\"\t\") + String("FBK_SVGS_roll") + String(\"\t\") + String("FBK_SVGS_z") + String(\"\t\") + String("FBK_SVGS_x")
                        + String(\"\t\") + String("derivative_pos_z") + String(\"\t\") + String("FBK_IMU_yaw_dot") + String(\"\t\") + String("Flag") +
                            //String("Time, s") + String(\"\t\") + String("FBK_SVGS_roll") + String(\"\t\") + String("FBK_SVGS_z") + String(\"\t\") + String("FBK_SVGS_x") + String(\"\t\") + String("derivative_pos_z")
                                + String(\"\t\") + String("FBK_SVGS_z") + String(\"\t\") + String("FBK_SVGS_x") + String(\"\t\") + String("mag_Effort_pos_sin")
                                   );
    File dataFile = SD.open(File_NAMEdotFORMAT, FILE_WRITE);
    if (dataFile) {
        dataFile.println(HeaderString);
        dataFile.close();
        Serial.println("Headers were created >> "); Serial.println(" Data Logging setup >> done!");
        // Print Headers to the Arduino Serial window also.
    }

    void Saving_Data() {
        // DataString = String(Data_Time) + String(\"\t\") + String(FBK_IMU_yaw) + String(\"\t\") + String(FBK_IMU_yaw_dot) + String(\"\t\") + String(mag_Effort_pos_sin);
        // Serial.println(DataString);
        File dataFile = SD.open(File_NAMEdotFORMAT, FILE_WRITE); //open a file name ".txt. FILE_WRITE
        mode specifies to append the string at the end of the file
        if (dataFile) {
            // if the file is available, write to it
            if (datafile' is returned 1 if SD.open was successful.
                datafile.println(DataString); //print the concatenated data string and
                finish the line with a carriage return (println adds the CR automatically after printing the string)
                dataFile.close(); //close the file. IT is a good idea always open/close a file before and after writing to it. That way, if someone removes the card the file is most
### 3.2.4 Fan

```cpp
#include <Servo.h>
#include <math.h>

// For ducted fan
#define Fan_Top_PinNr_1    12
#define Fan_Top_PinNr_2    11
#define Fan_Middle_PinNr_1 10
#define Fan_Middle_PinNr_2 9
#define Fan_Left_PinNr_1   8
#define Fan_Left_PinNr_2   7
#define Fan_Right_PinNr_1  6
#define Fan_Right_PinNr_2  5
#define Fan_Top_arm        40 //900
#define Fan_ALL_ZERO       990

Servo Fan_Top_1, Fan_Top_2, Fan_Middle_1, Fan_Middle_2, Fan_Right_1, Fan_Right_2, Fan_Left_1, Fan_Left_2;

int Fan_Top_1_Offset = 1017;
int Fan_Top_2_Offset = 1015;
int Fan_Middle_1_Offset = 1020;
int Fan_Middle_2_Offset = 1015;
int Fan_Left_1_Offset  = 1016; //1018; //1017;
int Fan_Left_2_Offset  = 1003;
int Fan_Right_1_Offset = 1016; //1018;
int Fan_Right_2_Offset = 1003;
float mag_Effort = 0;

void Fans_setup()
{
    Fan_Top_1.attach(Fan_Top_PinNr_1);
    Fan_Top_2.attach(Fan_Top_PinNr_2);
    Fan_Middle_1.attach(Fan_Middle_PinNr_1);
    Fan_Middle_2.attach(Fan_Middle_PinNr_2);
    Fan_Right_1.attach(Fan_Right_PinNr_1);
    Fan_Right_2.attach(Fan_Right_PinNr_2);
    Fan_Left_1.attach(Fan_Left_PinNr_1);
    Fan_Left_2.attach(Fan_Left_PinNr_2);
    Serial.println("Arm signal was sent ");
}

void ESC_arm()
{
    Fan_Top_1.write(Fan_Top_arm);
    Fan_Top_2.write(Fan_Top_arm);
    Fan_Middle_1.write(Fan_Top_arm);
    Fan_Middle_2.write(Fan_Top_arm);
    Fan_Right_1.write(Fan_Top_arm);
    Fan_Right_2.write(Fan_Top_arm);
    Fan_Left_1.write(Fan_Top_arm);
    Fan_Left_2.write(Fan_Top_arm);
    Serial.println("Arm signal was sent ");
}

void Fan_Stop_Fans()
{
    Fan_Top_1.write(Fan_Top_arm);
    Fan_Top_2.write(Fan_Top_arm);
    Fan_Middle_1.write(Fan_Top_arm);
    Fan_Middle_2.write(Fan_Top_arm);
    Fan_Right_1.write(Fan_Top_arm);
    Fan_Right_2.write(Fan_Top_arm);
    Fan_Left_1.write(Fan_Top_arm);
    Fan_Left_2.write(Fan_Top_arm);
}

void Fan_Middle_Stop()
{
    Fan_Middle_1.write(Fan_Top_arm);
    Fan_Middle_2.write(Fan_Top_arm);
}

void Fans_Effort()
{
    mag_Effort = abs(Effort); // We need just the magnitude.
    mag_Effort = mag_Effort * 10;
    if (mag_Effort >= 50) { // Apply a "safety" saturation function to prevent the fans from rotating too HIGH
        mag_Effort = 50;
    }
    // FOR CCW and CW directions:
    if (Effort < 0){
        Fan_Right_2.writeMicroseconds(Fan_Right_2_Offset);
        Fan_Left_2.writeMicroseconds(Fan_Left_2_Offset);
        Fan_Right_1.writeMicroseconds(mag_Effort + Fan_Right_1_Offset);
        Fan_Left_1.writeMicroseconds(mag_Effort + Fan_Left_1_Offset);
    }
    if (Effort > 0){
        Fan_Right_2.writeMicroseconds(Fan_Right_2_Offset);
        Fan_Left_2.writeMicroseconds(Fan_Left_2_Offset);
        Fan_Right_1.writeMicroseconds(mag_Effort + Fan_Right_1_Offset);
        Fan_Left_1.writeMicroseconds(mag_Effort + Fan_Left_1_Offset);
    }
}
```
} else if (Effort >= 0) {
    Fan_Right_1.writeMicroseconds(Fan_Right_1_Offset);
    Fan_Left_1.writeMicroseconds(Fan_Left_1_Offset);
    Fan_Right_2.writeMicroseconds(mag_Effort + Fan_Right_2_Offset);
    Fan_Left_2.writeMicroseconds(mag_Effort + Fan_Left_2_Offset);
}

void Fans_Effort_pos() {
    Fans_Effort_pos_x();
    Fans_Effort_pos_z();
}

void Fans_Effort_pos_x() {
    float mag_Effort_pos_x = abs(Effort_pos_x);  // We need just the magnitude.
    mag_Effort_pos_x = mag_Effort_pos_x * 10;   // Apply a "safety" saturation function to prevent the fans from rotating too HIGH
    if (mag_Effort_pos_x >= 100) {
        mag_Effort_pos_x = 100;
    }
    // FOR Right and Left directions:
    if (Effort_pos_x < 0) {
        Fan_Top_2.writeMicroseconds(Fan_Top_2_Offset);
        Fan_Top_1.writeMicroseconds(mag_Effort_pos_x + Fan_Top_1_Offset);
    } else if (Effort_pos_x >= 0) {
        Fan_Top_1.writeMicroseconds(Fan_Top_1_Offset);
        Fan_Top_2.writeMicroseconds(mag_Effort_pos_x + Fan_Top_2_Offset);
    }
}

void Fans_Effort_pos_z() {
    float mag_Effort_pos_z = abs(Effort_pos_z);  // We need just the magnitude.
    mag_Effort_pos_z = mag_Effort_pos_z * 10;   // Apply a "safety" saturation function to prevent the fans from rotating too HIGH
    if (mag_Effort_pos_z >= 100) {
        mag_Effort_pos_z = 100;
    }
    // FOR Right and Left directions:
    if (Effort_pos_z < 0) {
        Fan_Middle_2.writeMicroseconds(Fan_Middle_2_Offset);
        Fan_Middle_1.writeMicroseconds(mag_Effort_pos_z + Fan_Middle_1_Offset);
    } else if (Effort_pos_z >= 0) {
        Fan_Middle_1.writeMicroseconds(Fan_Middle_1_Offset);
        Fan_Middle_2.writeMicroseconds(mag_Effort_pos_z + Fan_Middle_2_Offset);
    }
}
3.2.5 IMU_BNO055

```c
#include <Arduino.h>
#include <Wire.h>
#define BNO055_ADDRESS 0x28
#define BNO055_CHIP_ID_REG 0x00
#define BNO055_OPR_MODE 0x3D // To select the operation mode (Register address 0x3D) default value: 0x1C
#define BNO055_AMG 0x07 // Accelerometer/Gyro/Magnemometer operation mode
#define BNO055_ACC_GYRO 0x05 // Accelerometer/Gyro only operation mode
#define BNO055_GYR_Config_0 0x0A
#define BNO055_GYR_Config_1 0x0B
#define BNO055_ACC_Config_0 0x08
#define BNO055_PAGE_ID 0x07
#define GYR_DATA_X_LSB 0x14
#define GYR_DATA_X_MSB 0x15
#define GYR_DATA_Y_LSB 0x16
#define GYR_DATA_Y_MSB 0x17
#define GYR_DATA_Z_LSB 0x18
#define GYR_DATA_Z_MSB 0x19
#define gyro_z_offset 0.06
#define dps125  0b100
#define dps2000 0b000
#define BNO055_UNIT_SEL 0x3B
#define HZ32  0b111000
#define HZ64  0b110000
#define HZ230 0b001000
// Accelerometer
#define ACC_DATA_X_LSB 0x08
#define ACC_DATA_X_MSB 0x09
#define ACC_DATA_Y_LSB 0x0A
#define ACC_DATA_Y_MSB 0x0B
#define ACC_DATA_Z_LSB 0x0C
#define ACC_DATA_Z_MSB 0x0D
#define acc_x_offset  0
#define acc_y_offset  0
#define acc_z_offset -0.62
unsigned long previousMicros = 0;
double dt = 0;

void IMU_BNO055_setup(){
  Serial.print("(2) Connecting to IMU BNO055... >> ");
  Reset_IMU_BNO055(); // Reset the power from the IMU. This will trigger a
  Wire.begin(); // Start I2C
  Wire.setClock(400000L); // Set clock to 400khz
  Check_IMU_BNO055_exists(); delay(100); // Check if the BNO055 detected.
  setGyr(); delay(100); // set Operation Mode
  setOprMode();
  IMU_Gyro_Z(); // For unknown reason, the first data of Gyro (-8.06) is wrong! So take it here in the setup (8, it won't drift the angle calc.)
  Serial.println("    >> IMU setup done!");
}

void Reset_IMU_BNO055(){
  pinMode(2, OUTPUT);
  // Accelerometer
  Wire.beginTransmission(BNO055_ADDRESS); // transmit to device #4
  Wire.write(BNO055_CHIP_ID_REG); // Tell slave we need to read lbyte from the
  Wire.requestFrom(BNO055_ADDRESS, 1); // current register
  byte result = Wire.read(); // Wire.endTransmission(); // stop transmitting
  if (result == 0xA0){
    Serial.print("IMU Device: BNO055 FOUND");
  }else{
    Serial.println("device not found");
    while (true);
  }
}

void setGyr(){
  Wire.beginTransmission(BNO055_ADDRESS);
```
void set_Operation_Mode()
{
    Wire.beginTransmission(BNO055_ADDRESS);
    Wire.write(BNO055_OPR_MODE);
    Wire.endTransmission();
}

void setUnit()
{
    Wire.beginTransmission(BNO055_ADDRESS);
    Wire.write(BNO055_UNIT_SEL);
    Wire.write(0b000);
    Wire.endTransmission();
}

void IMU_Gyro_Z()
{
    Wire.beginTransmission(BNO055_ADDRESS);
    Wire.write(GYR_DATA_Z_LSB);
    Wire.requestFrom(BNO055_ADDRESS, 2);
    byte lsb = Wire.read();
    byte msb = Wire.read();
    Wire.endTransmission();
    if (msb == 0xFF && lsb <= 30) {
        // Following if statement: to delete the random spikes
        occur within the gyro raw data
        msb = 0x00;
    } else if (msb == 0x00 && lsb >= 220) {
        msb = 0xFF;
    }

    int16_t data = (msb << 8) | lsb;
    double dataDouble = (double)data;
    FBK_IMU_yaw_dot = dataDouble / 16.0;  // RAW data (conversion factor)
    FBK_IMU_yaw_dot = FBK_IMU_yaw_dot - gyro_z_offset;
    // Serial.print("FBK IMU Gyro Z: "); Serial.println(FBK_IMU_yaw_dot);
    if (abs(FBK_IMU_yaw_dot) <= 0.25) {
        // Dead-band function to remove the noise, which makes yaw angle
        drift with time
        FBK_IMU_yaw_dot = 0;
    }
    FBK_IMU_yaw = dt * FBK_IMU_yaw_dot;  // x += y; >> equivalent to the expression x
                                           // = x + y;
}

void IMU_Acc_X()
{
    Wire.beginTransmission(BNO055_ADDRESS);
    Wire.write(ACC_DATA_X_LSB);
    Wire.requestFrom(BNO055_ADDRESS, 2);
    byte lsb = Wire.read();
    byte msb = Wire.read();
    Wire.endTransmission();
    int16_t data = (msb << 8) | lsb;
    double dataDouble = (double)data;
    FBK_IMU_Acc_x = dataDouble / 100.0;  // RAW data (conversion factor)
    FBK_IMU_Acc_x = FBK_IMU_Acc_x - acc_x_offset;
    if (abs(FBK_IMU_Acc_x) <= 0.3) {
        // Dead-band function to remove the noise
        FBK_IMU_Acc_x = 0;
    }
}

void IMU_Acc_Z()
{
    Wire.beginTransmission(BNO055_ADDRESS);
    Wire.write(ACC_DATA_Z_LSB);
    Wire.requestFrom(BNO055_ADDRESS, 2);
    byte lsb = Wire.read();
byte msb = Wire.read();
Wire.endTransmission();

int16_t data = (msb << 8) | lsb;
double dataDouble = (double)data;
FBK_IMU_Acc_z = dataDouble / 190.0; // RAW data (conversion factor)
FBK_IMU_Acc_z = FBK_IMU_Acc_z + acc_z_offset;

if (abs(FBK_IMU_Acc_z) <= 0.3) { // Dead-band function to remove the noise
    FBK_IMU_Acc_z = 0;
}

void IMU_Velocity_from_Accelerometer(){
    // Calculate dt
    unsigned long currMicros = micros();
dt = (double)(currMicros - prevMicros) / 1000000;
prevMicros = currMicros;

    // Calculate current velocity based on Accelerometer
    FBK_IMU_Velocity_x += dt * FBK_IMU_Acc_x; // x += y; >> equivalent to the
    FBK_IMU_Velocity_z += dt * FBK_IMU_Acc_z;

}
3.2.6 LQ_servo

```c
void LQ_servo(){
    double G1 = 57.7350;  //70.7107;  //35.3553;  //57.7350;
    double G2 = 134.4968;  //152.5813;  //91.4643;  //134.4968;
    // Effort = Error * G1;
    Effort = (Error * G1) - (G2 * FBK_IMU_yaw_dot);
    if (Effort >= 5) {
        // output saturation function
        Effort = 5;
    } else if (Effort <= -5) {
        Effort = -5;
    }
    LQ_servo_pos_x();
    LQ_servo_pos_z();
}

void LQ_servo_pos_x(){
    // For x-axis
    double G1 = 115.4701; //GOOD: 115.4701;  //100.0;    //91.2871;  //91.2871;  //64.5497;  //64.5497; //74.5356;  //64.5497;
    double G2 = 444.2100; //GOOD: 384.5561;  //406.3568; //326.6282;  //300.4414; //387.4873; //272.8606;  //310.5376; //274.7002;
    //  Effort_pos_x = (Error_pos_x * G1) - (G2 * derivative_pos_x);
    Effort_pos_x = (Error_pos_x * G1) - (G2 * derivative_pos_x_Filtered);
    if (Effort_pos_x >= 10) {
        // output saturation function
        Effort_pos_x = 10;
    } else if (Effort_pos_x <= -10) {
        Effort_pos_x = -10;
    }
}

void LQ_servo_pos_z(){
    double G1 = 115.4701; //GOOD: 115.4701;  //91.2871;  //91.2871;  //64.5497;  //64.5497; //74.5356;  //64.5497;
    double G2 = 448.7709; //GOOD: 389.9369;  //330.2965; //304.5015;  //290.2441; //275.8110; //313.5539; //274.7002;
    //  Effort_pos_z = (Error_pos_z * G1) - (G2 * derivative_pos_z);
    Effort_pos_z = (Error_pos_z * G1) - (G2 * derivative_pos_z_Filtered);
    if (Effort_pos_z >= 10) {
        // output saturation function
        Effort_pos_z = 10;
    } else if (Effort_pos_z <= -10) {
        Effort_pos_z = -10;
    }
}

void LQ_servo_pos_z_EMFF(){
    double G1 = 156.2;  //200;  //156.2;  //100.0;  //Best: 156.2;
    double G2 = 2282.7;  //2510.4;  //2282.7;    //703.0754;  //976.8104;  //Best: 2282.7;
    GainScheduling(); // Activate this line ONLY for Gain Scheduling controller (Staircase CMD)
    //  Effort_pos_z = (Error_pos_z * G1) - (G2 * derivative_pos_z);
    Effort_pos_z = (Error_pos_z * G1) - (G2 * z_dot_hat);
    if (Effort_pos_z >= 15) {
        // output saturation function
        Effort_pos_z = 15;
    } else if (Effort_pos_z <= -15) {
        Effort_pos_z = -15;
    }
}

void GainScheduling(){
    double G1 = 156.2;  //200;  //156.2;  //100.0;  //Best: 156.2;
    double G2 = 2282.7;
    if (FBK_SVGS_z >= 0.65 && FBK_SVGS_z <= 0.75) {
        G1 = 150.9;
        G2 = 2262.0;
    } else if (FBK_SVGS_z > 0.75 && FBK_SVGS_z <= 0.85) {
        G1 = 156.2;
        G2 = 2282.7;
    } else if (FBK_SVGS_z > 0.85 && FBK_SVGS_z <= 0.95) {
        G1 = 160.4;
        G2 = 2313.2;
    } else if (FBK_SVGS_z > 0.95 && FBK_SVGS_z <= 1.05) {
        G1 = 164.0;
        G2 = 2355.4;
    } else {
        G1 = 156.2;
        G2 = 2282.7;
    }
}
```

3.2.7 RINGS_EM

byte IDLE_Packet[9] = {0xFF, 0x02, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0A};
byte EMFF_Packet[9] = {0xFF, 0x02, 0x02, 0x00, 0x00, 0x00, 0x53, 0x0D, 0x0A};
byte EMFF_180_Packet[10] = {0xFF, 0x02, 0x02, 0x00, 0x00, 0x08, 0x53, 0xB4, 0x0D, 0x0A};

int DutyCycle = 0;
byte DutyCycle_Hex = 0;
byte EMFF_DutyCycle_Packet_part1[3] = {0xFF, 0x03, 0x00};
byte EMFF_DutyCycle_Packet_part2[7] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x0D, 0x0A};
byte EMFF_DutyCycle_Packet_part2_180phase[7] = {0xB4, 0x00, 0x00, 0x00, 0x00, 0x0D, 0x0A};

void RINGS_EM_Setup(){
  Serial2.begin (SERIAL_SPEED_RINGS_PIC32);  //Serial.println(" 1) Serial2 setup    >> done   ");
  flag_IDLE = 0;
  flag_EMFF = 0;
}

// DEFINING ALL EMFF MODES:
void IDLE_mode(){
  if (flag_IDLE == 0){
    static int n = 0;
    n = n + 1;
    if (n >= 20){
      Serial2.write(IDLE_Packet, sizeof(IDLE_Packet));
      n = 0;
      // flag_IDLE = 1;
    }
  }
}

void EMFF_mode(){
  if (flag_EMFF == 0){
    Serial2.write(EMFF_Packet, sizeof(EMFF_Packet));
    static int b = 0;
    b = b + 1;
    if (b == 5){
      flag_EMFF = 1;
      // b = 0;
    }
  }
}

void EMFF_180_mode(){
  Serial2.write(EMFF_180_Packet, sizeof(EMFF_180_Packet));
}
```c
void Send_EMFF_DutyCycle_CMD() {

    DutyCycle = abs(Effort_pos_z);
    DutyCycle = 0.2 * (Effort_pos_z * Effort_pos_z) + 1.15 * Effort_pos_z; // Conversion factor from current to duty cycle

    if (DutyCycle <= 0){  // Apply saturation function to DutyCycle
        DutyCycle = 0;
    } else if (DutyCycle >= 50){
        DutyCycle = 50;
    }

    DutyCycle_Hex = (DutyCycle, HEX);
    Serial2.write(EMFF_DutyCycle_Packet_part1,sizeof(EMFF_DutyCycle_Packet_part1));
    Serial2.write(DutyCycle);

    if (Effort_pos_z >= 0){ // To control EM force direction (attraction, or repulsion)
        Serial2.write(EMFF_DutyCycle_Packet_part2,sizeof(EMFF_DutyCycle_Packet_part2));
    } else if (Effort_pos_z < 0){
        Serial2.write(EMFF_DutyCycle_Packet_part2_180phase,sizeof(EMFF_DutyCycle_Packet_part2_180phase));
    }
}
```
3.2.8 SMC

```c
float SMC_effort = 0;
float sign = 0;
float sat = 0;
float s = 0;
float phi = 0.005;
float xd_dot = 0;
float xd_ddot = 0;
float D = 2 / 3;
float eta = 0.02;
float lambda = 0.2;
float M = 14.6;
float mu = 4.0 * Pi / 10000000.0;  // permeability of free space
float n = 100;
float I_A = 13.7;
float R = 0.62 / 2;

void SMC_u() {
    float x = FBK_SVGS_z;
    float x_dot = derivative_pos_z_Filtered;
    float x_delta = (x - CMD_pos_z);  // -1 * Error_pos_z;
    float x_delta_dot = x_dot - xd_dot;
    float b_hat = (3 * mu * n * n * I_A * Pi * (R * R * R * R)) / (2 * M * (x * x * x * x));
    // float F = -0.002 * x_dot;
    s = x_delta_dot + (lambda * x_delta); // xd_dot = (lambda*x_delta);
    float x_r_dot = x_dot - s;
    float k = (1 / (1 - D)) * ((D * abs(x_r_dot)) + eta);
    float u_hat = xd_ddot - (lambda * x_delta_dot);
    satFn();
    // SMC_effort = (1 / b_hat) * (x_r_dot - (k * sat));
    SMC_effort = (: / b_hat) * (u_hat - (k * sat));

    Effort_pos_z = SMC_effort;
    if (Effort_pos_z >= 14) {  // output saturation function
        Effort_pos_z = 14;
    } else if (Effort_pos_z <= -14) {
        Effort_pos_z = -14;
    }
}
```
void satFn() {
    if (abs(s) >= phi) {
        sgnFn();
        sat = sign;
    } else if (abs(s) < phi) {
        sat = s / phi;
    }
    // return sat;
}

void sgnFn() {
    if (s > 0) {
        sign = 1;
    } else if (s < 0) {
        sign = -1;
    } else {
        sign = 0;
    }
    // return sign;
}
3.2.9 SVGS

```c
int i = 0;
uint8_t Packet[32];

void SVGS_setup() {
    Serial1.begin(115200);
    Serial.println(" 3 Connections for Serial 1 started!");
}

void SVGS_data(){
    if (Serial1.available()) { // If there is a data from Serial1 (svgs) available:
        int inByte = Serial1.read(); // Store the income byte in the inByte variable
        Packet[i] = inByte;
        if ((Packet[i-1] == 219 && Packet[i] == 220) { // SLIP
            Packet[i-1] = 192;
            i = i-1;
        }
        if ((Packet[i-1] == 219 && Packet[i] == 221) { // SLIP
            Packet[i-1] = 219;
            i = i-1;
        }
    //--------------------------------------------------------------------- Check proper packet --------
    if (Packet[i-1] == 192 && Packet[i] == 1) {
        Packet[i] = Packet[i-1];
        Packet[i] = Packet[i];
        i = 1;
    } //---------------------------------------------------------------------
    i++;
    if (i<=32 && Packet[0] == 192 && Packet[3] == 192) {
        if (Packet[1]!=0 && Packet[2]!=0 && Packet[5]!=0 && Packet[8]!=0 && Packet[11]!=0 && Packet[14]!=0 && Packet[17]!=0 && Packet[20]!=0 && Packet[23]!=0 && Packet[26]!=0 && Packet[29]!=0 && Packet[32]!=0)
            for (int j=0; j<i; j++){
                if (Packet[3] == 254)
                    Serial.print (Packet[j],HEX);  Serial.print (" ");
                if (Packet[3] == 241)
                    Serial.print ("\n");
                // Serial.print("\n");
                FFB_SVGS_x = *(float*)&x;
                FFB_SVGS_x = FFB_SVGS_x * -1.0;
                Serial.print("x_pos:");  Serial.print (FFB_SVGS_x);
                // Serial.print("\n");
                /* Packet[6];*/
                // Serial.print("y_pos:");  Serial.print (FFB_SVGS_y);
                // Serial.print("\n");
                /* Packet[10];*/
                // Serial.print("z_pos:");  Serial.print (FFB_SVGS_z);
                // Serial.print("\n");
                /* Packet[22];*/
                // Serial.print("yaw:");  Serial.print (FFB_SVGS_yaw);
                // Serial.print("\n");
                /* Packet[14];*/
                // Serial.print("roll:");  Serial.print (FFB_SVGS_roll);
                // Serial.print("\n");
                // for Roll svgs FBF signal to match direction of rotation for IMU!!!!!!
            }
        }
    }
```
void LowPassFilter_Velocity_theta() {
  derivative_pos_theta_filtered = 0.3897 * derivative_pos_theta_filtered + 0.6103 * derivative_pos_theta;   // fc = .6;
}

void LowPassFilter_Velocity_x() {
  derivative_pos_x_filtered = 0.3897 * derivative_pos_x_filtered + 0.6103 * derivative_pos_x;   // fc = .6;
}

void LowPassFilter_Velocity_z() {
  derivative_pos_z_filtered = 0.3897 * derivative_pos_z_filtered + 0.6103 * derivative_pos_z;   // fc = .6;
}

void SVGS_TimeStart() {
  SVGS_TimeStart = currentMillis;
  previous_FBK_SVGS_roll = FBK_SVGS_roll;
  previous_FBK_SVGS_x = FBK_SVGS_x;
  previous_FBK_SVGS_z = FBK_SVGS_z;
  previous_FBK_SVGS_theta = FBK_SVGS_roll;
}

String DataString = String(Data_Time) + String(FBK_SVGS_roll, 4) + String(FBK_SVGS_x, 4) + String(FBK_SVGS_z, 4) + String(Effort_x) + String(Effort_z) + String(CMD_pos_x) + String(CMD_pos_z) + String(Effort_pos_x) + String(Effort_pos_z) + String(CMD_pos_x) + String(CMD_pos_z) + String(Effort_pos_x) + String(Effort_pos_z) + String(theta_hat) + String(theta_dot_hat) + String(x_hat) + String(x_dot_hat) + String(z_hat) + String(z_dot_hat) + String(Flag);

if (SVGS_Started = true) {
  Serial.flush();
  Serial1.flush();
}

if (SVGS_Started == 1 && SVGS_i == 0) {
  digitalWrite(13, LOW);   // turn ON air once 1st SVGS data was received
}
Flag = 1;
SVGS_Started_TimeStamp_Previous = micros();
SVGS_i = 1;
Data_Time = 0;
}

void KalmanFilter(){
// Define the output of the system (6 outputs)
Yn[] = FBK_IMU_yaw_roll;
Yn[] = FBK_IMU_yaw dot; //FBK IMU_yaw dot;
Yn[] = FBK_SVGS_x;
Yn[] = derivative_pos_x_Filtered; //derivative_pos_x ;
Yn[] = FBK_SVGS_z;
Yn[] = derivative_pos_z_Filtered; //derivative_pos_z ;
// Define the control inputs of the system (3 inputs)
un[] = Effort;
un[] = Effort_pos_x;
un[] = Effort_pos_z;

// BH_term (look in Nazir's note for the term BH in the design of Kalman filter notes..)
BH_term[] = (YN[] - (C[][]) * Xn_hat_previous[]) + (C[][]) * Xn_hat_previous[] + (C[][]) * Xn_hat_previous[] + (C[][]) * Xn_hat_previous[] + (C[][]) * Xn_hat_previous[]
BH_term[] = (YN[] - (C[][]) * Xn_hat_previous[]) + (C[][]) * Xn_hat_previous[] + (C[][]) * Xn_hat_previous[] + (C[][]) * Xn_hat_previous[] + (C[][]) * Xn_hat_previous[]
BH_term[] = (YN[] - (C[][]) * Xn_hat_previous[]) + (C[][]) * Xn_hat_previous[] + (C[][]) * Xn_hat_previous[] + (C[][]) * Xn_hat_previous[] + (C[][]) * Xn_hat_previous[]
BH_term[] = (YN[] - (C[][]) * Xn_hat_previous[]) + (C[][]) * Xn_hat_previous[] + (C[][]) * Xn_hat_previous[] + (C[][]) * Xn_hat_previous[] + (C[][]) * Xn_hat_previous[]
BH_term[] = (YN[] - (C[][]) * Xn_hat_previous[]) + (C[][]) * Xn_hat_previous[] + (C[][]) * Xn_hat_previous[] + (C[][]) * Xn_hat_previous[] + (C[][]) * Xn_hat_previous[]

// theta:
Xn_hat[] = (A[][]) * Xn_hat_previous[] + A[][] * Xn_hat_previous[] + A[][] * Xn_hat_previous[] + A[][] * Xn_hat_previous[] + A[][] * Xn_hat_previous[] + A[][] * Xn_hat_previous[] + A[][] * Xn_hat_previous[] + A[][] * Xn_hat_previous[]
+ (B[][]) * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[]
+ (C[][]) * BH_term[] + (C[][]) * BH_term[] + (C[][]) * BH_term[] + (C[][]) * BH_term[] + (C[][]) * BH_term[] + (C[][]) * BH_term[]
+ (C[][]) * BH_term[] + (C[][]) * BH_term[]}

// x dot:
Xn_hat[] = (A[][]) * Xn_hat_previous[] + A[][] * Xn_hat_previous[] + A[][] * Xn_hat_previous[] + A[][] * Xn_hat_previous[] + A[][] * Xn_hat_previous[] + A[][] * Xn_hat_previous[] + A[][] * Xn_hat_previous[] + A[][] * Xn_hat_previous[]
+ (B[][]) * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[] + B[][] * un[]
+ (C[][]) * BH_term[] + (C[][]) * BH_term[] + (C[][]) * BH_term[] + (C[][]) * BH_term[] + (C[][]) * BH_term[] + (C[][]) * BH_term[]
+ (C[][]) * BH_term[] + (C[][]) * BH_term[]

// x dot hat:

36 | Page
\[
X_{n+1} = A \cdot X_n + B \cdot u_n + H \cdot \text{BH}_n
\]

where

- \(X_n\) is the state at time \(n\)
- \(A\) is the state transition matrix
- \(B\) is the input matrix
- \(H\) is the measurement matrix
- \(\text{BH}_n\) is the BH term at time \(n\)
- \(u_n\) is the input at time \(n\)
- \(H \cdot \text{BH}_n\) is a term that depends on the measurement matrix and \(\text{BH}_n\)

The equations for \(X_{n+1}\), \(z\), \(\dot{z}\) are defined recursively for each time step.
3.3 Code for IR triggering circuitry

```c
void setup() {
    Serial.begin(9600);
    Serial2.begin(9600);
    Serial3.begin(9600);

    Serial.println("Serial 0 and 3 started: ");
}

void loop() {
    Send_IR();
    // Read_IR();
    delay(500);
}

void Send_IR(){
    static int inINT = 1;
    if (Serial.available()){
        inINT = Serial.parseFloat();
        // Serial3.write(inINT);
    }
    // Serial3.write("Pulse");
    Serial3.write(inINT);
    // Serial3.write(inINT);
    // Serial3.write(inINT);
    Serial.println(inINT);
}

void Read_IR(){
    if (Serial2.available()){
        // If there is a data from Serial2 (svgs) available:
        int inByte = Serial2.read();
        // Store the income byte in the inByte variable
        static int i = 0;
        i = i + 1;
        Serial.print(i);
        Serial.print('t');
        Serial.print("Received byte: ");
        Serial.print('t');
        Serial.println(inByte);
    }
}
```