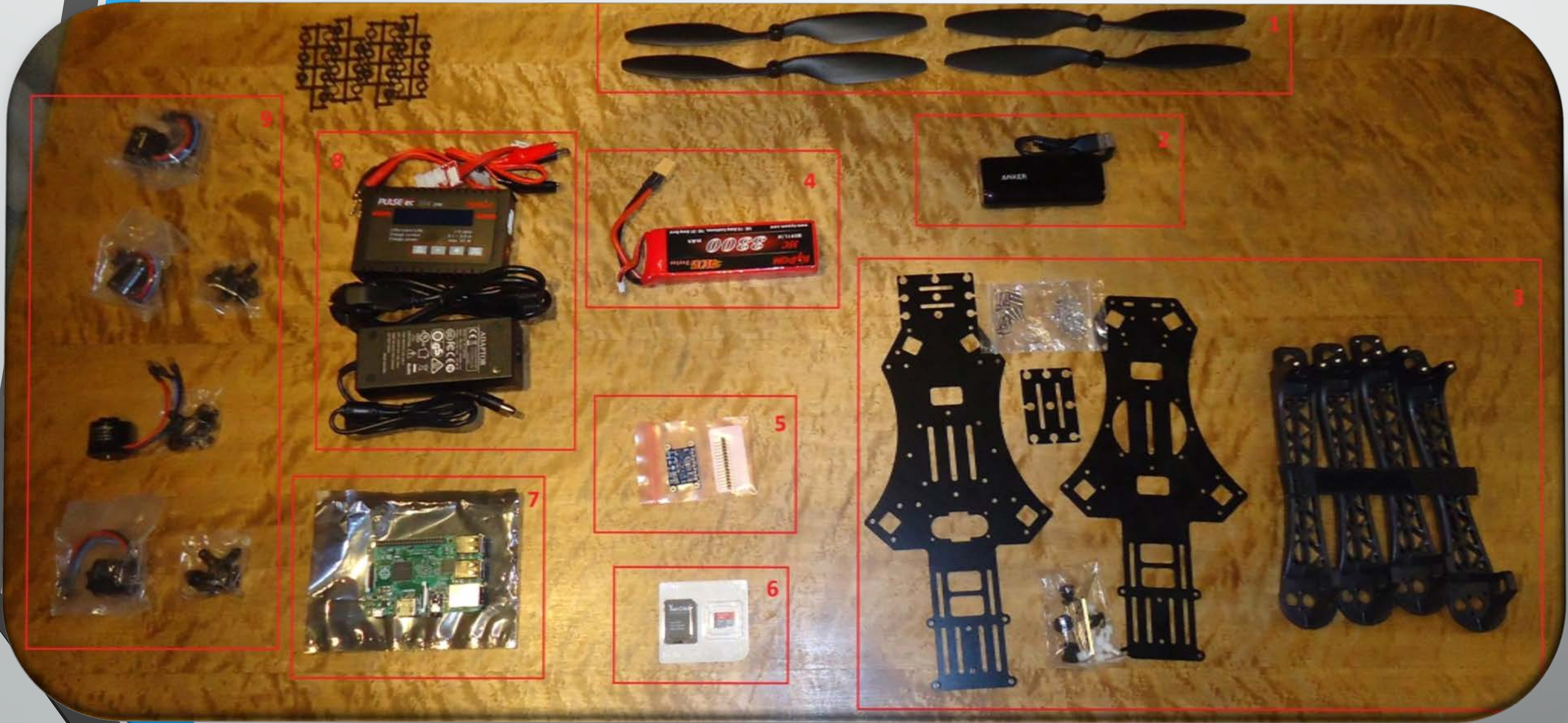
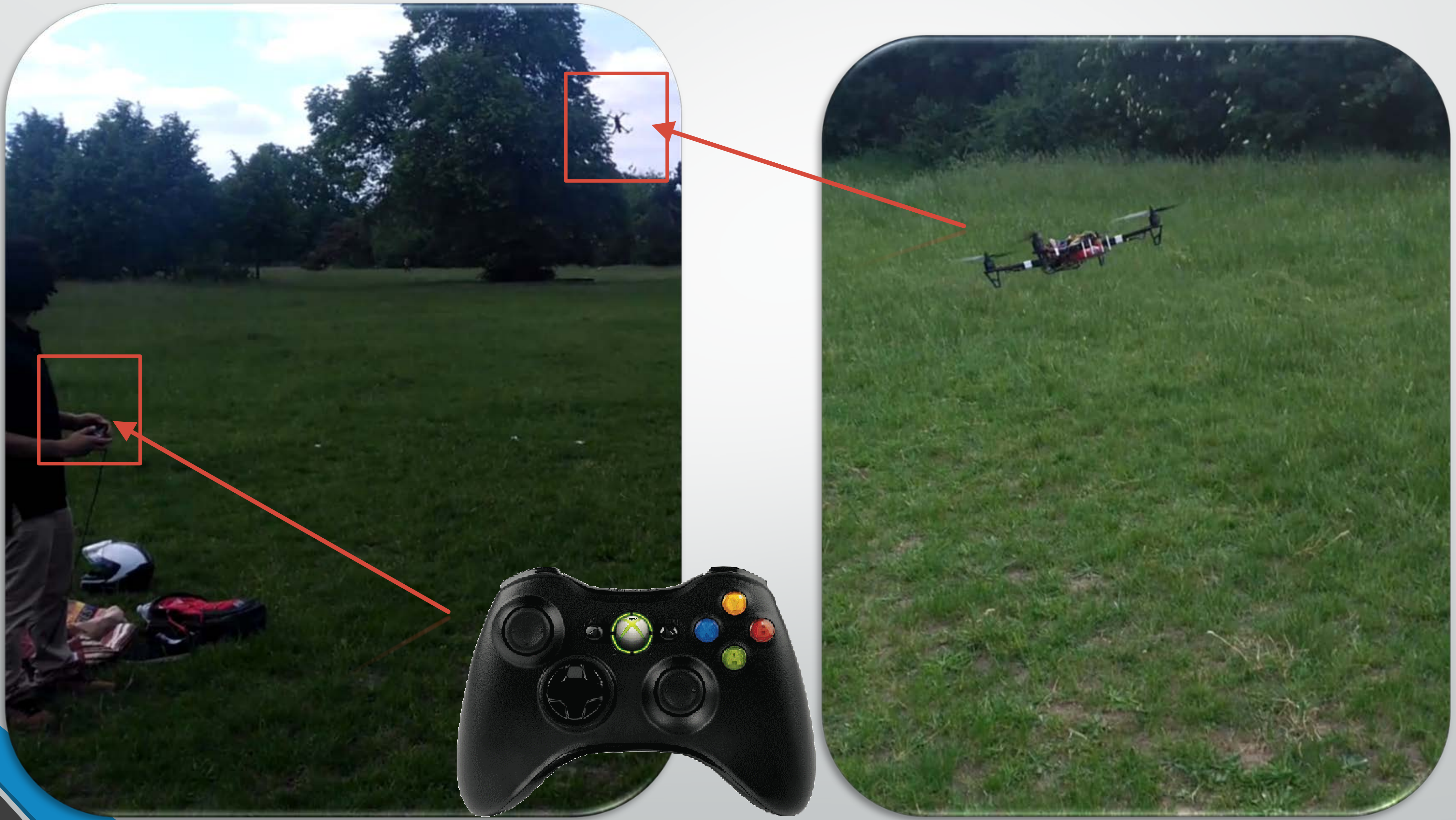
DRONE M1

Quadricoptere PID

Composants

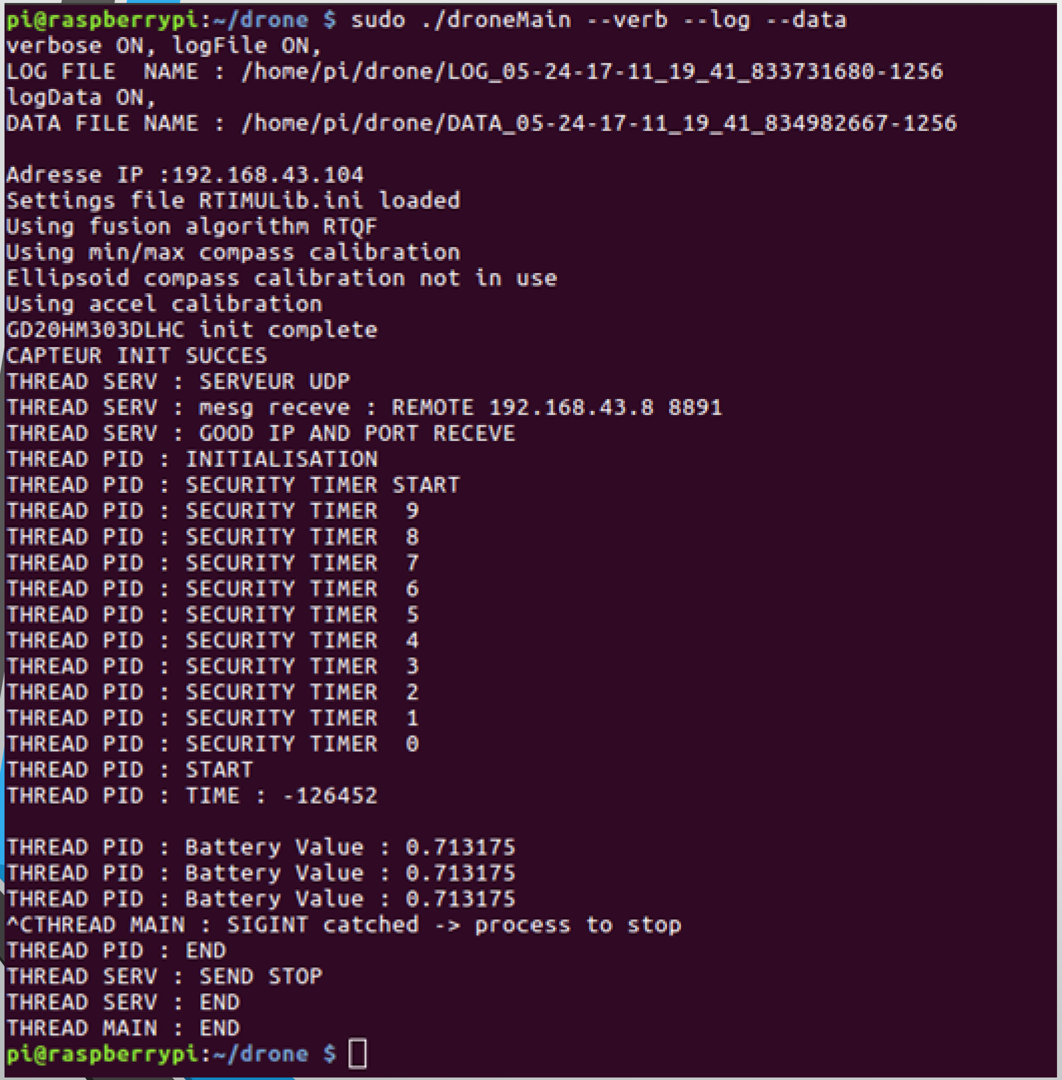


VOL avec Manette

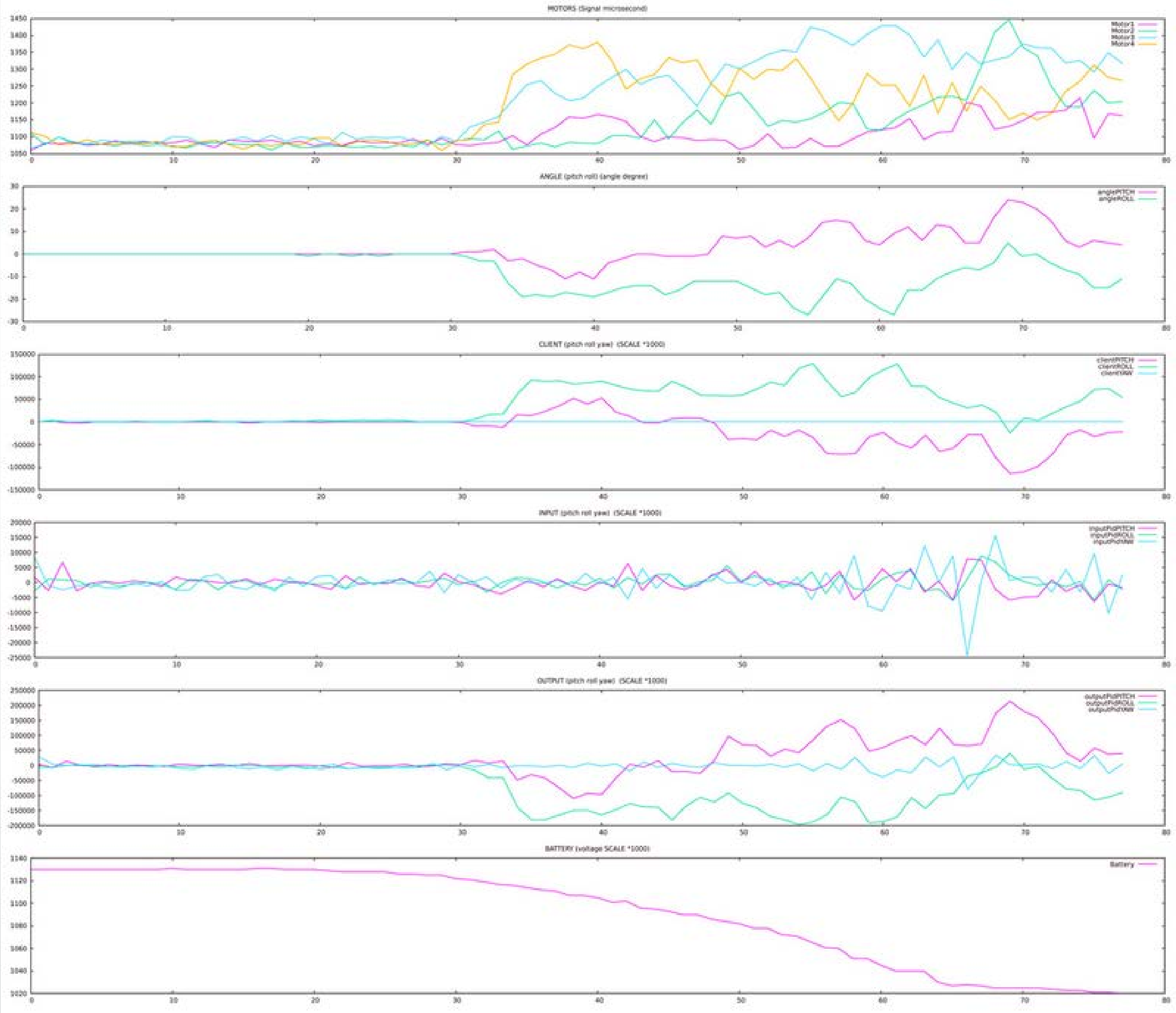


Analyse de Vol

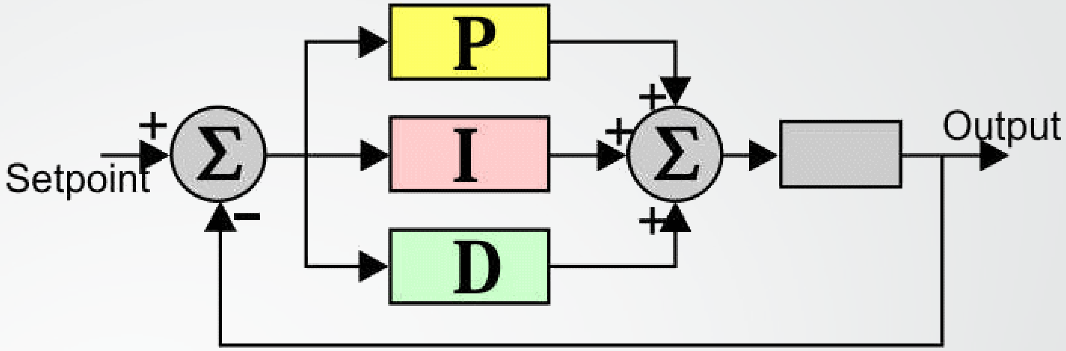
GNUPLOT

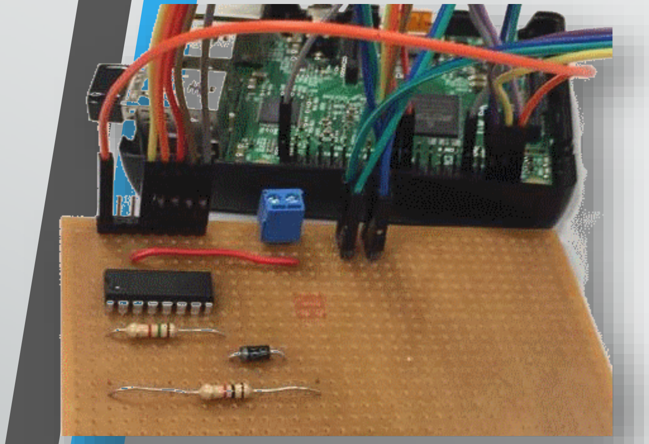


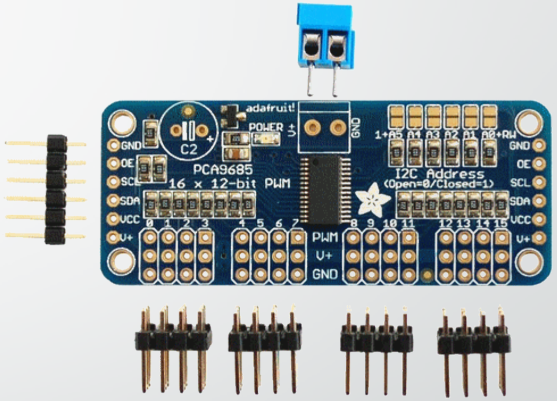
```
pi@raspberrypi:~/drone $ sudo ./droneMain --verb --log --data
verbose ON, logFile ON,
LOG FILE  NAME : /home/pi/drone/LOG_05-24-17-11_19_41_833731680-1256
logData ON,
DATA FILE NAME : /home/pi/drone/DATA_05-24-17-11_19_41_834982667-1256

Adresse IP :192.168.43.104
Settings file RTIMULib.ini loaded
Using fusion algorithm RTQF
Using min/max compass calibration
Ellipsoid compass calibration not in use
Using accel calibration
GD20HM303DLHC init complete
CAPTEUR INIT SUCCES
THREAD SERV : SERVEUR UDP
THREAD SERV : mesg receve : REMOTE 192.168.43.8 8891
THREAD SERV : GOOD IP AND PORT RECEVE
THREAD PID : INITIALISATION
THREAD PID : SECURITY TIMER START
THREAD PID : SECURITY TIMER  9
THREAD PID : SECURITY TIMER  8
THREAD PID : SECURITY TIMER  7
THREAD PID : SECURITY TIMER  6
THREAD PID : SECURITY TIMER  5
THREAD PID : SECURITY TIMER  4
THREAD PID : SECURITY TIMER  3
THREAD PID : SECURITY TIMER  2
THREAD PID : SECURITY TIMER  1
THREAD PID : SECURITY TIMER  0
THREAD PID : START
THREAD PID : TIME : -126452

THREAD PID : Battery Value : 0.713175
THREAD PID : Battery Value : 0.713175
THREAD PID : Battery Value : 0.713175
^CTHREAD MAIN : SIGINT catched -> process to stop
THREAD PID : END
THREAD SERV : SEND STOP
THREAD SERV : END
THREAD MAIN : END
pi@raspberrypi:~/drone $ 
```



Les sous projets majeurs



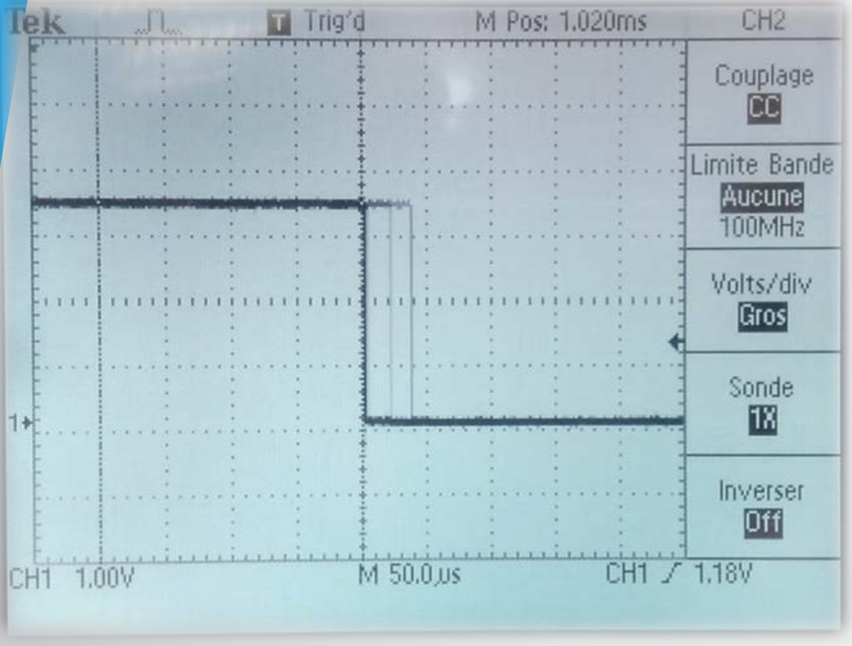




Real-Time
LINUX

...

Le balancier



Tek Trig'd M Pos: 1.020ms CH2

Couplage CC

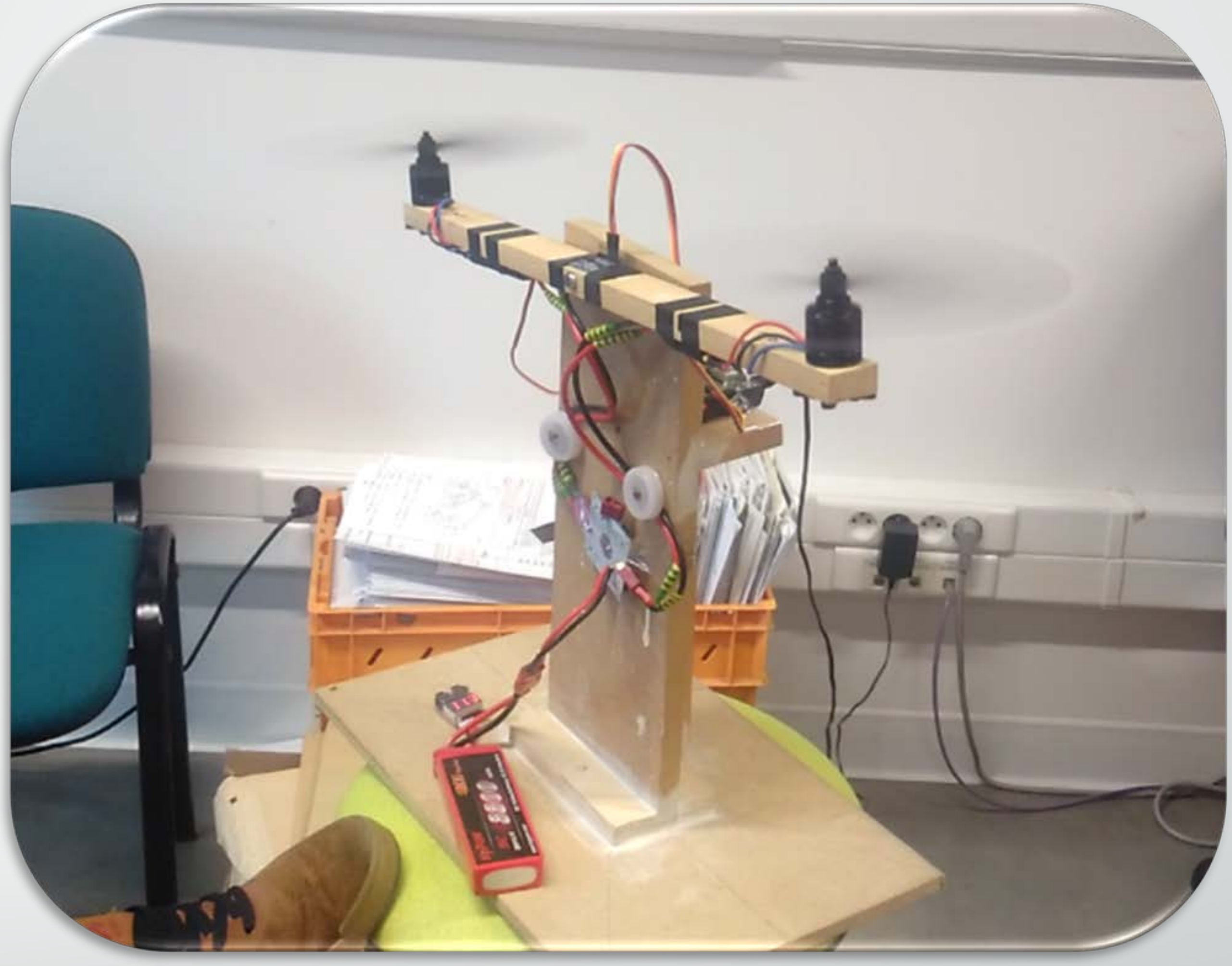
Limite Bande Aucune 100MHz

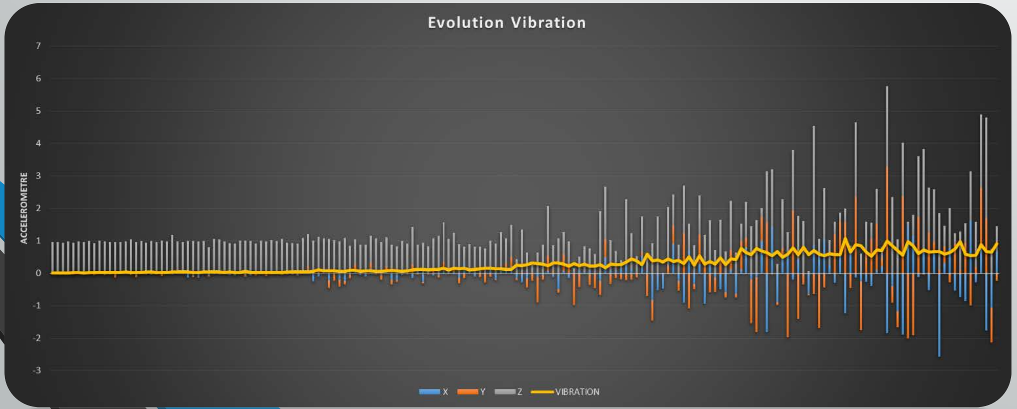
Volts/div Gros

Sonde 1X

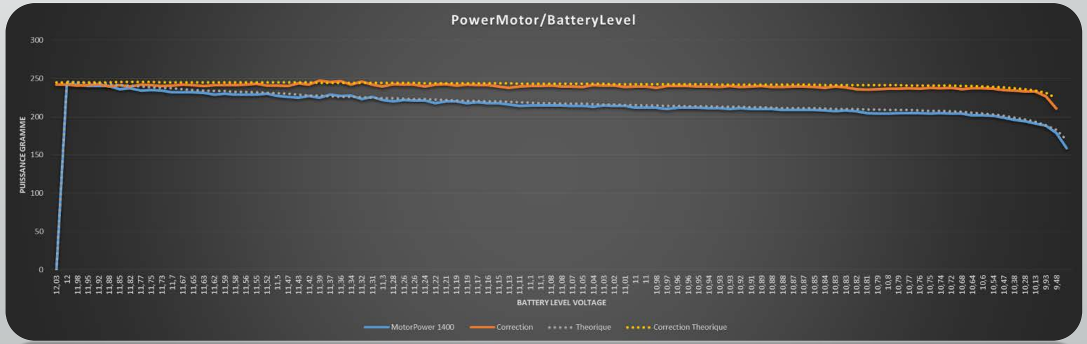
Inverser Off

CH1 1.00V M 50.0µs CH1 ╱ 1.18V



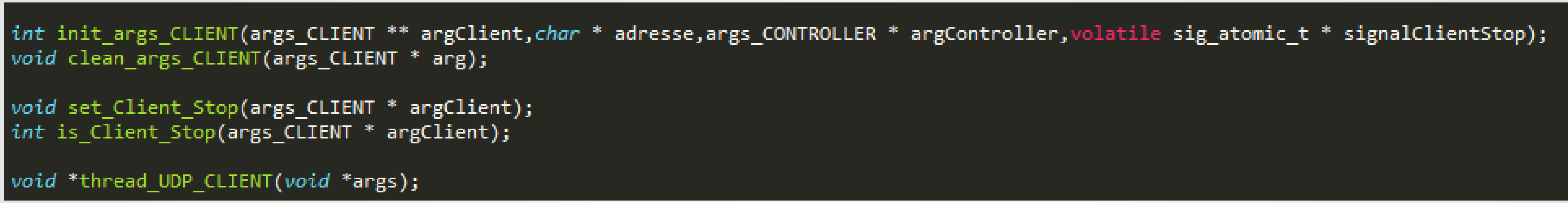


Evolution Vibration

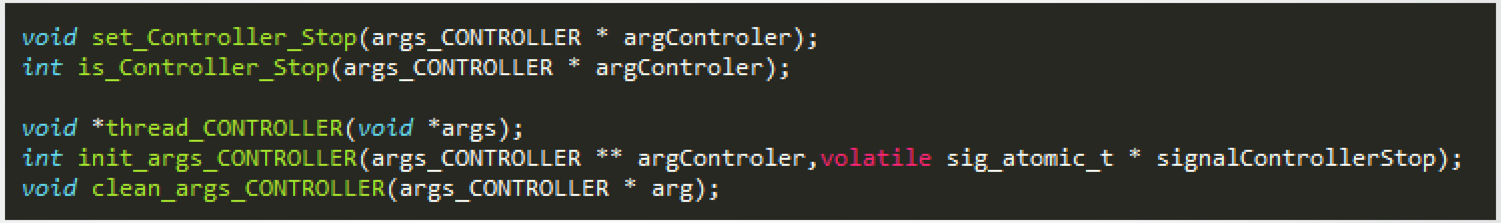


PowerMotor/BatteryLevel

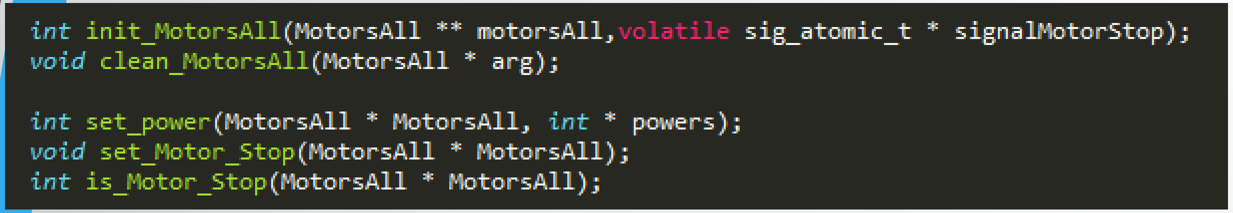
API



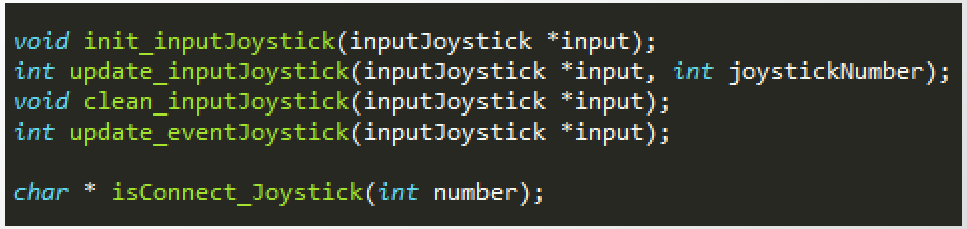
```
int init_args_CLIENT(args_CLIENT ** argClient,char * adresse,args_CONTROLLER * argController,volatile sig_atomic_t * signalClientStop);
void clean_args_CLIENT(args_CLIENT * arg);

void set_Client_Stop(args_CLIENT * argClient);
int is_Client_Stop(args_CLIENT * argClient);

void *thread_UDP_CLIENT(void *args);
```



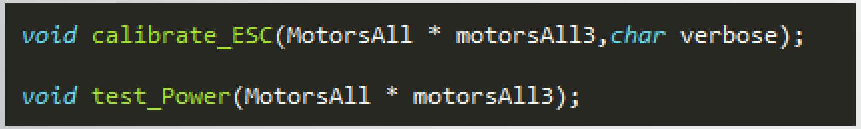
```
void set_Controller_Stop(args_CONTROLLER * argControler);
int is_Controller_Stop(args_CONTROLLER * argControler);

void *thread_CONTROLLER(void *args);
int init_args_CONTROLLER(args_CONTROLLER ** argControler,volatile sig_atomic_t * signalControllerStop);
void clean_args_CONTROLLER(args_CONTROLLER * arg);
```



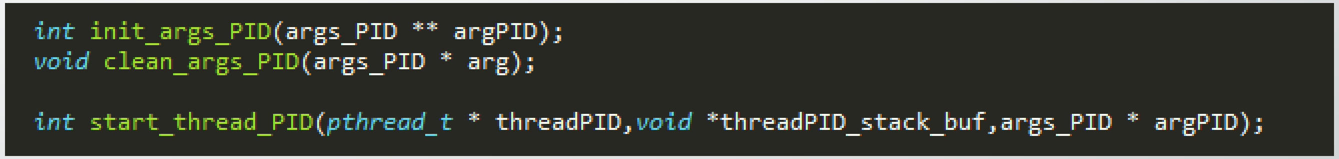
```
int init_MotorsAll(MotorsAll ** motorsAll,volatile sig_atomic_t * signalMotorStop);
void clean_MotorsAll(MotorsAll * arg);

int set_power(MotorsAll * MotorsAll, int * powers);
void set_Motor_Stop(MotorsAll * MotorsAll);
int is_Motor_Stop(MotorsAll * MotorsAll);
```



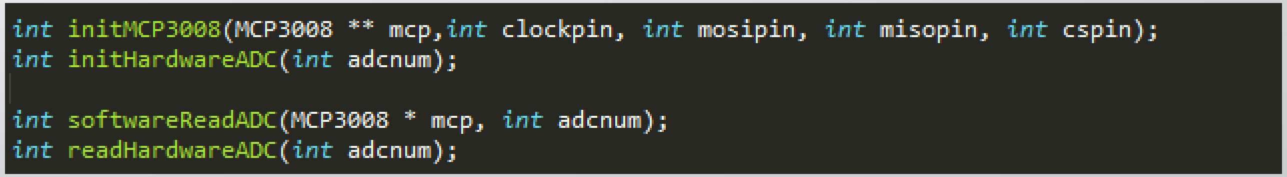
```
void init_inputJoystick(inputJoystick *input);
int update_inputJoystick(inputJoystick *input, int joystickNumber);
void clean_inputJoystick(inputJoystick *input);
int update_eventJoystick(inputJoystick *input);

char * isConnect_Joystick(int number);
```



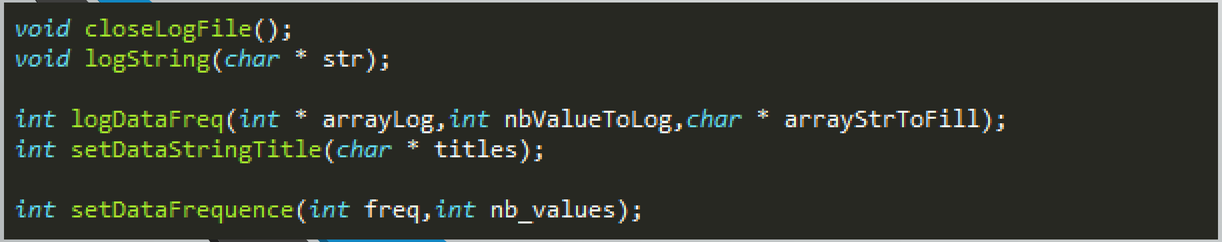
```
void calibrate_ESC(MotorsAll * motorsAll3,char verbose);

void test_Power(MotorsAll * motorsAll3);
```



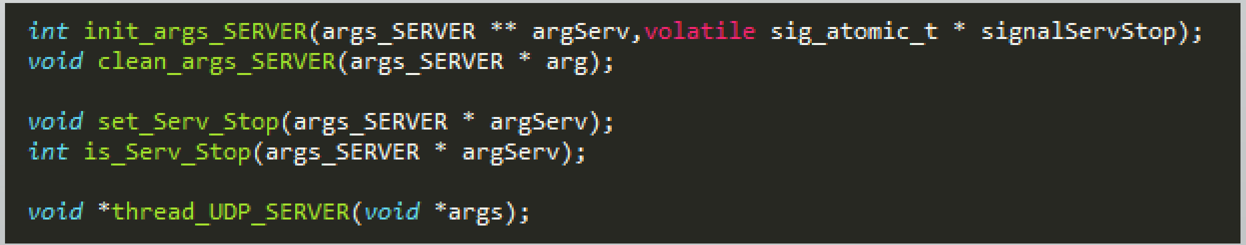
```
int init_args_PID(args_PID ** argPID);
void clean_args_PID(args_PID * arg);

int start_thread_PID(pthread_t * threadPID,void *threadPID_stack_buf,args_PID * argPID);
```



```
int initMCP3008(MCP3008 ** mcp,int clockpin, int mosipin, int misopin, int cspin);
int initHardwareADC(int adcnum);

int softwareReadADC(MCP3008 * mcp, int adcnum);
int readHardwareADC(int adcnum);
```

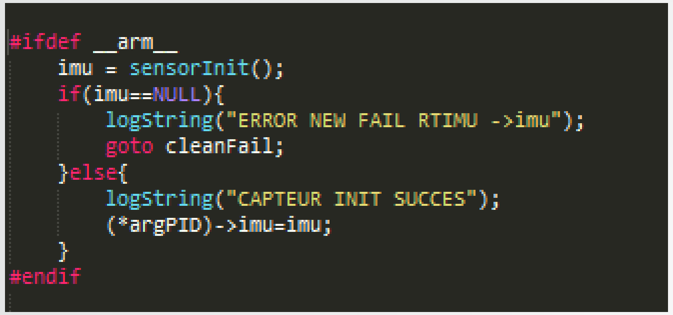


```
void closeLogFile();
void logString(char * str);

int logDataFreq(int * arrayLog,int nbValueToLog,char * arrayStrToFill);
int setDataStringTitle(char * titles);

int setDataFrequence(int freq,int nb_values);
```

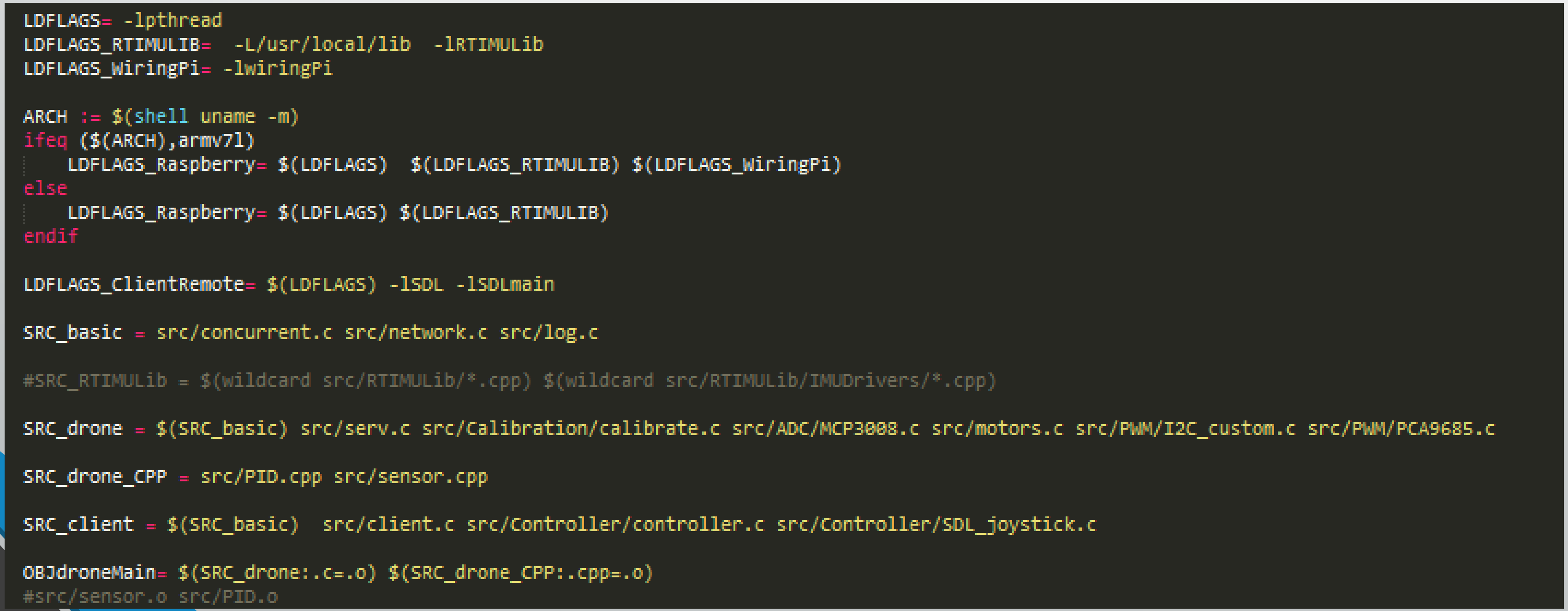


```
int init_args_SERVER(args_SERVER ** argServ,volatile sig_atomic_t * signalServStop);
void clean_args_SERVER(args_SERVER * arg);

void set_Serv_Stop(args_SERVER * argServ);
int is_Serv_Stop(args_SERVER * argServ);

void *thread_UDP_SERVER(void *args);
```

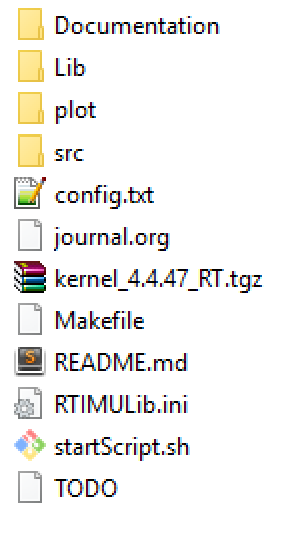
Gestion Projet



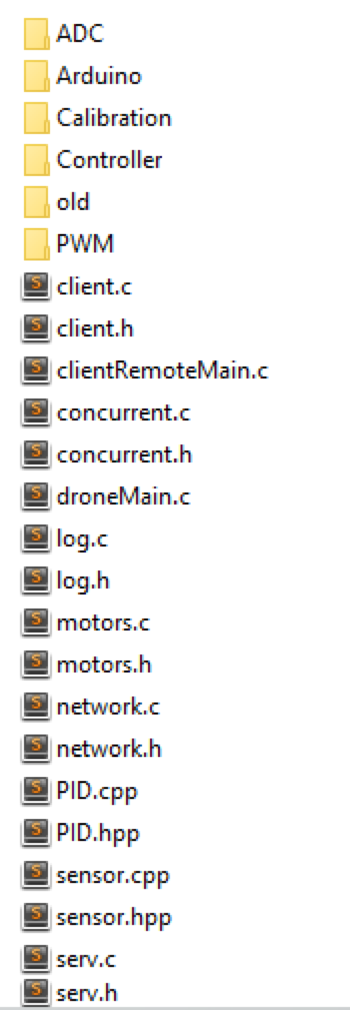
```
#ifdef __arm__
    imu = sensorInit();
    if(imu==NULL){
        logString("ERROR NEW FAIL RTIMU ->imu");
        goto cleanFail;
    }else{
        logString("CAPTEUR INIT SUCCES");
        (*argPID)->imu=imu;
    }
#endif
```



```
LDFLAGS= -lpthread
LDFLAGS_RTIMULIB=  -L/usr/local/lib  -lRTIMULib
LDFLAGS_WiringPi= -lwiringPi

ARCH := $(shell uname -m)
ifeq ($(ARCH),armv7l)
    LDFLAGS_Raspberry= $(LDFLAGS)  $(LDFLAGS_RTIMULIB) $(LDFLAGS_WiringPi)
else
    LDFLAGS_Raspberry= $(LDFLAGS) $(LDFLAGS_RTIMULIB)
endif

LDFLAGS_ClientRemote= $(LDFLAGS) -lSDL -lSDLmain

SRC_basic = src/concurrent.c src/network.c src/log.c

#SRC_RTIMULib = $(wildcard src/RTIMULib/*.cpp) $(wildcard src/RTIMULib/IMUDrivers/*.cpp)

SRC_drone = $(SRC_basic) src/serv.c src/Calibration/calibrate.c src/ADC/MCP3008.c src/motors.c src/PWM/I2C_custom.c src/PWM/PCA9685.c

SRC_drone_CPP = src/PID.cpp src/sensor.cpp

SRC_client = $(SRC_basic)  src/client.c src/Controller/controller.c src/Controller/SDL_joystick.c

OBJdroneMain= $(SRC_drone:.c=.o) $(SRC_drone_CPP:.cpp=.o)
#src/sensor.o src/PID.o
```

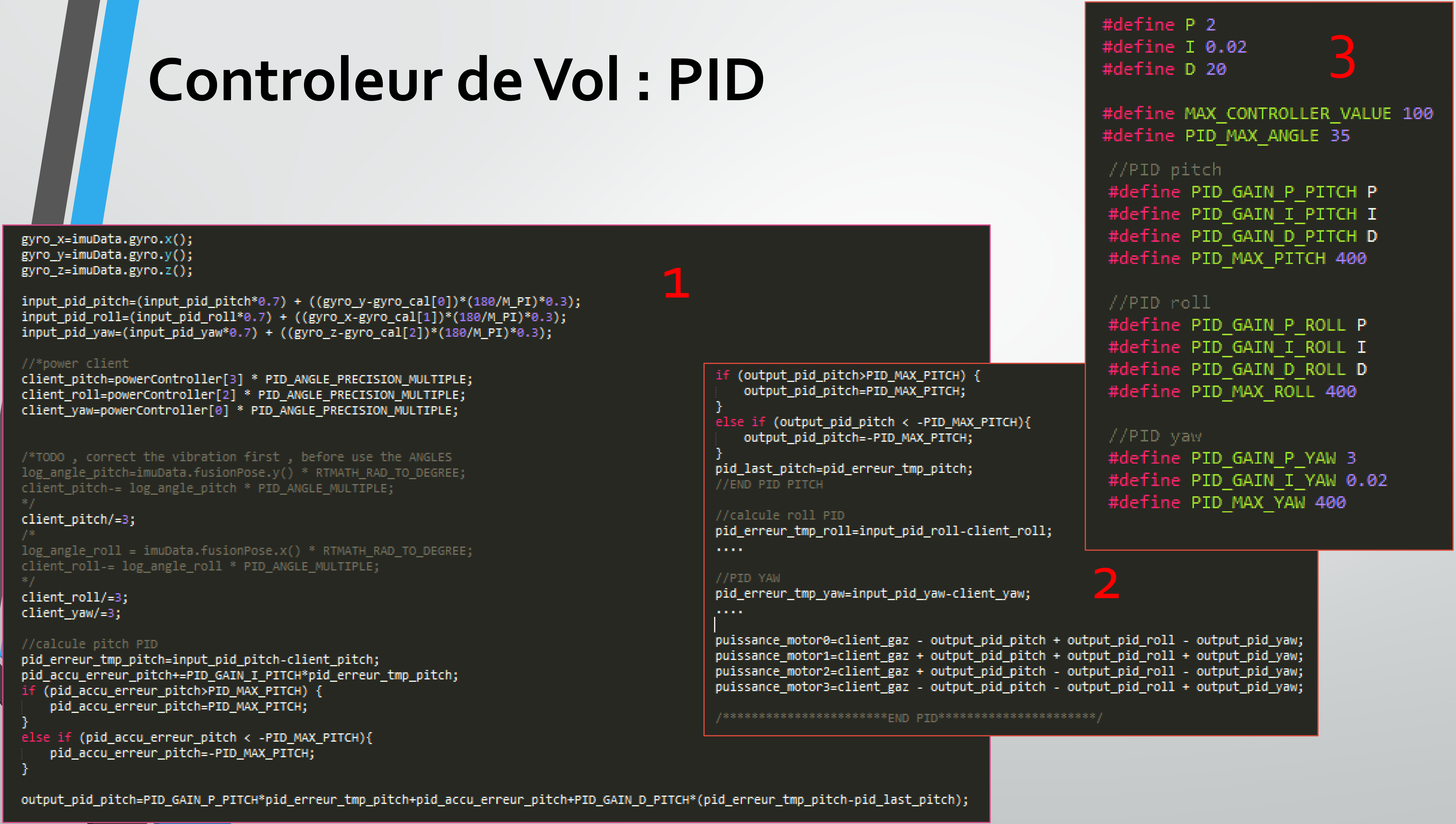


Documentation
Lib
plot
src
config.txt
journal.org
kernel_4.4.47_RT.tgz
Makefile
README.md
RTIMULib.ini
startScript.sh
TODO



ADC
Arduino
Calibration
Controller
old
PWM
client.c
client.h
clientRemoteMain.c
concurrent.c
concurrent.h
droneMain.c
log.c
log.h
motors.c
motors.h
network.c
network.h
PID.cpp
PID.hpp
sensor.cpp
sensor.hpp
serv.c
serv.h

Controleur de Vol : PID



1

```
gyro_x=imuData.gyro.x();
gyro_y=imuData.gyro.y();
gyro_z=imuData.gyro.z();

input_pid_pitch=(input_pid_pitch*0.7) + ((gyro_y-gyro_cal[0])*(180/M_PI)*0.3);
input_pid_roll=(input_pid_roll*0.7) + ((gyro_x-gyro_cal[1])*(180/M_PI)*0.3);
input_pid_yaw=(input_pid_yaw*0.7) + ((gyro_z-gyro_cal[2])*(180/M_PI)*0.3);

//*power client
client_pitch=powerController[3] * PID_ANGLE_PRECISION_MULTIPLE;
client_roll=powerController[2] * PID_ANGLE_PRECISION_MULTIPLE;
client_yaw=powerController[0] * PID_ANGLE_PRECISION_MULTIPLE;


/*TODO , correct the vibration first , before use the ANGLES
log_angle_pitch=imuData.fusionPose.y() * RTMATH_RAD_TO_DEGREE;
client_pitch-= log_angle_pitch * PID_ANGLE_MULTIPLE;
*/
client_pitch/=3;
/*
log_angle_roll = imuData.fusionPose.x() * RTMATH_RAD_TO_DEGREE;
client_roll-= log_angle_roll * PID_ANGLE_MULTIPLE;
*/
client_roll/=3;
client_yaw/=3;

//calcule pitch PID
pid_erreur_tmp_pitch=input_pid_pitch-client_pitch;
pid_accu_erreur_pitch+=PID_GAIN_I_PITCH*pid_erreur_tmp_pitch;
if (pid_accu_erreur_pitch>PID_MAX_PITCH) {
    pid_accu_erreur_pitch=PID_MAX_PITCH;
}
else if (pid_accu_erreur_pitch < -PID_MAX_PITCH){
    pid_accu_erreur_pitch=-PID_MAX_PITCH;
}

output_pid_pitch=PID_GAIN_P_PITCH*pid_erreur_tmp_pitch+pid_accu_erreur_pitch+PID_GAIN_D_PITCH*(pid_erreur_tmp_pitch-pid_last_pitch);
```

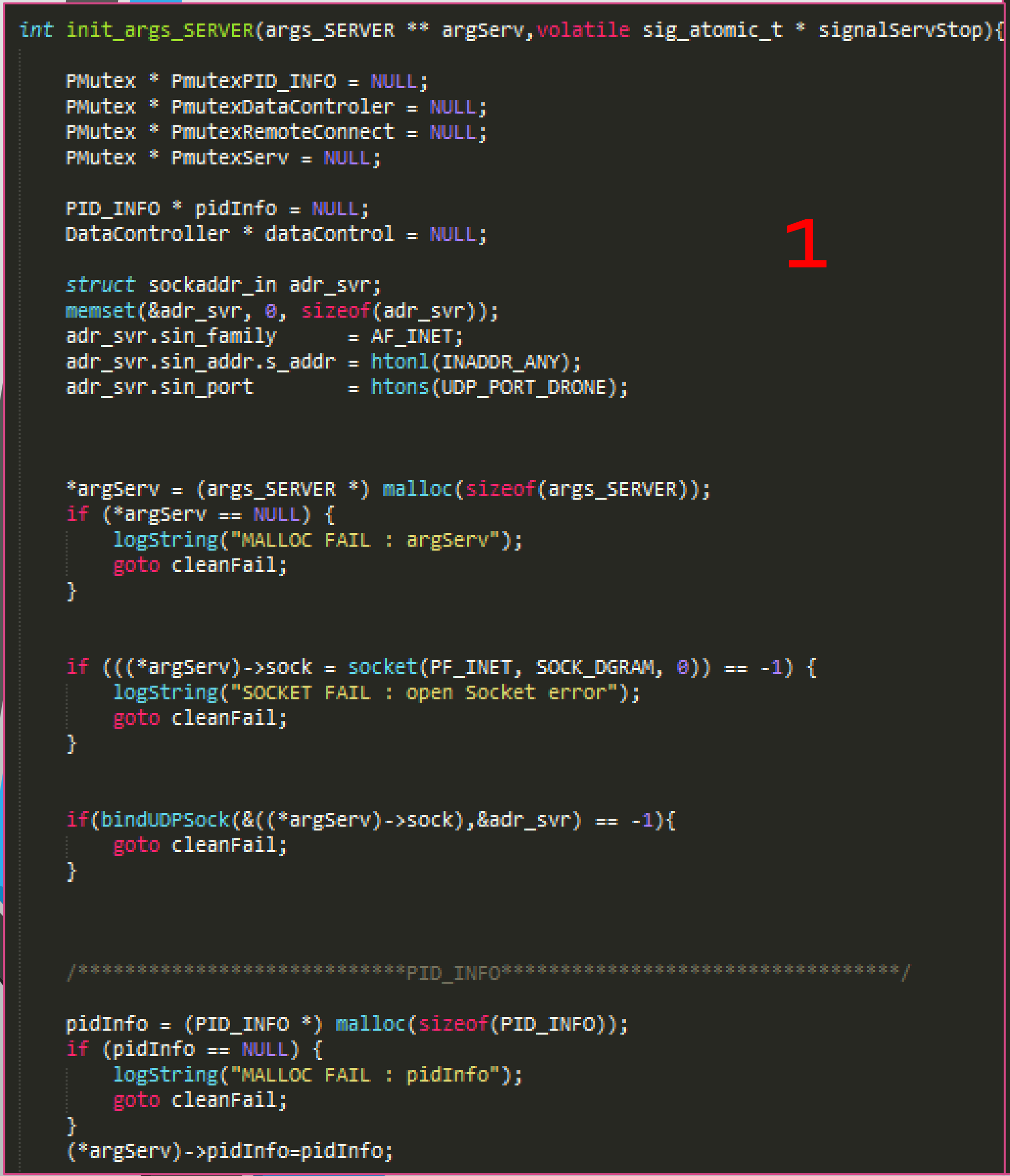
2

```
if (output_pid_pitch>PID_MAX_PITCH) {
    output_pid_pitch=PID_MAX_PITCH;
}
else if (output_pid_pitch < -PID_MAX_PITCH){
    output_pid_pitch=-PID_MAX_PITCH;
}
pid_last_pitch=pid_erreur_tmp_pitch;
//END PID PITCH

//calcule roll PID
pid_erreur_tmp_roll=input_pid_roll-client_roll;
....

//PID YAW
pid_erreur_tmp_yaw=input_pid_yaw-client_yaw;
....

puissance_motor0=client_gaz - output_pid_pitch + output_pid_roll - output_pid_yaw;
puissance_motor1=client_gaz + output_pid_pitch + output_pid_roll + output_pid_yaw;
puissance_motor2=client_gaz + output_pid_pitch - output_pid_roll - output_pid_yaw;
puissance_motor3=client_gaz - output_pid_pitch - output_pid_roll + output_pid_yaw;

/***********************END PID**********************/
```

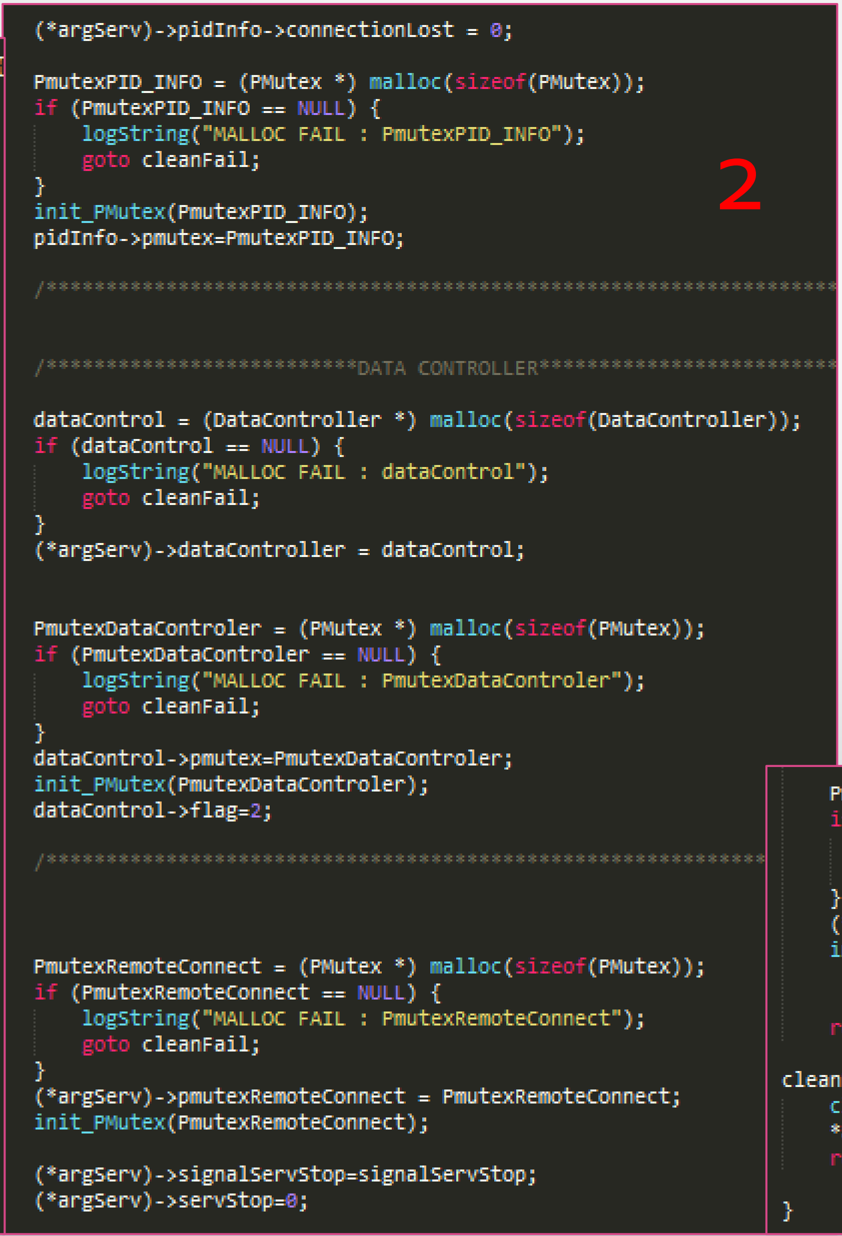
3

```
#define P 2
#define I 0.02
#define D 20

#define MAX_CONTROLLER_VALUE 100
#define PID_MAX_ANGLE 35

 //PID pitch
 #define PID_GAIN_P_PITCH P
 #define PID_GAIN_I_PITCH I
 #define PID_GAIN_D_PITCH D
 #define PID_MAX_PITCH 400

 //PID roll
 #define PID_GAIN_P_ROLL P
 #define PID_GAIN_I_ROLL I
 #define PID_GAIN_D_ROLL D
 #define PID_MAX_ROLL 400

 //PID yaw
 #define PID_GAIN_P_YAW 3
 #define PID_GAIN_I_YAW 0.02
 #define PID_MAX_YAW 400
```

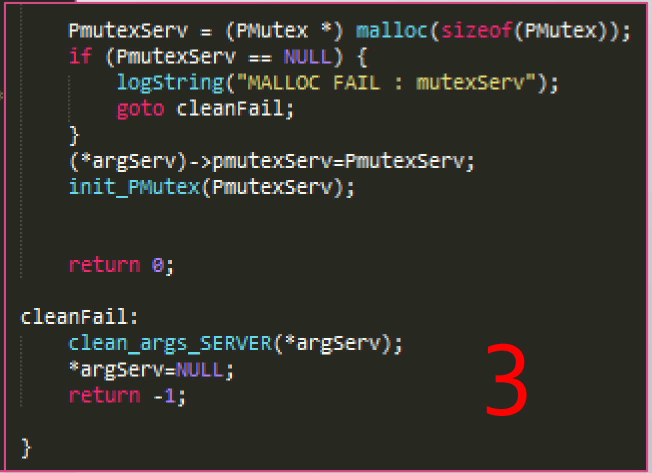
Code Serveur



```c
int init_args_SERVER(args_SERVER ** argServ,volatile sig_atomic_t * signalServStop){

    PMutex * PmutexPID_INFO = NULL;
    PMutex * PmutexDataControler = NULL;
    PMutex * PmutexRemoteConnect = NULL;
    PMutex * PmutexServ = NULL;

    PID_INFO * pidInfo = NULL;
    DataController * dataControl = NULL;

    struct sockaddr_in adr_svr;
    memset(&adr_svr, 0, sizeof(adr_svr));
    adr_svr.sin_family      = AF_INET;
    adr_svr.sin_addr.s_addr = htonl(INADDR_ANY);
    adr_svr.sin_port        = htons(UDP_PORT_DRONE);



    *argServ = (args_SERVER *) malloc(sizeof(args_SERVER));
    if (*argServ == NULL) {
        logString("MALLOC FAIL : argServ");
        goto cleanFail;
    }


    if (((*argServ)->sock = socket(PF_INET, SOCK_DGRAM, 0)) == -1) {
        logString("SOCKET FAIL : open Socket error");
        goto cleanFail;
    }


    if(bindUDPSock(&((*argServ)->sock),&adr_svr) == -1){
        goto cleanFail;
    }



    /****************************PID_INFO**********************************/

    pidInfo = (PID_INFO *) malloc(sizeof(PID_INFO));
    if (pidInfo == NULL) {
        logString("MALLOC FAIL : pidInfo");
        goto cleanFail;
    }
    (*argServ)->pidInfo=pidInfo;
```

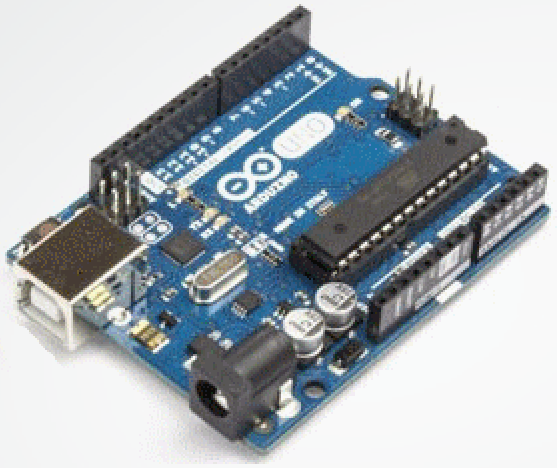


```c
(*argServ)->pidInfo->connectionLost = 0;

PmutexPID_INFO = (PMutex *) malloc(sizeof(PMutex));
if (PmutexPID_INFO == NULL) {
    logString("MALLOC FAIL : PmutexPID_INFO");
    goto cleanFail;
}
init_PMutex(PmutexPID_INFO);
pidInfo->pmutex=PmutexPID_INFO;

/*****************************************************************


/**************************DATA CONTROLLER************************

dataControl = (DataController *) malloc(sizeof(DataController));
if (dataControl == NULL) {
    logString("MALLOC FAIL : dataControl");
    goto cleanFail;
}
(*argServ)->dataController = dataControl;


PmutexDataControler = (PMutex *) malloc(sizeof(PMutex));
if (PmutexDataControler == NULL) {
    logString("MALLOC FAIL : PmutexDataControler");
    goto cleanFail;
}
dataControl->pmutex=PmutexDataControler;
init_PMutex(PmutexDataControler);
dataControl->flag=2;

/***********************************************************



PmutexRemoteConnect = (PMutex *) malloc(sizeof(PMutex));
if (PmutexRemoteConnect == NULL) {
    logString("MALLOC FAIL : PmutexRemoteConnect");
    goto cleanFail;
}
(*argServ)->pmutexRemoteConnect = PmutexRemoteConnect;
init_PMutex(PmutexRemoteConnect);

(*argServ)->signalServStop=signalServStop;
(*argServ)->servStop=0;
```

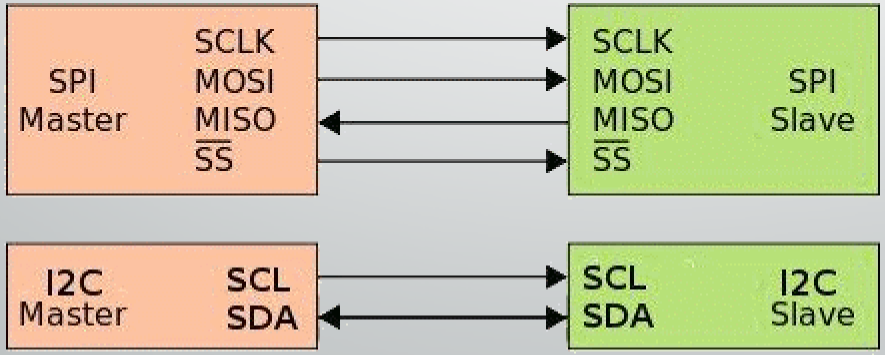


```c
    PmutexServ = (PMutex *) malloc(sizeof(PMutex));
    if (PmutexServ == NULL) {
        logString("MALLOC FAIL : mutexServ");
        goto cleanFail;
    }
    (*argServ)->pmutexServ=PmutexServ;
    init_PMutex(PmutexServ);


    return 0;

cleanFail:
    clean_args_SERVER(*argServ);
    *argServ=NULL;
    return -1;

}
```

Conclusion



Real-Time LINUX



SPI Master

SCLK
MOSI
MISO
SS

SCLK
MOSI
MISO
SS

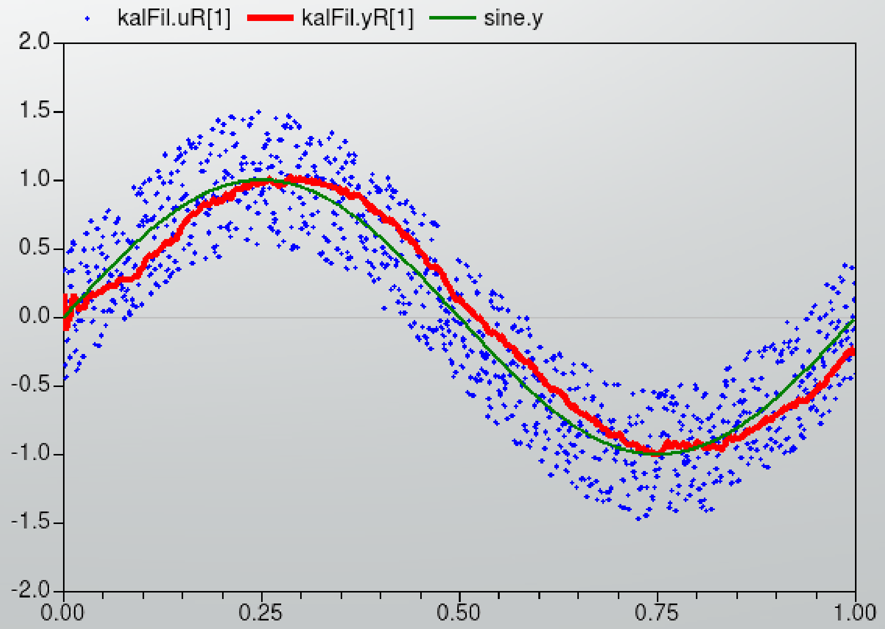
SPI Slave

I2C Master

SCL
SDA

SCL
SDA

I2C Slave



Reprise de Projet