```c
/*****************************************************

    PA10 Control Program ( 1-Axis Only )

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


#include "include.h"
#include "main_cfg.h"
#include "cfg_arm.h"
#include "params_arm.h"
#include "./Arcnet/arc_pci.h"
#include "timer.h"

struct params motorp;
int brakeoff_joint;
int ctrlEndFlag = 0;
int ctrltrig = OFF;

const double inertia = INERTIA1;
const double Kd = KD1;
const double Kp = KP1;
const double joint_limit[2] = {175.*DEG2RAD, 85.*DEG2RAD};
const double max_torque    =  MAX_TORQUE1 ;
struct path    path_j;
struct status  cur_j, des_j;
double torque;

void init(void);
int initializeAll(void);
void initializeData(void);
void start(void);
int ctrlTask(struct params *motor);
void fin(void);
void jointCtrl(struct params *param, int trig);
void getCurrentPosition(struct status *Cur_j);
void pathInit_j(double *Start, double *Destination, double Time);
void pathGenerate_j(struct status *Des_j, unsigned long Time);
void pdCtrl(struct status *Des_j, struct status *Cur_j);
void allzero(void);
void allboff(void);
void allbon(void);
void boff1(void);
void boff2(void);
void boff3(void);
void boff4(void);
void boff5(void);
void boff6(void);
void boff7(void);
void control(struct params *motor);
void allbrakeoff(void);
void brakeoff(int joint);
void Nop(void);
int endTask(void);
void joint_moveto(double Angle,double Time);
void All_OFFBrake(void);
void OFFBrake(void);
void ONBrake(void);

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
extern thread_pool_t * cp (int argc, char **argv);

/*----------------------------------------------------------------------*/

void
init(void)
```

```c
{
    ThreadCtl(_NTO_TCTL_IO,0);

    if ((chid = ChannelCreate (0)) == -1){
        fprintf (stderr,"timer.c: couldn't create channel!\n");
        perror (NULL);
        exit (EXIT_FAILURE);
    }

    setupTimer();

    initializeAll();

}


int
initializeAll(void)
{
    char key[16];


    printf("\n\t***** Choose the Control Mode *****\n\n");
    printf("\tOperation or Simulation ? [o/s] \n\n");
    fgets(key, sizeof(key), stdin);

    if (*key == 'o') {
        CtrlMode = operation;
        arcInit();
        printf("\n\tControl Mode -> Operation \n\n");
    }
    else if (*key == 's') {
        CtrlMode = simulation;
        printf("\n\tControl Mode -> Simulation \n\n");
    }

    else {
        CtrlMode = simulation;
        printf("\n\tControl Mode -> Simulation \n\n");
    }


    initializeData();

    return OK;
}

void
initializeData(void)
{

    motorp.mode = nop;
    motorp.desPos = 0.;
    motorp.desTime = 0.;
    memset(&path_j, 0, sizeof(path_j));

}

void
start(void)
{
    arcnet_start();



    timer_settime(timerid, 0, &timer, NULL);
    ctrltrig = 1;
    ctrlEndFlag = 1;

    ctrlTask(&motorp);

}

int
ctrlTask(struct params *motor)
{
    int rcvid=1;
    MessageT msg;
    static unsigned long ticks = 0;
    pthread_attr_t attr;

    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_DETACHED );

    while(ctrltrig){
        rcvid = MsgReceive(chid, &msg, sizeof(msg), NULL);
        if(rcvid == 0){


            if(!ctrlEndFlag){
                printf("ERROR : Control Task is out of time.\n        fin\n");
                fin();
                return (ERROR);
            }
            ctrlEndFlag = 0;
            pthread_create(NULL,&attr,(void *)control,motor);
            ticks++;

        }
    }

    return (EXIT_SUCCESS);

}

void
control(struct params *motor)
{

    int stat1;
    static struct params param;
    static int mode = nop;
    static int joint;
    static int trig;


    if( pthread_mutex_lock ( &mutex ) != EOK ){
        printf(" ERROR : mutex cannnot lock.\n        fin\n");
        fin();
        return;
    }

    switch(CtrlMode){
    case operation:
        stat1 = iSend_C();
        break;
    case simulation:
        break;

    default:
        break;
    }

    memcpy(&param, motor, sizeof(struct params));

    getCurrentPosition(&cur_j);

    if(param.trig){
        trig = TRUE;
        motor->trig = FALSE;
    }
    else
        trig = FALSE;

    mode = param.mode;
    joint = param.joint;

    switch(mode){

    case joint_mode:
        jointCtrl(&param, trig);
        break;

    case allbrakeoff_mode:
        allbrakeoff();
        break;
    case brakeoff_mode:
        brakeoff(joint);
        break;
    default:
        Nop();

        break;
    }

    switch(CtrlMode){
    case operation:
        stat1 = RecData();
        break;
    case simulation:
        break;

    default:
        break;
    }

    endTask();
    return;

}

void
jointCtrl(struct params *param, int trig)
{

    static unsigned long tick;
    static double desPos,desTime;


    if(trig){

        desPos = param->desPos;
    }

    desTime = param->desTime;
    tick = 0;

    pathInit_j(des_j.pos, desPos, desTime);

    return;
}

pathGenerate_j(&des_j, tick);

tick++;
pdCtrl(&des_j, &cur_j);


}

void
getCurrentPosition(struct status *Cur_j)
{

double angle = 0.;
static double pre_j_pos = 0.;
static double pre_j_vel = 0.;

/*--- get joint angles ---*/

GetPosition(angle);

Cur_j->pos = angle;

Cur_j->vel = (Cur_j->pos - pre_j_pos) / TICKS;
Cur_j->acc = (Cur_j->vel - pre_j_vel) / TICKS;

pre_j_pos = Cur_j->pos;
pre_j_vel = Cur_j->vel;
}

/*--- Selection of control mode ---*/

switch(CtrlMode){
case operation:
break;
case simulation:


break;

default:
break;
}
}

void
pathInit_j(double *Start, double *Destination, double Time)
{
int jnt;
double t3, t4, t5;
double diff;

t3 = Time * Time * Time;
t4 = t3 * Time;
t5 = t4 * Time;


diff = Destination - Start;

path_j.pos[0] = Start;
path_j.pos[3] =  10. * diff / t3;
path_j.pos[4] = -15. * diff / t4;
path_j.pos[5] =   6. * diff / t5;

path_j.vel[2] = 3. * path_j.pos[3];
path_j.vel[3] = 4. * path_j.pos[4];
path_j.vel[4] = 5. * path_j.pos[5];

path_j.acc[1] =  6. * path_j.pos[3];
path_j.acc[2] = 12. * path_j.pos[4];
path_j.acc[3] = 20. * path_j.pos[5];
}
path_j.time = Time;
}

void
pathGenerate_j(struct status *Des_j, unsigned long Time)
{

double t;


t = (Time * TICKS);
t = (t > path_j.time) ? path_j.time : t;


Des_j->pos = path_j.pos[0]
+ t * t * t * (path_j.pos[3]
+ t * (path_j.pos[4] + t * path_j.pos[5]));

Des_j->vel = t * t * (path_j.vel[2]
+ t * (path_j.vel[3] + t * path_j.vel[4]));

Des_j->acc = t * (path_j.acc[1] + t * (path_j.acc[2]
+ t * path_j.acc[3]));
}
}

void
pdCtrl(struct status *Des_j, struct status *Cur_j)
{

double accel;

accel = Des_j->acc
+ Kd * (Des_j->vel - Cur_j->vel)
    + Kp * (Des_j->pos - Cur_j->pos);

    torque = inertia * accel;

    if(torque > max_torque)
    torque = max_torque;
    else if(torque < - max_torque)
    torque = - max_torque;
    }

switch(CtrlMode){
case operation:
SetTorq(torque);
break;
case simulation:
NoTorq();
break;

default:
NoTorq();
break;
}
}
```

```
int
endTask( void )
{
ctrlEndFlag = 1;
pthread_mutex_unlock ( &mutex );

return (EXIT_SUCCESS);

}

void
allbrakeoff(void)
{
    AllBrakeOFF();
}

void
brakeoff(int joint)
{
    BrakeOFF(joint);

}

void
Nop(void)
{
    NoTorq();
}

void
joint_moveto(double Angle, double Time)
{
    motorp.desPos = Angle;
    motorp.desTime = Time;
    motorp.trig    = TRUE;
    motorp.mode    = joint_mode;
}

void
All_OFFBrake(void)
{

    motorp.mode = allbrakeoff_mode;
}

OFFBrake(void)
{

    motorp.mode = brakeoff_mode;
    motorp.joint = brakeoff_joint;
}

void
ONBrake(void)
{
    motorp.mode = nop;
}

void
fin(void)
{
    ctrltrig = 0;
    ONBrake();
    arcFin();
    timer_delete(timerid);
```

```c
}

int
main(void)
{

    cp(NULL, NULL);
    return(OK);

}
```