robot_IO_ctrl_phleb38.cpp

```cpp

#include <Engine.h>
#include <JacoUdp.h>

#include <Windows.h>
#include <conio.h>
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <ctime>
#include "windows.h"
#include <chrono>
#include <math.h>

#include <time.h>
#include <sys\timeb.h>
#include <winbase.h>

#include "WindowsExample_AngularControl\Lib_Examples\CommunicationLayerWindows.h"
#include "WindowsExample_AngularControl\Lib_Examples\CommandLayer.h"
#include "WindowsExample_AngularControl\Lib_Examples\KinovaTypes.h"
#include <SDL.h>
#undef main
#pragma comment(lib,"libmat.lib")
#pragma comment(lib,"libeng.lib")
#pragma comment(lib,"libmex.lib")
#pragma comment(lib,"libmx.lib")
using namespace std;
using namespace System;
using namespace System::Threading;
using namespace System::Collections::Generic;

//A handle to the API.
HINSTANCE commandLayer_handle;

//Function pointers to the functions we need
int(*MyInitAPI)();
int(*MyCloseAPI)();
int(*MySendBasicTrajectory)(TrajectoryPoint command);
int(*MyGetDevices)(KinovaDevice devices[MAX_KINOVA_DEVICE], int &result);
int(*MySetActiveDevice)(KinovaDevice device);
int(*MyMoveHome)();
int(*MyInitFingers)();
int(*MyGetAngularCommand)(AngularPosition &);
int(*MyEraseAllTrajectories)();
int(*MyGetAngularPosition)(AngularPosition &);
int quitter = 1;
Engine *m_pEngine = engOpen("null");

double mc_get_accurate_time()
{
    return((double)GetTickCount() / 1000.0);
}

double getTime()
{
    struct timeb lTp;

    ftime(&lTp);

    double lTime = (double)lTp.time + 0.001f*lTp.millitm;

    return lTime;
}

void afficherGraphique()
{
    mxArray* dstate = mxCreateDoubleMatrix(1, 1, mxREAL);
    double* pstate = mxGetPr(dstate);
    *pstate = 0;
    engPutVariable(m_pEngine, "state", dstate);
    engEvalString(m_pEngine, "graphique");
    *pstate = 1;
    engPutVariable(m_pEngine, "state", dstate);
    while (!quitter)
    {
        engEvalString(m_pEngine, "graphique");
    }
}

int main(array<System::String ^> ^args)
{
    // Ouverture de l'engine matlab
    Engine *m_pEngine;
    m_pEngine = engOpen("null");


    //Chargement de l'API de kinova
    commandLayer_handle = LoadLibrary(L"CommandLayerWindows.dll");

    int programResult = 0;
    Sint16 x_move = 0, y_move = 0, z_move = 0, z_move_1 = 0, z_move_2 = 0;
    JacoUdp jacoInst;
    int i = 0;
    int j = 0;
    int k = 0;
    int flag = 0;
    double time1, time2, ang_1, ang_2, ang_3;
    int iteration = 10000;
    void *ans = NULL;
```

```
103
104     int dummy = 0;
105     AngularPosition dataPosition;
106
107     // Definition des variables communiquant avec matlab (memes noms que dans matlab)
108     mxArray* dmove = mxCreateDoubleMatrix(1, 3, mxREAL);
109     mxArray* dtheta = mxCreateDoubleMatrix(1, 3, mxREAL);
110     mxArray* dtheta_next = mxCreateDoubleMatrix(1, 3, mxREAL);
111     mxArray* dtheta_delt = mxCreateDoubleMatrix(1, 3, mxREAL);
112     mxArray* dstate = mxCreateDoubleMatrix(1, 1, mxREAL);
113
114
115     // initialisation des valeurs des variables
116     double *pmove = new double[3];
117     pmove[0] = 0;
118     pmove[1] = 0;
119     pmove[2] = 0;
120     double *ptheta = new double[3];
121     ptheta[0] = 0;
122     ptheta[1] = 0;
123     ptheta[2] = 0;
124     double *ptheta_next = new double[3];
125     ptheta_next[0] = 0;
126     ptheta_next[1] = 0;
127     ptheta_next[2] = 0;
128
129     double *ptheta_delt = new double[3];
130     ptheta_delt[0] = 0;
131     ptheta_delt[1] = 0;
132     ptheta_delt[2] = 0;
133     double *pstate = mxGetPr(dstate);
134
135     memcpy(mxGetPr(dmove), pmove, 3 * sizeof(double));
136     memcpy(mxGetPr(dtheta), ptheta, 3 * sizeof(double));
137     memcpy(mxGetPr(dtheta_next), ptheta_next, 3 * sizeof(double));
138     memcpy(mxGetPr(dtheta_delt), ptheta_delt, 3 * sizeof(double));
139
140     *pstate = (double)0; // propre au script matlab
141
142     // envoi des valeur initiales dans le script maple
143
144     engPutVariable(m_pEngine, "move", dmove);
145     engPutVariable(m_pEngine, "theta", dtheta);
146
147     //execution de commandes matlab. Il faut ajouter le path dans lequel le script matlab se situe.
148     engEvalString(m_pEngine, "addpath('U:\\matlab\\rrr version phil')");
149     engEvalString(m_pEngine, "addpath('U:\\matlab\\rrr version phil\\geom3d\\geom3d')");
150     engEvalString(m_pEngine, "addpath('U:\\matlab\\rrr version phil\\geom3d\\meshes3d')");
151
152     //execution du script matlab.
153     engEvalString(m_pEngine, "slide_multi_3ddl_impl");
154
155     //changement d'etat dans le script
156     *pstate = (double)1;
157     engPutVariable(m_pEngine, "state", dstate);
158     engEvalString(m_pEngine, "slide_multi_3ddl_impl");
159
160     *pstate = (double)2;
161     engPutVariable(m_pEngine, "state", dstate);
162
163     // assignation des valeurs des position articulaires du bras jaco aux variable communiquant avec matlab
164     ptheta[0] = abs(((double)((dataPosition.Actuators.Actuator1 - 90.0) - 360.0) - 360.0)*3.14159265 / 180.0);
165     ptheta[1] = (double)((dataPosition.Actuators.Actuator2 - 0)*3.14159265 / 180.0);
166     ptheta[2] = abs(((double)((dataPosition.Actuators.Actuator3 - 210.0) - 360.0) - 360.0)*3.14159265 / 180.0);
167     ptheta_next[0] = abs(((double)((dataPosition.Actuators.Actuator1 - 90.0) - 360.0) - 360.0)*3.14159265 / 180.0);
168     ptheta_next[1] = (double)((dataPosition.Actuators.Actuator2 - 0)*3.14159265 / 180.0);
169     ptheta_next[2] = abs(((double)((dataPosition.Actuators.Actuator3 - 210.0) - 360.0) - 360.0)*3.14159265 / 180.0);
170
171     // envoi de ces valeurs
172     engPutVariable(m_pEngine, "theta", dtheta);
173
174     // Initialisation des pointeurs de fonctions de l'API de kinova. Ceci permet d'utiliser les fonctions dans ce script
175     MyInitAPI = (int(*)()) GetProcAddress(commandLayer_handle, "InitAPI");
176     MyCloseAPI = (int(*)()) GetProcAddress(commandLayer_handle, "CloseAPI");
177     MyGetDevices = (int(*)(KinovaDevice[MAX_KINOVA_DEVICE], int&)) GetProcAddress(commandLayer_handle, "GetDevices");
178     MySetActiveDevice = (int(*)(KinovaDevice)) GetProcAddress(commandLayer_handle, "SetActiveDevice");
179     MySendBasicTrajectory = (int(*)(TrajectoryPoint)) GetProcAddress(commandLayer_handle, "SendBasicTrajectory");
180     MyGetAngularCommand = (int(*)(AngularPosition &)) GetProcAddress(commandLayer_handle, "GetAngularCommand");
181     MyMoveHome = (int(*)()) GetProcAddress(commandLayer_handle, "MoveHome");
182     MyInitFingers = (int(*)()) GetProcAddress(commandLayer_handle, "InitFingers");
183     MyEraseAllTrajectories = (int(*)()) GetProcAddress(commandLayer_handle, "EraseAllTrajectories");
184     MyGetAngularPosition = (int(*)(AngularPosition &)) GetProcAddress(commandLayer_handle, "GetAngularPosition");
185
186     //Verify that all functions has been loaded correctly
187     if ((MyInitAPI == NULL) || (MyCloseAPI == NULL) || (MySendBasicTrajectory == NULL) ||
188         (MyGetDevices == NULL) || (MySetActiveDevice == NULL) || (MyGetAngularCommand == NULL) ||
189         (MyMoveHome == NULL) || (MyInitFingers == NULL))
190
191     {
192         cout << "* * *  E R R O R   D U R I N G   I N I T I A L I Z A T I O N  * * *" << endl;
193         programResult = 0;
194     }
195     else
196     {
197         cout << "I N I T I A L I Z A T I O N   C O M P L E T E D" << endl << endl;
198
199         int result = (*MyInitAPI)();
200
201         AngularPosition currentCommand;
202
203         cout << "Initialization's result :" << result << endl;
204
205         KinovaDevice list[MAX_KINOVA_DEVICE];
206         int devicesCount = MyGetDevices(list, result);
207
208         cout << "Found a robot on the USB bus (" << list[0].SerialNumber << ")" << endl;
```

```
209         MySetActiveDevice(list[0]);
210
211
212         TrajectoryPoint pointToSend;
213         int t = 0;
214         pointToSend.InitStruct();
215         MyMoveHome();
216         pointToSend.Position.Type = ANGULAR_VELOCITY;
217
218         t = 0;
219
220         MySendBasicTrajectory(pointToSend);
221
222         // definition des vitesses articulaires pour positionner le robot dans se position de départ
223
224         pointToSend.Position.Actuators.Actuator1 = 40;
225         pointToSend.Position.Actuators.Actuator2 = 0;
226         pointToSend.Position.Actuators.Actuator3 = 40;
227
228         // envoi des commandes au bras jaco
229         while (t < 600)
230         {
231             MySendBasicTrajectory(pointToSend);
232             t++;
233             Sleep(4);
234         }
235
236
237
238         SDL_Init(SDL_INIT_JOYSTICK);// initialise juste le joystick
239         SDL_Joystick *joystick; // on instancie
240         joystick = SDL_JoystickOpen(0); // on l'assigne au numero 0
241
242         time1 = getTime();
243         printf("time1 %f  \n", (time1));
244
245         if (quitter)
246         {
247             printf("Demarer le programme: boutton vert\n");
248             printf("arreter le programme : bouton rouge(apres avoir appuyer le bouton vert)\n");
249         }
250         while (quitter)
251         {
252             // lecture des boutons du joystick
253             SDL_JoystickUpdate();
254             if (SDL_JoystickGetButton(joystick, 0))
255             {
256                 printf("go");
257                 quitter = 0;
258                 //jacoInst.doReceiveData();
259             }
260         }
261         typedef std::chrono::high_resolution_clock Time;
262         auto start = Time::now();
263         auto start3 = Time::now();
264         auto temps_now = Time::now();
265         typedef std::chrono::microseconds micro_sec;
266         typedef std::chrono::duration<float> fsec;
267         fsec delta_t;
268         fsec delta_t3;
269         micro_sec dur = chrono::duration_cast<micro_sec>(delta_t);
270         micro_sec dur3 = chrono::duration_cast<micro_sec>(delta_t3);
271
272         pointToSend.Position.Type = ANGULAR_VELOCITY;
273
274
275         while (!quitter)
276             //for (int l = 0; l < 2000; l++ )
277         {
278             //lecture des positions articulaires du bras jaco
279             (*MyGetAngularPosition)(dataPosition);
280             temps_now = chrono::high_resolution_clock::now();
281             delta_t = temps_now - start;
282             dur = chrono::duration_cast<micro_sec>(delta_t);
283             // lecture des boutons du joystick
284             x_move = SDL_JoystickGetAxis(joystick, 0);
285             y_move = SDL_JoystickGetAxis(joystick, 1);
286             z_move_1 = SDL_JoystickGetAxis(joystick, 4);
287             z_move_2 = SDL_JoystickGetAxis(joystick, 5);
288             z_move = z_move_1;
289             if (abs((int)x_move) < 400) x_move = 0;
290             if (abs((int)y_move) < 400) y_move = 0;
291             if (abs((int)z_move) < 400) z_move = 0;
292
293
294             pmove[0] = (double)x_move / 32768;
295             pmove[1] = (double)y_move / 32768;
296             pmove[2] = (double)z_move / 32768;
297
298             // assignation des valeurs des position articulaires du bras jaco aux variable communiquant avec matlab
299             ptheta[0] = (double)((dataPosition.Actuators.Actuator1 + 270.0)*3.14159265 / 180.0);
300             ptheta[1] = (double)((dataPosition.Actuators.Actuator2 - 180.0)*3.14159265 / 180.0);
301             ptheta[2] = abs(((double)((dataPosition.Actuators.Actuator3 - 200.0) - 360.0) - 360.0)*3.14159265 / 180.0);
302             //ptheta[2] = (double)((dataPosition.Actuators.Actuator3 - 210.0)*3.14159265 / 180.0);
303             memcpy(mxGetPr(dmove), pmove, 3 * sizeof(double));
304             memcpy(mxGetPr(dtheta), ptheta, 3 * sizeof(double));
305
306             //envoi des variables vers matlab
307             engPutVariable(m_pEngine, "move", dmove);
308             engPutVariable(m_pEngine, "theta", dtheta);
309
310             // re-execution du script
311             engEvalString(m_pEngine, "slide_multi_3ddl_impl");
312
313             // lecture des valeurs mises a jour par le script
314             dtheta_delt = engGetVariable(m_pEngine, "delta_theta");
```

```
315             ans = mxGetData(dtheta_delt);
316             ptheta_delt[0] = ((double*)ans)[0];
317             ptheta_delt[1] = ((double*)ans)[1];
318             ptheta_delt[2] = ((double*)ans)[2];
319 
320 
321             // envoi des commandes au bras jaco
322             pointToSend.Position.Actuators.Actuator1 = ptheta_delt[0] / 0.01;
323             pointToSend.Position.Actuators.Actuator2 = ptheta_delt[1] / 0.01;
324             pointToSend.Position.Actuators.Actuator3 = -ptheta_delt[2] / 0.01;
325             MySendBasicTrajectory(pointToSend);
326 
327 
328             // lecture des boutons du joystick
329             SDL_JoystickUpdate();//n?cessaire pour refaire la lecture des input du joystick (si non c'est tout le temps la meme valeur qui est lue
330 
331             if (SDL_JoystickGetButton(joystick, 1))//on pousse le bouton rouge et le robot va vers home et on sort de la loop
332             {
333                 cout << endl << "WARNING: Your robot is now set to angular control. If you use the joystick, it will be a joint by joint movement." << endl;
334                 cout << endl << "C L O S I N G   A P I" << endl;
335                 quitter = 1;
336                 engEvalString(m_pEngine, "close all");
337                 engClose(m_pEngine);
338                 Sleep(1000);
339             }
340             if (SDL_JoystickGetButton(joystick, 2))//on pousse le bouton rouge et le robot va vers home et on sort de la loop
341             {
342                 engEvalString(m_pEngine, "plot3(pose_act(1,1), pose_act(1,2), pose_act(1,3), 'X', 'Color', 'b');");
343                 engEvalString(m_pEngine, "plot3(pose_act(3,1), pose_act(3,2), pose_act(3,3), 'X', 'Color', 'r');");
344             }
345 
346             start3 = Time::now();
347             delta_t3 = temps_now - start3;
348             dur3 = chrono::duration_cast<micro_sec>(delta_t3);
349 
350             while (abs(dur3.count()) < 9900)
351             {
352                 start3 = Time::now();
353                 delta_t3 = temps_now - start3;
354                 dur3 = chrono::duration_cast<micro_sec>(delta_t3);
355             }
356             //cout << dur3.count() << endl;
357         }
358         MyMoveHome();
359         result = (*MyCloseAPI)();
360         time2 = getTime();
361         Console::WriteLine("Example done...");
362         Console::WriteLine("Temps total:  " + (time2 - time1));
363 
364         Thread::Sleep(2000);
365 
366 
367     }
368     return 0;
369 
370 }
371 
```