```cpp
#define WIN32_LEAN_AND_MEAN
#define UNICODE
#ifdef _MSC_VER
#define _CRT_SECURE_NO_WARNINGS
#endif

#include "KinovaTypes.h"
#include <Windows.h>
#include <objbase.h>
#include <WS2tcpip.h> //function lib for win socket networks
#include "CommunicationLayerWindows.h"
#include "CommandLayer.h"
#include <conio.h>
#include <cstring>
#include <iostream>

#include "NuitrackGLSample.h"
#include <GL/glut.h>
#include <iomanip>

#include <royale.hpp>
#include <mutex>
#include <opencv2/opencv.hpp>
#include <sample_utils/PlatformResources.hpp>

// Dynamixel
#include "dynamixel2.h"
#include "FunctionDefine.h"


#include <string.h>
#include <stdio.h>

#include<boost/thread.hpp>
#include<boost/atomic.hpp>
#pragma comment(lib, "dynamixel2_win32.lib")

#ifdef _DEBUG
#pragma comment (lib, "ws2_32.lib")
#endif

#define MAX_IN_CHAR 128

// OpenCV 4.1.0
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/calib3d.hpp>
#include <opencv2/imgproc.hpp>
using namespace std;

HINSTANCE  commandLayer_handle;
NuitrackGLSample sample;

//Function pointers to the functions we need
int(*MyInitAPI)();
int(*MyCloseAPI)();
```

```
57  int(*MySendBasicTrajectory)(TrajectoryPoint command);
58  int(*MyGetDevices)(KinovaDevice devices[MAX_KINOVA_DEVICE], int &result);
59  int(*MySetActiveDevice)(KinovaDevice device);
60  int(*MyMoveHome)();
61  int(*MyInitFingers)();
62  int(*MyGetCartesianCommand)(CartesianPosition &);
63  // State machine
64  enum G_Mode {
65      Direct,
66      HandTracking,
67      FreeMoving
68  };
69
70  //global variables
71  int programResult = 0;
72  float* RHandPos;
73  float* LHandPos;
74  float* HeadPos;
75  int NumofBodies = 0;
76  int FirstDetect = 0;
77  int Right_Hand_Grip = -1;
78  int Left_Hand_Grip = -1;
79  int Bodyflag = 0;
80  double rob_pos[3];
81  double Dtogoal = 1000;
82  double GUrep_bnd[] = { 0,0,0 };
83  double GUrep_obs[] = { 0,0,0 };
84  double GUrep[] = { 0,0,0 };
85  double GUatt[3];
86  double gradient[3];
87  double D;
88  double DtoCenter;
89  double Cons;
90  double norm_gradient;
91  int numloop = 0;
92  double bnd[2][3] = { { -0.3,-0.7,-0.1 },{ 0.4,-0.3,0.7 } }; //boundary
93  double bnd_center[] = { 0.5*(bnd[1][1] + bnd[2][1]),0.5*(bnd[1][2] + bnd[2]
      [2]),0.5*(bnd[1][3] + bnd[2][3]) };
94  int T_gap = 1200;
95  int c_gap = -5000;
96  double norm_momentum = 0.0;
97  int imagetype = -1;
98  int obj_id = 1000;
99  float Down_Z = 1000;
100 cv::Vec3f d_XYZt_Kinova = { 0,0,0 };
101 atomic<bool> flag = true;
102
103 //test case 3 - w/ momentum, 2 balls
104 double Kappa = 0.4;
105 double Nu = 1.0e-6;
106 double rate = 0.9;
107 double ObsTh = 0.05;
108 double start[] = { 0.034,-0.2,0.26 };
109 double temp[] = { 0,0,0 };
110 double start_theta[] = { -3.14,0.0,0.0 };
111 double goal[] = { 0.0,0.0,0.0 };
```

```
112 double momentum[] = { 0,0,0 };
113 double obs[2][4] = { { 0.373,-0.44,0.190,0.12 },
      { 0.163,-0.587,0.100,0.12 } }; // x,y,z position + R radius
114 double stepsize = 0.01;
115 int obsnum = 2;
116 double goal_theta[] = { 3.14,0.0,0.0 };
117
118 #define PI 3.141592
119
120 // UDP Threads
121 void UDPthread_1(atomic<bool>& flag) {
122     // 1.3) UDP Communication
123     // Startup Winsock
124     WSADATA data;
125     WORD version = MAKEWORD(2, 2);
126     int wsOk = WSAStartup(version, &data);
127     if (wsOk != 0)
128     {
129         cout << "Can't start Winsock!" << wsOk;
130     }
131     cout << "start winsock" << endl;
132     SOCKET in = socket(AF_INET, SOCK_DGRAM, 0); // in 은 그냥 interger
133                                                 // UDP는 packet이 도착하는게 보
                    장되지 않음. 제어할 수 없음.
134                                                 // TCP_IP는 5개 도착할때까지 기
                    다림. 안도착한 packet을 재요청 및 기다림.
135                                                 /*struct timeval optVal =
                    { 10, 0 };
136                                                 int optLen = sizeof(optVal);
137                                                 setsockopt(in, SOL_SOCKET,
                    SO_RCVTIMEO, (char*)&optVal, optLen);*/
138     DWORD recvTO = 5000;
139     setsockopt(in, SOL_SOCKET, SO_RCVTIMEO, (char*)&recvTO, sizeof(recvTO));
140     sockaddr_in serverHint;
141     serverHint.sin_addr.S_un.S_addr = ADDR_ANY; // give me any address,
        whatever address give that to me
142     serverHint.sin_family = AF_INET;
143     serverHint.sin_port = htons(58430); // Conver from little to big endian //
        htons:host to network short
144
145     sockaddr_in client;
146     int clientLength = sizeof(client);
147     ZeroMemory(&client, clientLength);
148
149     // 2.3) UDP Communication
150     // Bind sokcet to ip address and port
151     if (bind(static_cast<SOCKET>(in), static_cast<const sockaddr*>((sockaddr*)
        &serverHint), static_cast<int>(sizeof(serverHint))) == SOCKET_ERROR) //
        Socket - IP - Port (Triple connection binding)
152     {
153         cout << "Can't bind socket! " << WSAGetLastError() << endl;
154     }
155
156     int Tacloop = 0;
157     char *Tacloop_S;
158     char buf[1024]; // message from client saved to buf
```

```
159
160     while (flag) {
161         cout << "Thread loop ... " << flag << endl;
162         /*char sendbuf[100] = "Server send\n";
163         int bytesIn = sendto(in, sendbuf, strlen(sendbuf), 0, (struct
              sockaddr*) &client, sizeof(client));
164         if(bytesIn == SOCKET_ERROR) {
165         cout << "Error sending to client " << WSAGetLastError() << endl;
166         continue;
167         }*/
168         ZeroMemory(buf, 1024);
169         // Wait for message
170         int bytesIn = recvfrom(in, buf, 1024, 0, (sockaddr*)&client,
              &clientLength); // recv 는 TCP용
171         if (bytesIn == SOCKET_ERROR)
172         {
173             cout << "Error receiving from client " << WSAGetLastError() <<
                  endl;
174             continue;
175         }
176         // Display message and client info
177         char clientIp[256];
178         ZeroMemory(clientIp, 256);
179
180         inet_ntop(AF_INET, &client.sin_addr, clientIp, 256); //version 4 IP
              address type is AF_INET
181
182                                                              //cout <<
                        "Message recv from " << clientIp << " : " << buf << endl;
183         char * pos;
184         char * context;
185
186         printf("원본: %s\n", buf);
187         //strtok_s 함수 이용
188         //printf("== 공백이나 콤마, 느낌표, 마침표를 기준으로 분할 ==\n");
189         pos = strtok_s(buf, "SBF():,", &context);  //처음 호출 시에 대상 문자열
              전달
190         Tacloop = atoi(pos);
191         printf("%d : ", Tacloop);
192         while (pos != NULL)
193         {
194             pos = strtok_s(context, "SBF():,", &context);//이 후 NULL 혹은
                  context 전달
195             if (pos != NULL) {
196                 printf("%f", atof(pos));
197             }
198         }
199         printf("\n Copy Done");
200     }
201     cout << "UDP Done" << endl;
202     closesocket(in);
203     // Shutdown Winsock
204     WSACleanup();
205     return;
206 }
207
```

```
208 void UDPthread_2(atomic<bool>& flag) {
209     // 1.3) UDP Communication
210     // Startup Winsock
211     WSADATA data;
212     WORD version = MAKEWORD(2, 2);
213     int wsOk = WSAStartup(version, &data);
214     if (wsOk != 0)
215     {
216         cout << "Can't start Winsock!" << wsOk;
217     }
218     cout << "start winsock" << endl;
219     SOCKET in = socket(AF_INET, SOCK_DGRAM, 0); // in 은 그냥 interger
220                                                 // UDP는 packet이 도착하는게 보장되지 않음. 제어할 수 없음.
221                                                 // TCP_IP는 5개 도착할때까지 기다림. 안도착한 packet을 재요청 및 기다림.
222                                                 /*struct timeval optVal =
223                     { 10, 0 };
224                                                 int optLen = sizeof(optVal);
225                                                 setsockopt(in, SOL_SOCKET,
226                     SO_RCVTIMEO, (char*)&optVal, optLen);*/
227     DWORD recvTO = 5000;
228     setsockopt(in, SOL_SOCKET, SO_RCVTIMEO, (char*)&recvTO, sizeof(recvTO));
229     sockaddr_in serverHint;
230     serverHint.sin_addr.S_un.S_addr = ADDR_ANY; // give me any address, whatever address give that to me
231     serverHint.sin_family = AF_INET;
232     serverHint.sin_port = htons(58432); // Conver from little to big endian // htons:host to network short
233
234     sockaddr_in client;
235     int clientLength = sizeof(client);
236     ZeroMemory(&client, clientLength);
237
238     // 2.3) UDP Communication
239     // Bind sokcet to ip address and port
240     if (bind(static_cast<SOCKET>(in), static_cast<const sockaddr*>((sockaddr*)&serverHint), static_cast<int>(sizeof(serverHint))) == SOCKET_ERROR) // Socket - IP - Port (Triple connection binding)
241     {
242         cout << "Can't bind socket! " << WSAGetLastError() << endl;
243     }
244
245     int Tacloop = 0;
246     char *Tacloop_S;
247     char buf[1024]; // message from client saved to buf
248
249     while (flag) {
250         cout << "Thread loop ... " << flag << endl;
251         /*char sendbuf[100] = "Server send\n";
252         int bytesIn = sendto(in, sendbuf, strlen(sendbuf), 0, (struct sockaddr*) &client, sizeof(client));
253         if(bytesIn == SOCKET_ERROR) {
254         cout << "Error sending to client " << WSAGetLastError() << endl;
255         continue;
256         }*/
```

```
255         ZeroMemory(buf, 1024);
256         // Wait for message
257         int bytesIn = recvfrom(in, buf, 1024, 0, (sockaddr*)&client,
              &clientLength); // recv 는 TCP용
258         if (bytesIn == SOCKET_ERROR)
259         {
260             cout << "Error receiving from client " << WSAGetLastError() <<
                endl;
261             continue;
262         }
263         // Display message and client info
264         char clientIp[256];
265         ZeroMemory(clientIp, 256);
266
267         inet_ntop(AF_INET, &client.sin_addr, clientIp, 256); //version 4 IP
              address type is AF_INET
268
269                                                              //cout <<
                      "Message recv from " << clientIp << " : " << buf << endl;
270         char * pos;
271         char * context;
272
273         printf("원본: %s\n", buf);
274         //strtok_s 함수 이용
275         //printf("== 공백이나 콤마, 느낌표, 마침표를 기준으로 분할 ==\n");
276         pos = strtok_s(buf, "SBF():,", &context);  //처음 호출 시에 대상 문자열
              전달
277         Tacloop = atoi(pos);
278         printf("%d : ", Tacloop);
279         while (pos != NULL)
280         {
281             pos = strtok_s(context, "SBF():,", &context);//이 후 NULL 혹은
                  context 전달
282             if (pos != NULL) {
283                 printf("%f", atof(pos));
284             }
285         }
286         printf("\n Copy Done");
287     }
288     cout << "UDP Done" << endl;
289     closesocket(in);
290     // Shutdown Winsock
291     WSACleanup();
292     return;
293 }
294
295 // Pico Flexx
296 class MyListener : public royale::IDepthDataListener
297 {
298
299 public:
300
301     MyListener() : undistortImage(false)
302     {
303     }
304
```

```
305
306     void onNewData(const royale::DepthData *data)
307     {
308         // this callback function will be called for every new depth frame
309
310         std::lock_guard<std::mutex> lock(flagMutex);
311
312         float *zRowPtr, *grayRowPtr = NULL;
313         // zImage
314         zImage.create(cv::Size(data->width, data->height), CV_32FC1);
315         zImage = cv::Scalar::all(0); // set the image to zero
316
317         int k = 0;
318         for (int y = 0; y < zImage.rows; y++)
319         {
320             zRowPtr = zImage.ptr<float>(y);
321
322             for (int x = 0; x < zImage.cols; x++, k++)
323             {
324                 auto curPoint = data->points.at(k);
325                 if (curPoint.depthConfidence > 0)
326                 {
327                     // if the point is valid, map the pixel from 3D world
328                     // coordinates to a 2D plane (this will distort the image)

329                     zRowPtr[x] = adjustZValue(curPoint.z);
330                 }
331                 else {
332                     zRowPtr[x] = 255; //distortion part => black
333                 }
334             }
335         }
336         zImage8.create(cv::Size(data->width, data->height), CV_8UC1);
337         zImage.convertTo(zImage8, CV_8UC1);     // normalize(zImage, zImage8,
              0, 255, NORM_MINMAX, CV_8UC1)
338
339         if (undistortImage)
340         {
341             // call the undistortion function on the z image
342             cv::Mat temp = zImage8.clone();
343             undistort(temp, zImage8, cameraMatrix, distortionCoefficients);
344         }
345
346         scaledZImage.create(cv::Size(data->width * 4, data->height * 4),
              CV_8UC1);       // scale and display the depth image
347         cv::resize(zImage8, scaledZImage, scaledZImage.size());
348
349         //cv::imshow("Depth", scaledZImage);
350
351         // grayImage
352         grayImage.create(cv::Size(data->width, data->height), CV_32FC1);
353         grayImage = cv::Scalar::all(0); // set the image to zero
354
355         k = 0;
356         for (int y = 0; y < grayImage.rows; y++)
357         {
```

```
358             grayRowPtr = grayImage.ptr<float>(y);
359
360             for (int x = 0; x < grayImage.cols; x++, k++)
361             {
362                 auto curPoint = data->points.at(k);
363                 if (curPoint.depthConfidence > 0)
364                 {
365                     // if the point is valid, map the pixel from 3D world
366                     // coordinates to a 2D plane (this will distort the image)
367                     grayRowPtr[x] = adjustGrayValue(curPoint.grayValue);
368                 }
369                 else {
370                     grayRowPtr[x] = 255; //distortion part => black
371                 }
372             }
373         }
374
375         grayImage8.create(cv::Size(data->width, data->height), CV_8UC1);
376         grayImage.convertTo(grayImage8, CV_8UC1);       // normalize
              (grayImage, grayImage8, 0, 255, NORM_MINMAX, CV_8UC1)
377
378         if (undistortImage)
379         {
380             // call the undistortion function on the gray image
381             cv::Mat temp = grayImage8.clone();
382             undistort(temp, grayImage8, cameraMatrix, distortionCoefficients);
383         }
384
385         // scale and display the gray image
386         scaledGrayImage.create(cv::Size(data->width * 4, data->height * 4),
              CV_8UC1);
387         cv::resize(grayImage8, scaledGrayImage, scaledGrayImage.size());
388
389         //cv::imshow("Gray", scaledGrayImage);
390
391         cv::Vec4f pOutLine;
392
393         imagetype = 0;
394
395         result_ZImage.create(cv::Size(data->width * 4, data->height * 4),
              CV_8UC1);
396         overlay_Bounding_Box(scaledZImage, data, result_ZImage, &pOutLine);
397         Kinova_calib(&pOutLine, &d_XYZt_Kinova);
398         cv::imshow("Test_Z", result_ZImage);
399
400         //imagetype = 1;
401         //result_GImage.create(Size(data->width * 4, data->height * 4),
              CV_8UC1);
402         //overlay_Bounding_Box(scaledGrayImage, data, result_GImage,
              &pOutLine);// read from gray image
403         //Kinova_calib(&pOutLine, &d_XYZt_Kinova);
404         //imshow("Test_G", result_GImage);
405     }
406
407     void Kinova_calib(cv::Vec4f* OutLine, cv::Vec3f *d_XYZt_Kinova) {
408         // Kinova Gripper Initial : (Z=0, theta=0) ===>> pico : (X =466/960,
```

```
            Y=125/720), real_world : (0.055 m, 0.02 m)
409         float dx = ((*OutLine)[2] * 0.0322 - 15)*0.01; //466
410         float dy = ((*OutLine)[3] * -0.032 + 4)*0.01; //125
411         float dth = atan((float)(*OutLine)[1] / (float)(*OutLine)[0]) - PI /
              2.0f + 0.2f;
412
413         *d_XYZt_Kinova = { dx, dy, dth };
414
415         /*  cout << " Out : " << dx << ", " << dy << ", " << dth << endl;
416         cout << " OutLine : " << *OutLine << endl;
417         cout << " d_Kinova : " << *d_XYZt_Kinova << endl;*/
418     }
419
420
421     void overlay_Bounding_Box(cv::Mat tImage, const royale::DepthData *data,
          cv::Mat result_tImage, cv::Vec4f *pOutLine) {
422
423         normalize(tImage, tImage8, 0, 255, cv::NORM_MINMAX, CV_8UC1);
424
425         if (imagetype == 0) { // Z image
426             cv::threshold(~tImage8, tImage8, 150, 255, cv::THRESH_BINARY +
                  cv::THRESH_OTSU);// +cv::THRESH_BINARY); cv::THRESH_OTSU
427
                    //Adaptive Thresholding을 한다.
428
                    //adaptiveThreshold(tImage8, tImage8, 255,
    ADAPTIVE_THRESH_MEAN_C, THRESH_BINARY, 5, 10);
429         }
430         else if (imagetype == 1) { // Gray image
431             cv::threshold(tImage8, tImage8, 127, 255, cv::THRESH_BINARY +
                  cv::THRESH_OTSU);// +cv::THRESH_BINARY);
432         }
433         cv::GaussianBlur(tImage8, tImage8, cv::Size(5, 5), 1, 1, 1);
434         cv::Mat tImageMat = tImage8.clone();
435
436
437         double area, max_area = 0;
438         double min_err = 10000;
439         obj_id = 0;
440         cv::Rect bounding_rect;
441         vector<cv::Vec4i> hierarchy;
442         vector<vector<cv::Point> > contours;
443         findContours(tImageMat, contours, hierarchy, cv::RETR_CCOMP,
              cv::CHAIN_APPROX_SIMPLE);
444         for (int i = 0; i < contours.size(); i++) {
445             /* // Maximum Area Detection
446             if (area > max_area) {
447             max_area = area;
448             max_id = i;
449             bounding_rect = boundingRect(contours[i]);
450             }*/
451             area = contourArea(contours[i], false);
452             if (area > 5000 & area < 80000) {
453                 //cout << "Area :" << area << endl;
454                 cv::Moments M = cv::moments(contours[i]);
455                 int cX = int(M.m10 / M.m00);
```

```
456                 int cY = int(M.m01 / M.m00);
457                 // cout << "CX" << cX << "CY" << cY << endl << endl;
458                 double d_err = sqrt((cX - 466) ^ 2 + (cY - 125) ^ 2);
459                 if (d_err < min_err) {
460                     min_err = d_err;
461                     obj_id = i;
462                     bounding_rect = boundingRect(contours[i]);
463                 }
464             }
465 
466         }
467 
468         rectangle(tImage8, bounding_rect, cv::Scalar(100, 100, 100), 1, 0);
469 
470         cv::Vec4f OutLine;
471         fitLine(cv::Mat(contours[obj_id]), OutLine, cv::DIST_L2, 0, 0.01,
              0.01);
472 
473         //cout << "BR :" << bounding_rect.x << "  " << bounding_rect.width <<
              endl;
474         //cout << "zImage.rows :" << zImage.rows << "zImage.cols :" <<
              zImage.cols << endl;
475 
476         // Find Down Z value
477         int k = 0;
478         float min_z = 1000;
479         for (int y = 0; y < zImage.rows; y++)
480         {
481             for (int x = 0; x < zImage.cols; x++, k++)
482             {
483                 auto curPoint = data->points.at(k);
484                 if (((bounding_rect.x / 4) < (k % zImage.rows)) &
                      (((bounding_rect.x / 4) + (bounding_rect.width / 4) - 1) >(k
                       % zImage.rows))) {
485                     if (min_z > curPoint.z & curPoint.z > 0) {
486                         min_z = curPoint.z;
487                     }
488                 }
489             }
490         }
491         if (min_z < 1) {
492             Down_Z = min_z;
493         }
494         //cout << " Z :" << Down_Z << endl;
495 
496 
497         /****************/
498         //cout << "OutLine Data" << OutLine << endl;
499         cv::line(tImage8, cv::Point(OutLine[2], OutLine[3]), cv::Point(100 *
              OutLine[0] + OutLine[2], 100 * OutLine[1] + OutLine[3]), cv::Scalar
              (50, 50, 50), 2);
500         //line(tImage, Point(0,0), Point(200,200), Scalar(50, 50, 50), 2);
501         cv::circle(tImage8, cv::Point(OutLine[2], OutLine[3]), 4, cv::Scalar
              (0, 255, 0), 2);
502         cv::circle(tImage8, cv::Point(20 * OutLine[0] + OutLine[2], 20 *
              OutLine[1] + OutLine[3]), 4, cv::Scalar(0, 255, 0), 2);
```

```
503
504         //Sleep(1);
505         /*scaledtImage.create(Size(data->width * 4, data->height * 4),
              CV_8UC1);
506         resize(tImage8, scaledtImage, scaledtImage.size());
507         imshow("Test", scaledtImage);*/
508         *pOutLine = OutLine;
509
510         resize(tImage8, result_tImage, result_tImage.size());
511     }
512
513     void setLensParameters(const royale::LensParameters &lensParameters)
514     {
515         // Construct the camera matrix
516         // (fx   0    cx)
517         // (0    fy   cy)
518         // (0    0    1 )
519         cameraMatrix = (cv::Mat1d(3, 3) << lensParameters.focalLength.first,
              0, lensParameters.principalPoint.first,
520             0, lensParameters.focalLength.second,
                  lensParameters.principalPoint.second,
521             0, 0, 1);
522
523         // Construct the distortion coefficients
524         // k1 k2 p1 p2 k3
525         distortionCoefficients = (cv::Mat1d(1, 5) <<
              lensParameters.distortionRadial[0],
526             lensParameters.distortionRadial[1],
527             lensParameters.distortionTangential.first,
528             lensParameters.distortionTangential.second,
529             lensParameters.distortionRadial[2]);
530     }
531
532     void toggleUndistort()
533     {
534         std::lock_guard<std::mutex> lock(flagMutex);
535         undistortImage = !undistortImage;
536     }
537
538 private:
539
540     // adjust z value to fit fixed scaling, here max dist is 2.5m
541     // the max dist here is used as an example and can be modified
542     float adjustZValue(float zValue)
543     {
544         float clampedDist = std::min(0.44f, zValue);
545         float newZValue = clampedDist / 0.44f * 255.0f;
546         return newZValue;
547     }
548
549     // adjust gray value to fit fixed scaling, here max value is 180
550     // the max value here is used as an example and can be modified
551     float adjustGrayValue(uint16_t grayValue)
552     {
553         float clampedVal = std::min(2350.0f, grayValue * 1.0f); // 180.0f
              (Original clamp Value)
```

```
554         float newGrayValue = clampedVal / 2350.f * 255.0f;
555         return newGrayValue;
556     }
557
558     // define images for depth and gray
559     // and for their 8Bit and scaled versions
560     cv::Mat zImage, zImage8, scaledZImage, result_ZImage;
561     cv::Mat grayImage, grayImage8, scaledGrayImage, result_GImage;
562
563     // 2018Oct27 YJ
564     cv::Mat tImage, tImage8, scaledtImage; //Test image
565
566                                             // lens matrices used for the undistortion of
567                                             // the image
568     cv::Mat cameraMatrix;
569     cv::Mat distortionCoefficients;
570
571     std::mutex flagMutex;
572     bool undistortImage;
573 };
574
575 // Keyboard handler
576 void keyboard(unsigned char key, int x, int y)
577 {
578     switch (key)
579     {
580         // On Esc key press
581     case 27:
582     {
583         sample.release();
584         int result = (*MyCloseAPI)();
585         FreeLibrary(commandLayer_handle);
586
587         glutDestroyWindow(glutGetWindow());
588         exit(EXIT_FAILURE);
589     }
590
591     default:
592     {
593         // Do nothing otherwise
594         break;
595     }
596     }
597 }
598
599 void mouseClick(int button, int state, int x, int y)
600 {
601     if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
602     {
603         sample.nextViewMode();
604     }
605 }
606
607
608 // Update tracking data and visualize it
```

```
609 void display()
610 {
611     // Delegate this action to example's main class
612     bool update = sample.update();
613 
614     if (!update)
615     {
616         // End the work if update failed
617         sample.release();
618         int result = (*MyCloseAPI)();
619         FreeLibrary(commandLayer_handle);
620         glutDestroyWindow(glutGetWindow());
621         exit(EXIT_FAILURE);
622     }
623 
624     // Do flush and swap buffers to update viewport
625     glFlush();
626     glutSwapBuffers();
627 }
628 
629 void idle()
630 {
631     glutPostRedisplay();
632     //printf("Width : %d, Height : %d \n", sample.getWidth(), sample.getHeight
        ());
633     RHandPos = sample.getRHandPos();
634     LHandPos = sample.getLHandPos();
635     HeadPos = sample.getHeadPos();
636     //printf("Rx: %f Ry: %f Rz: %f // Lx : %f Ly: %f Lz: %f \n", *(RHandPos),
        *(RHandPos + 1), *(RHandPos + 2), *(LHandPos), *(LHandPos + 1), *
        (LHandPos + 2));
637     float R_X = (*(RHandPos))*0.001f - 0.20;
638     float R_Y = (-sin(PI / 4.0f)*(*(RHandPos + 2) * 0.001) - sin(PI / 4.0f)*(*
        (RHandPos + 1) * 0.001) + 0.16);
639     float R_Z = (-sin(PI / 4.0f)*(*(RHandPos + 2) * 0.001) + sin(PI / 4.0f)*(*
        (RHandPos + 1) * 0.001) + 1.50);
640     float L_X = (*(LHandPos))*0.001f - 0.20;
641     float L_Y = (-sin(PI / 4.0f)*(*(LHandPos + 2) * 0.001) - sin(PI / 4.0f)*(*
        (LHandPos + 1) * 0.001) + 0.16);
642     float L_Z = (-sin(PI / 4.0f)*(*(LHandPos + 2) * 0.001) + sin(PI / 4.0f)*(*
        (LHandPos + 1) * 0.001) + 1.50);
643     float H_X = (*(HeadPos))*0.001f - 0.20;
644     float H_Y = (-sin(PI / 4.0f)*(*(HeadPos + 2) * 0.001) - sin(PI / 4.0f)*(*
        (HeadPos + 1) * 0.001) + 0.16);
645     float H_Z = (-sin(PI / 4.0f)*(*(HeadPos + 2) * 0.001) + sin(PI / 4.0f)*(*
        (HeadPos + 1) * 0.001) + 1.50);
646     printf("Rx: %f Ry: %f Rz: %f // Lx: %f Ly: %f Lz: %f // Hx: %f Hy: %f Hz:
        %f \n", R_X, R_Y, R_Z, L_X, L_Y, L_Z, H_X, H_Y, H_Z);
647 
648     goal[0] = R_X;
649     goal[1] = R_Y+0.6f; // 0.5f*H_Y - (R_Y - 0.5f*H_Y);
650     goal[2] = R_Z;
651 
652     CartesianPosition currentPosition;
653     MyGetCartesianCommand(currentPosition);
654     rob_pos[0] = currentPosition.Coordinates.X;
```

```
655     rob_pos[1] = currentPosition.Coordinates.Y;
656     rob_pos[2] = currentPosition.Coordinates.Z;
657     Dtogoal = sqrt(pow(goal[0] - rob_pos[0] - momentum[0], 2) + pow(goal[1] -
          rob_pos[1] - momentum[1], 2) + pow(goal[2] - rob_pos[2] - momentum[2],
          2));
658
659
660     //reset PF
661     GUrep_bnd[0] = 0;
662     GUrep_bnd[1] = 0;
663     GUrep_bnd[2] = 0;
664     GUrep_obs[0] = 0;
665     GUrep_obs[1] = 0;
666     GUrep_obs[2] = 0;
667
668     temp[0] = rob_pos[0] + rate*momentum[0];
669     temp[1] = rob_pos[1] + rate*momentum[1];
670     temp[2] = rob_pos[2] + rate*momentum[2];
671
672     ////goal update
673     //if ((abs(currentPosition.Coordinates.X - goal[0]) > 0.3)
674     //  || (abs(currentPosition.Coordinates.Y - goal[1]) > 0.3)
675     //  || (abs(currentPosition.Coordinates.Z - goal[2]) > 0.3)
676     //  )
677     //{
678     //  std::cout << "You moved too fast!" << endl;
679     //  return;
680     //}
681
682     //boundary PF at temp
683     for (int i = 0; i < 3; i++)
684     {
685         D = abs(bnd[0][i] - temp[i]) < abs(bnd[1][i] - temp[i]) ? abs(bnd[0]
              [i] - temp[i]) : abs(bnd[1][i] - temp[i]);
686         Cons = Nu*(1.0 / ObsTh - 1.0 / D)*pow(1.0 / D, 2); //negative
687         DtoCenter = sqrt(pow(bnd_center[0] - temp[0], 2) + pow(bnd_center[1] -
              temp[1], 2) + pow(bnd_center[2] - temp[2], 2));
688         if (D <= ObsTh)
689         {
690             GUrep_bnd[i] = Cons*((bnd_center[i] - temp[i]) / DtoCenter);
691         }
692     }
693
694     //obstacles PF at temp
695     for (int i = 0; i < obsnum; i++)
696     {
697         D = sqrt(pow(obs[i][0] - temp[0], 2) + pow(obs[i][1] - temp[1], 2) +
              pow(obs[i][2] - temp[2], 2)) - obs[i][3];
698         if (D <= ObsTh)
699         {
700             DtoCenter = sqrt(pow(obs[i][0] - temp[0], 2) + pow(obs[i][1] -
                  temp[1], 2) + pow(obs[i][2] - temp[2], 2));
701             Cons = Nu*(1.0 / ObsTh - 1.0 / D)*pow(1.0 / D, 2); //negative
702             GUrep_obs[0] = GUrep_obs[0] + Cons*((temp[0] - obs[i][0]) /
                  DtoCenter); // x direction
703             GUrep_obs[1] = GUrep_obs[1] + Cons*((temp[1] - obs[i][1]) /
```

```
                DtoCenter); // y direction
704             GUrep_obs[2] = GUrep_obs[2] + Cons*((temp[2] - obs[i][2]) /
                DtoCenter); // z direction
705         }
706     }
707
708     GUrep[0] = GUrep_bnd[0] + GUrep_obs[0];
709     GUrep[1] = GUrep_bnd[1] + GUrep_obs[1];
710     GUrep[2] = GUrep_bnd[2] + GUrep_obs[2];
711
712     GUatt[0] = Kappa * (temp[0] - goal[0]);
713     GUatt[1] = Kappa * (temp[1] - goal[1]);
714     GUatt[2] = Kappa * (temp[2] - goal[2]);
715
716     gradient[0] = -GUrep[0] - GUatt[0];
717     gradient[1] = -GUrep[1] - GUatt[1];
718     gradient[2] = -GUrep[2] - GUatt[2];
719
720     norm_gradient = sqrt(pow(gradient[0], 2) + pow(gradient[1], 2) + pow
        (gradient[2], 2));
721
722     //momentum(delta pos) = rate*previous momentum + PF at temp
723     momentum[0] = rate * momentum[0] + stepsize*gradient[0] / norm_gradient;
724     momentum[1] = rate * momentum[1] + stepsize*gradient[1] / norm_gradient;
725     momentum[2] = rate * momentum[2] + stepsize*gradient[2] / norm_gradient;
726     norm_momentum = sqrt(pow(momentum[0], 2) + pow(momentum[1], 2) + pow
        (momentum[2], 2));
727     momentum[0] = stepsize * momentum[0] / norm_momentum;
728     momentum[1] = stepsize * momentum[1] / norm_momentum;
729     momentum[2] = stepsize * momentum[2] / norm_momentum;
730
731     //send the robot to next pos
732     TrajectoryPoint pointToSend;
733     pointToSend.InitStruct();
734     pointToSend.Position.Type = CARTESIAN_POSITION;
735
736     pointToSend.Position.CartesianPosition.X = rob_pos[0] + momentum[0];
737     pointToSend.Position.CartesianPosition.Y = rob_pos[1] + momentum[1];
738     pointToSend.Position.CartesianPosition.Z = rob_pos[2] + momentum[2];
739     pointToSend.Position.CartesianPosition.ThetaX =
        currentPosition.Coordinates.ThetaX;
740     pointToSend.Position.CartesianPosition.ThetaY =
        currentPosition.Coordinates.ThetaY;
741     pointToSend.Position.CartesianPosition.ThetaZ =
        currentPosition.Coordinates.ThetaZ;
742
743
744     Dtogoal = sqrt(pow(goal[0] - rob_pos[0] - momentum[0], 2) + pow(goal[1] -
        rob_pos[1] - momentum[1], 2) + pow(goal[2] - rob_pos[2] - momentum[2],
        2));
745     numloop = numloop + 1;
746     MySendBasicTrajectory(pointToSend);
747
748     std::cout << numloop << endl;
749     std::cout << "rob X : " << rob_pos[0] << "  rob Y : " << rob_pos[1] << "
              rob Z : " << rob_pos[2] << endl;
```

```
750     std::cout << "delta X : " << momentum[0] << "   delta Y : " << momentum[1] ↵
           << "   delta Z : " << momentum[2] << endl;
751     std::cout << "goal X : " << goal[0] << "    goal Y : " << goal[1] << "     ↵
          goal Z : " << goal[2] << endl << endl;
752
753     Sleep(80);
754 }
755
756 void showHelpInfo()
757 {
758     std::cout << "Usage: nuitrack_gl_sample [path/to/nuitrack.config]\n"
759         "Press Esc to close window." << std::endl;
760 }
761
762 void InitializeDynamixel() {
763     char input[MAX_IN_CHAR];
764     char *token, *context;
765     char param[20][30];
766     char cmd[80];
767     int port_num = 0, baud_rate = 0, input_len, num_param;
768
769
770     port_num = 3;
771     baud_rate = 57600;
772
773     printf("\nYour input info. is\n");
774     printf("COM port number : %d\n", port_num);
775     printf("       Baudrate : %d\n", baud_rate);
776
777
778     if (dxl_initialize(port_num, baud_rate) == 0)
779     {
780         printf("Failed to open USB2Dynamixel!\n");
781         printf("Press any key to terminate...\n");
782         _getch();
783         return;
784     }
785     else
786         printf("Succeed to open USB2Dynamixel!\n\n");
787
788
789     printf("\n");
790     printf("Ping Using Protocol 2.0\n");
791     for (int i = 1; i < 3; i++)
792     {
793         dxl2_ping(i);
794         if (dxl_get_comm_result() == COMM_RXSUCCESS)
795             printf("                    ... SUCCESS \r");
796         else
797             printf("                    ... FAIL \r");
798         printf(" CHECK ID : %d \n", i);
799     }
800     printf("\n");
801 }
802
803 int main(int argc, char* argv[])
```

```
804 {
805     commandLayer_handle = LoadLibrary(L"CommandLayerWindows.dll");
806
807     //We load the functions from the library
808     MyInitAPI = (int(*)()) GetProcAddress(commandLayer_handle, "InitAPI");
809     MyCloseAPI = (int(*)()) GetProcAddress(commandLayer_handle, "CloseAPI");
810     MyMoveHome = (int(*)()) GetProcAddress(commandLayer_handle, "MoveHome");
811     MyInitFingers = (int(*)()) GetProcAddress(commandLayer_handle,
        "InitFingers");
812     MyGetDevices = (int(*)(KinovaDevice devices[MAX_KINOVA_DEVICE], int
        &result)) GetProcAddress(commandLayer_handle, "GetDevices");
813     MySetActiveDevice = (int(*)(KinovaDevice devices)) GetProcAddress
        (commandLayer_handle, "SetActiveDevice");
814     MySendBasicTrajectory = (int(*)(TrajectoryPoint)) GetProcAddress
        (commandLayer_handle, "SendBasicTrajectory");
815     MyGetCartesianCommand = (int(*)(CartesianPosition &)) GetProcAddress
        (commandLayer_handle, "GetCartesianCommand");
816
817     //Verify that all functions has been loaded correctly
818     if ((MyInitAPI == NULL) || (MyCloseAPI == NULL) || (MySendBasicTrajectory
        == NULL) ||
819         (MyGetDevices == NULL) || (MySetActiveDevice == NULL) ||
            (MyGetCartesianCommand == NULL) ||
820         (MyMoveHome == NULL) || (MyInitFingers == NULL))
821
822     {
823         std::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;
824         programResult = 0;
825         return 0;
826     }
827     else
828     {
829         std::cout << "I N I T I A L I Z A T I O N   C O M P L E T E D - M A I
            N" << endl << endl;
830     }
831     int result = (*MyInitAPI)();
832
833     std::cout << "Main Initialization's result :" << result << endl;
834
835     KinovaDevice list[MAX_KINOVA_DEVICE];
836
837     int devicesCount = MyGetDevices(list, result);
838
839     std::cout << "Found a robot on the USB bus (" << list[0].SerialNumber <<
        ")" << endl;
840
841     // Setting the current device as the active device.
842     MySetActiveDevice(list[0]);
843
844     std::cout << "Send the Robot to Home Position" << endl;
845     MyMoveHome();
846     Sleep(3000);
847
848     std::cout << "Send the Robot to Initial Position" << endl;
849     TrajectoryPoint HomePosition;
```

```
850     HomePosition.InitStruct();
851     HomePosition.Position.Type = ANGULAR_POSITION;
852     HomePosition.Position.Actuators.SetValues(360, 143.3, -85, 53.3, 220,
          251.9, 300);
853     MySendBasicTrajectory(HomePosition); Sleep(2000);
854     HomePosition.Position.Actuators.SetValues(445.8, 143.3, -169.8, 53.3,
          181.9, 251.9, 322.1);
855     MySendBasicTrajectory(HomePosition); Sleep(2000);
856     std::cout << "Initializing Fingers" << endl;
857
858     InitializeDynamixel();
859     Read2(1, 132, 4); Read2(2, 132, 4);
860     Write2(1, 64, 1, 4); Write2(2, 64, 1, 4); // Torque ON
861     Write2(1, 116, 620, 4); Write2(2, 116, 460, 4); Sleep(500);
862     Write2(1, 116, 450, 4); Write2(2, 116, 290, 4); Sleep(500);
863     // Write2(1, 64, 0, 4); Write2(2, 64, 0, 4); // Torque OFF
864
865     // UDP Thread Initialization
866     boost::thread U_thread_1 = boost::thread(&UDPthread_1, ref(flag));
867     boost::thread U_thread_2 = boost::thread(&UDPthread_2, ref(flag));
868
869     // Nuitrack Initialize
870     showHelpInfo();
871     sample.init();
872     auto outputMode = sample.getOutputMode();
873     // Initialize GLUT window
874     glutInit(&argc, argv);
875     glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
876     glutInitWindowSize(outputMode.xres, outputMode.yres);
877     glutCreateWindow("Nuitrack GL Sample (Nuitrack API)");
878     //glutSetCursor(GLUT_CURSOR_NONE);
879
880     // Connect GLUT callbacks
881     glutKeyboardFunc(keyboard);
882     glutDisplayFunc(display);
883     glutIdleFunc(idle);
884     glutMouseFunc(mouseClick);
885
886     // Setup OpenGL
887     glDisable(GL_DEPTH_TEST);
888     glEnable(GL_TEXTURE_2D);
889
890     glEnableClientState(GL_VERTEX_ARRAY);
891     glDisableClientState(GL_COLOR_ARRAY);
892
893
894     glOrtho(0, outputMode.xres, outputMode.yres, 0, -1.0, 1.0);
895     glMatrixMode(GL_PROJECTION);
896     glPushMatrix();
897     glLoadIdentity();
898
899     // Pico Flexx Camera Initialization
900     sample_utils::PlatformResources resources;
901     MyListener Picolistener;
902     std::unique_ptr<royale::ICameraDevice> cameraDevice;
903     {
```

```
904         royale::CameraManager manager;
905
906         royale::Vector<royale::String> camlist(manager.getConnectedCameraList
              ());
907         cout << "Detected " << camlist.size() << "pico camera(s)." << endl;
908
909         if (!camlist.empty())
910         {
911             cameraDevice = manager.createCamera(camlist[0]);
912         }
913         else
914         {
915             cerr << "No suitable camera device detected." << endl;
916             return 1;
917         }
918
919         camlist.clear();
920     }
921
922     if (cameraDevice == nullptr)
923     {
924         // no cameraDevice available
925         cerr << "Cannot create the camera device" << endl;
926         return 1;
927     }
928
929     // IMPORTANT: call the initialize method before working with the camera
          device
930     auto status = cameraDevice->initialize();
931     if (status != royale::CameraStatus::SUCCESS)
932     {
933         cerr << "Cannot initialize the camera device, error string : " <<
              getErrorString(status) << endl;
934         return 1;
935     }
936
937     royale::String usecaseName = "MODE_5_35FPS_600"; // MODE_9_5FPS_2000,
            MODE_9_10FPS_1000,      MODE_9_15FPS_700,       MODE_9_25FPS_450,
938                                                       // MODE_5_35FPS_600,
                        MODE_5_45FPS_500,       MODE_MIXED_30_5
939
940     status = cameraDevice->setUseCase(usecaseName);
941     if (status == royale::CameraStatus::SUCCESS) {
942         cout << " The use cases is successfully adapted to " << usecaseName <<
              endl;
943     }
944     else {
945         cout << "Fail to set use case." << endl;
946     }
947
948     // retrieve the lens parameters from Royale
949     royale::LensParameters lensParameters;
950     status = cameraDevice->getLensParameters(lensParameters);
951     if (status != royale::CameraStatus::SUCCESS)
952     {
953         cerr << "Can't read out the lens parameters" << endl;
```

```
954         return 1;
955     }
956
957     Picolistener.setLensParameters(lensParameters);
958     Picolistener.toggleUndistort(); //한번 toggle하고 시작
959
960                                     // register a data listener
961     if (cameraDevice->registerDataListener(&Picolistener) !=
          royale::CameraStatus::SUCCESS)
962     {
963         cerr << "Error registering data listener" << endl;
964         return 1;
965     }
966
967     // create windows
968     //cv::namedWindow("Depth", cv::WINDOW_AUTOSIZE);
969     //cv::namedWindow("Gray", cv::WINDOW_AUTOSIZE);
970     //cv::namedWindow("Test_G", WINDOW_AUTOSIZE);
971     cv::namedWindow("Test_Z", cv::WINDOW_AUTOSIZE);
972
973     // start capture mode
974     if (cameraDevice->startCapture() != royale::CameraStatus::SUCCESS)
975     {
976         cerr << "Error starting the capturing" << endl;
977         return 1;
978     }
979
980     // Pico scanning
981
982     std::cout << "**********START**********" <<
          endl;
983
984     // Start main loop
985     glutMainLoop();
986
987     // Terminate
988     cout << "stopping thread" << endl;
989     flag = false;
990     U_thread_1.join();
991     U_thread_2.join();
992     dxl_terminate();
993     return programResult;
994 }
```