Anexos

Anexo 1:

```
// Programa para el movimiento predeterminado del brazo robótico

#include <iostream>
#include <dlfcn.h> //Ubuntu
#include <vector>
#include "KinovaTypes.h"
//#include <time.h>

using namespace std;

int main()
{
      int result;

      cout << " Envío de trajectoria..." << endl;

      void * commandLayer_handle;

      int (*MyInitAPI)();
      int (*MyCloseAPI)();
      int (*MySendAdvanceTrajectory)(TrajectoryPoint command);
      int (*MyStartControlAPI)();

      commandLayer_handle =
dlopen("Kinova.API.USBCommandLayerUbuntu.so",RTLD_NOW|RTLD_GLOBAL);

      MyInitAPI = (int (*)()) dlsym(commandLayer_handle,"InitAPI");
      MyCloseAPI = (int (*)()) dlsym(commandLayer_handle,"CloseAPI");
      MySendAdvanceTrajectory = (int (*)(TrajectoryPoint))
dlsym(commandLayer_handle,"SendAdvanceTrajectory");
      MyStartControlAPI = (int (*)())
dlsym(commandLayer_handle,"StartControlAPI");

      if((MyInitAPI == NULL) || (MyCloseAPI == NULL) ||
(MySendAdvanceTrajectory == NULL) || (MyStartControlAPI == NULL))
      {
            cout << "Incapaz de realizer el movimiento." << endl;
      }
      else
      {
            cout << "El programa fue inicializado correctamente." << endl <<
endl;

            cout << "Calling the method InitAPI()" << endl;
            result = (*MyInitAPI)();
            cout << "result of InitAPI() = " << result << endl << endl;

            cout << "Se toma el control del brazo robótico." << endl;
            result = (*MyStartControlAPI)();

            TrajectoryPoint trajectoryPoint;

            trajectoryPoint.InitStruct();

            trajectoryPoint.Position.HandMode = POSITION_MODE;

            trajectoryPoint.Position.Fingers.Finger1 = 6;
            trajectoryPoint.Position.Fingers.Finger2 = 6;

            trajectoryPoint.Position.CartesianPosition.X = -0.0311603f;
            trajectoryPoint.Position.CartesianPosition.Y = -0.391776f;
            trajectoryPoint.Position.CartesianPosition.Z = 0.0881645f;

            trajectoryPoint.Position.CartesianPosition.ThetaX = 1.55669f;
```

```
trajectoryPoint.Position.CartesianPosition.ThetaY = 0.243988f;
trajectoryPoint.Position.CartesianPosition.ThetaZ = -0.0416119f;

(*MySendAdvanceTrajectory)(trajectoryPoint);

sleep(5);

trajectoryPoint.Position.CartesianPosition.X = -0.0314688f;
trajectoryPoint.Position.CartesianPosition.Y = -0.487803f;
trajectoryPoint.Position.CartesianPosition.Z = 0.082571f;

trajectoryPoint.Position.CartesianPosition.ThetaX = 1.56742f;
trajectoryPoint.Position.CartesianPosition.ThetaY = 0.255597f;
trajectoryPoint.Position.CartesianPosition.ThetaZ = -0.03503f;

(*MySendAdvanceTrajectory)(trajectoryPoint);

trajectoryPoint.Position.Fingers.Finger1 = 6000;
trajectoryPoint.Position.Fingers.Finger2 = 6000;

          (*MySendAdvanceTrajectory)(trajectoryPoint);

sleep(3);

trajectoryPoint.Position.CartesianPosition.X = -0.6051f;
trajectoryPoint.Position.CartesianPosition.Y = 0.0139847f;
trajectoryPoint.Position.CartesianPosition.Z = 0.319085f;

trajectoryPoint.Position.CartesianPosition.ThetaX = 1.59435f;
trajectoryPoint.Position.CartesianPosition.ThetaY = -1.48822f;
trajectoryPoint.Position.CartesianPosition.ThetaZ = -0.016619f;

          (*MySendAdvanceTrajectory)(trajectoryPoint);

sleep(3);

trajectoryPoint.Position.CartesianPosition.ThetaZ = 1.13173f;

          (*MySendAdvanceTrajectory)(trajectoryPoint);

sleep(4);


trajectoryPoint.Position.CartesianPosition.ThetaZ = 1.5305f;

          (*MySendAdvanceTrajectory)(trajectoryPoint);

sleep(2);


trajectoryPoint.Position.CartesianPosition.ThetaZ = 1.0305f;

          (*MySendAdvanceTrajectory)(trajectoryPoint);

sleep(2);


trajectoryPoint.Position.CartesianPosition.ThetaZ = -0.016619f;

          (*MySendAdvanceTrajectory)(trajectoryPoint);

sleep(3);

trajectoryPoint.Position.CartesianPosition.X = -0.0314688f;
trajectoryPoint.Position.CartesianPosition.Y = -0.487803f;
trajectoryPoint.Position.CartesianPosition.Z = 0.082571f;

trajectoryPoint.Position.CartesianPosition.ThetaX = 1.56742f;
```

```
                trajectoryPoint.Position.CartesianPosition.ThetaY = 0.255597f;
                trajectoryPoint.Position.CartesianPosition.ThetaZ = -0.03503f;

                        (*MySendAdvanceTrajectory)(trajectoryPoint);

                sleep(4);

                trajectoryPoint.Position.Fingers.Finger1 = 6;
                trajectoryPoint.Position.Fingers.Finger2 = 6;

                        (*MySendAdvanceTrajectory)(trajectoryPoint);

                sleep(2);

                trajectoryPoint.Position.Fingers.Finger1 = 4000;
                trajectoryPoint.Position.Fingers.Finger2 = 4000;

                trajectoryPoint.Position.CartesianPosition.X = -0.0311603f;
                trajectoryPoint.Position.CartesianPosition.Y = -0.391776f;
                trajectoryPoint.Position.CartesianPosition.Z = 0.0881645f;

                trajectoryPoint.Position.CartesianPosition.ThetaX = 1.55669f;
                trajectoryPoint.Position.CartesianPosition.ThetaY = 0.243988f;
                trajectoryPoint.Position.CartesianPosition.ThetaZ = -0.0416119f;

                        (*MySendAdvanceTrajectory)(trajectoryPoint);

                sleep(2);

                trajectoryPoint.Position.CartesianPosition.X = 0.210235f;
                trajectoryPoint.Position.CartesianPosition.Y = -0.260805f;
                trajectoryPoint.Position.CartesianPosition.Z = 0.475993f;

                trajectoryPoint.Position.CartesianPosition.ThetaX = 1.56256f;
                trajectoryPoint.Position.CartesianPosition.ThetaY = 1.14795f;
                trajectoryPoint.Position.CartesianPosition.ThetaZ = -0.0385581f;

                        (*MySendAdvanceTrajectory)(trajectoryPoint);

                cout << endl << "Calling the method CloseAPI()" << endl;
                result = (*MyCloseAPI)();
                cout << "result of CloseAPI() = " << result << endl;
        }

        return 0;
}
```

Anexo 2:

```
// Programa de reconocimiento de gestos faciales y movimientos
articulares de la cabeza para el control del brazo robótico Jaco de
Kinova de forma incremental y con movimientos predeterminados

#include "opencv2/core/core.hpp"
#include "opencv2/contrib/contrib.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/objdetect/objdetect.hpp"

#include <iostream>
#include <fstream>
#include <sstream>
#include <unistd.h>
#include <vector>

using namespace cv;
using namespace std;
int kk, ll;

static void read_csv(const string& filename, vector<Mat>& images,
vector<int>& labels, char separator = ';') {
    std::ifstream file(filename.c_str(), ifstream::in);
    if (!file) {
        string error_message = "No valid input file was given,
please check the given filename.";
        CV_Error(CV_StsBadArg, error_message);
    }
    string line, path, classlabel;
    while (getline(file, line)) {
        stringstream liness(line);
        getline(liness, path, separator);
        getline(liness, classlabel);
        if(!path.empty() && !classlabel.empty()) {
            images.push_back(imread(path, 0));
            labels.push_back(atoi(classlabel.c_str()));
        }
    }
}

int main(int argc, const char *argv[]) {

    // Revisa argumento de linea de commando correctos
    if (argc != 5) {
        cout << "usage: " << argv[0] << " </path/to/haar_cascade>
</path/to/csv.ext> </path/to/device id>" << endl;
        cout << "\t </path/to/haar_cascade> -- Path to the Haar
Cascade for face detection." << endl;
        cout << "\t </path/to/csv.ext> -- Path to the CSV file with
the face database." << endl;
        cout << "\t <device id> -- The webcam device id to grab
frames from." << endl;
        cout << "\t </path/to/csv2.ext> -- Path to the CSV file with
the face database." << endl;
        exit(1);
    }
    // Get the path to your CSV:
    string fn_haar = string(argv[1]);
    string fn_csv = string(argv[2]);
    int deviceId = atoi(argv[3]);
```

```
    string fn_csv2 = string(argv[4]);


for(;;) {

selector:
//cout << "Proceso inicializado" << endl;
    // Este vector mantiene a la imagen con su correspondiente
etiqueta:
    vector<Mat> images;
    vector<int> labels;
    // Lectura de la data
    if (kk==0){
     try {
          read_csv(fn_csv, images, labels);
         } catch (cv::Exception& e) {
          cerr << "Error abriendo el archivo \"" << fn_csv << "\".
Reason: " << e.msg << endl;
          // nada mas se puede hacer
          exit(1);
         }
    }

    if (kk==1){
     ll=1;
     try {
          read_csv(fn_csv2, images, labels);
         } catch (cv::Exception& e) {
          cerr << "Error abriendo el archivo \"" << fn_csv << "\".
Reason: " << e.msg << endl;
          // nothing more we can do
          exit(1);
         }
    }
    int im_width = images[0].cols;
    int im_height = images[0].rows;
    // Create a FaceRecognizer and train it on the given images:
    Ptr<FaceRecognizer> model = createEigenFaceRecognizer();
    model->train(images, labels);
    CascadeClassifier haar_cascade;
    haar_cascade.load(fn_haar);

    // Conseguir el permiso para usar la cámara:
    VideoCapture cap(deviceId);
    // Revisar si tenemos permiso para usar la cámara por completo:
    if(!cap.isOpened()) {
        cerr << "Capture Device ID " << deviceId << "No puede ser
abierto." << endl;
        return -1;
    }
    // Mantener el cuadro actual para el procesamiento:
    Mat frame;
    for(;;) {
        cap >> frame;
        // Clona la imagen actual:
        Mat original = frame.clone();
        // Convertir a escala de grises:

        Mat gray;

        cvtColor(original, gray, CV_BGR2GRAY);
```

```
            vector< Rect_<int> > faces;
            haar_cascade.detectMultiScale(gray, faces);

            for(int i = 0; i < faces.size(); i++) {
                Rect face_i = faces[i];
                Mat face = gray(face_i);
                Mat face_resized;
                cv::resize(face, face_resized, Size(im_width,
im_height), 1.0, 1.0, INTER_CUBIC);
                // Evaluar la predicción sobre rostro:
               int prediction = model->predict(face_resized);
            int ii, jj, mm;
        if(kk==0){
             // Atribuimos una accion para cada valor predicho
             if(prediction == 1){
                  usleep(1000);
                  ii = 0;
                  jj = 0;
                  cout << "Esperando modo..." << endl;
                  usleep(100000);
                  }

             if(prediction == 2){
                  ii = ii + 1;
                  usleep(1000);
                  if(ii == 15){
                       ii = 0;
                       jj = 0;
                       cout << "Modo 1 detectado" << endl;
                       cout << "Dando de tomar agua... " << endl;
                       usleep(10000000);
                       kk = 1; ll = 0;
                  }
             }

             if(prediction == 3){
                  jj = jj + 1;
                  usleep(1000);
                  if(jj == 15){
                       jj = 0;
                       ii = 0;
                       cout << "Modo 2 detectado" << endl;

                  }
             }
        }

        if(kk==1 && ll==1){
             if(prediction == 1)
             {
             cout << "Neutro" << endl;
             usleep(1000);
             mm = mm + 1;
             usleep(1000);
             if(mm == 50){
                  mm = 0;
                  cout << "Selector de modo" << endl;
                  kk = 0; ll = 0;
                  goto selector;
             }
             }
```

```
            if(prediction == 2)
            {
            cout << "Abajo" << endl;
            usleep(1000);
            }
            if(prediction == 3)
            {
            cout << "Arriba" << endl;
            usleep(1000);
            }
            if(prediction == 4)
            {
            cout << "Izquierda" << endl;
            usleep(1000);
            }
            if(prediction == 5)
            {
            cout << "Derecha" << endl;
            usleep(1000);
            }
     }

               rectangle(original, face_i, CV_RGB(0, 255,0), 1);
               string box_text = format("Prediction = %d", prediction);
               int pos_x = std::max(face_i.tl().x - 10, 0);
               int pos_y = std::max(face_i.tl().y - 10, 0);
               putText(original, box_text, Point(pos_x, pos_y),
FONT_HERSHEY_PLAIN, 1.0, CV_RGB(0,255,0), 2.0);
            }
            // Mostrar resultado:
            imshow("Reconocimiento de rostro", original);
            char key = (char) waitKey(20);
            // Salir del bucle:
            if((key == 27) || ((kk==1) && (ll==0)))
            break;
    }

cout << "Proceso terminado" << endl;
char key = (char) waitKey(20);
if((key == 27))
break;
}
    return 0;
}
```

Anexo 3:

```
// Programa para la toma de imágenes para la implementación de
la base de datos

#include "opencv2/core/core.hpp"
#include "opencv2/contrib/contrib.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/objdetect/objdetect.hpp"

#include <iostream>
#include <fstream>
#include <sstream>

using namespace cv;
using namespace std;

static void read_csv(const string& filename, vector<Mat>& images,
vector<int>& labels, char separator = ';') {
    std::ifstream file(filename.c_str(), ifstream::in);
    if (!file) {
        string error_message = "No valid input file was given,
please check the given filename.";
        CV_Error(CV_StsBadArg, error_message);
    }
    string line, path, classlabel;
    while (getline(file, line)) {
        stringstream liness(line);
        getline(liness, path, separator);
        getline(liness, classlabel);
        if(!path.empty() && !classlabel.empty()) {
            images.push_back(imread(path, 0));
            labels.push_back(atoi(classlabel.c_str()));
        }
    }
}

int main(int argc, const char *argv[]) {

    if (argc != 4) {
        cout << "usage: " << argv[0] << " </path/to/haar_cascade>
</path/to/csv.ext> </path/to/device id>" << endl;
        cout << "\t </path/to/haar_cascade> -- Path to the Haar
Cascade for face detection." << endl;
        cout << "\t </path/to/csv.ext> -- Path to the CSV file with
the face database." << endl;
        cout << "\t <device id> -- The webcam device id to grab
frames from." << endl;
        exit(1);
    }

    string fn_haar = string(argv[1]);
    string fn_csv = string(argv[2]);
    int deviceId = atoi(argv[3]);

    vector<Mat> images;
    vector<int> labels;

    try {
        read_csv(fn_csv, images, labels);
```

```
    } catch (cv::Exception& e) {
        cerr << "Error opening file \"" << fn_csv << "\". Reason: "
<< e.msg << endl;

        exit(1);
    }

    int im_width = images[0].cols;
    int im_height = images[0].rows;

    Ptr<FaceRecognizer> model = createFisherFaceRecognizer();
    model->train(images, labels);

    CascadeClassifier haar_cascade;
    haar_cascade.load(fn_haar);

    VideoCapture cap(deviceId);

    if(!cap.isOpened()) {
        cerr << "Capture Device ID " << deviceId << "cannot be
opened." << endl;
        return -1;
    }



Mat frame;
    for(;;) {
        cap >> frame;

        Mat original = frame.clone();

        Mat gray;
        cvtColor(original, gray, CV_BGR2GRAY);

        vector< Rect_<int> > faces;
        haar_cascade.detectMultiScale(gray, faces);

        for(int i = 0; i < faces.size(); i++) {

            Rect face_i = faces[i];

            Mat face = gray(face_i);

            Mat face_resized;
            cv::resize(face, face_resized, Size(im_width,
im_height), 1.0, 1.0, INTER_CUBIC);

            int prediction = model->predict(face_resized);

         cv::resize(face, original, Size(im_width, im_height), 1.0,
1.0, INTER_CUBIC);

            string box_text = format("Prediction = %d", prediction);
         PLAIN, 1.0, CV_RGB(0,255,0), 2.0);
        }

        imshow("face_recognizer", original);

        char key = (char) waitKey(20);
```

```
            if(key == 27)
                break;
        }
        return 0;
    }
```