UNIVERSIDAD DE COSTA RICA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉCTRICA
ESTRUCTURAS ABSTRACTAS DE DATOS Y ALGORITMOS PARA INGENIERÍA
IE-0217
II CICLO 2014
PROYECTO FINAL

Detección de objetos en 2D mediante la cámara superior del NAO

Estudiantes/Carné:
Boris Altamirano Chinchilla - B30255
Carlos Eduardo Solano Abarca - B36685
Heberth Valverde Gardela - B37174

1. Abstract

El presente trabajo se basó en la creación de una librería en C++ para la detección de objetos con representación bidimensional con las cámaras del robot NAO. El programa indica al NAO que gire su cabeza en busca de un objeto que se indica previamente, mediante los algoritmos SIFT y SURF se compara las imágenes captadas por la cámara del NAO y cuando lo encuentra mueve su cabeza con el fin de centrar el objeto en la escena vista por el NAO, y así posteriormente se dirige hacia el mismo.

2. Introducción

NAO es un robot humanoide que ha ganado mucha popularidad en los últimos años debido a sus aplicaciones educativas, el NAO H25 es el modelo más nuevo de esta gama de robots, cuenta con 25 grados de libertad, puede mover su cabeza $68°$ en posición vertical y $239°$ en posición vertical, lo que le permite ver mediante sus dos cámaras $61°$ horizontalmente y $47°$ verticalmente, además posee un sensor de inercia y cuatro micrófonos direccionales, entre otras funciones, que les permiten detectar lo que hay a su alrededor(Aldebaran Community, 2013).

Las dos cámaras del NAO pueden capturar hasta 30 imágenes por segundo, estas son procesadas por el software, y mediante la implementación de una serie de algoritmos el robot puede reconocer quién le habla o una bola roja pequeña, también mediante el programa Choregraphe se pueden guardar nuevos objetos en la memoria del NAO para que pueda reconocerlos posteriormente. Es posible crear módulos propios que interactúan con OpenCV para permitir al NAO reconocer una gama más amplia de objetos.

Como ya se mencionó, el sistema operativo de los NAO cuenta con un módulo precargado que permite el reconocimiento

de objetos circulares rojos, este módulo se vale del procesamiento de video para obtener las imágenes captadas por las cámaras del NAO, luego filtra la imagen por el color de los pixeles y finalmente analiza la forma del conjunto de pixeles filtrados. Una vez que se detectada la imagen se necesario guardar varios parámetros relacionados con ella, como el tiempo en el cual se detectó, las coordenadas y el tamaño del objeto y la posición de la cámara, además de cuál de las dos fue la que detectó el objeto (Solano, 2014).

El reconocimiento facial es más complejo que la detección de objetos, ya que no consiste únicamente en detectar las características básicas de un rostro (ojos, nariz, boca, etcétera) y ubicarlas en el espacio, sino que también se debe comparar con una base de datos para identificar el rostro, por lo que se requiere mucha precisión para distinguir una característica de otra. También existen una serie de parámetros que se guardan en la memoria al momento de detectar un rostro.

3. Objetivos

3.1. Objetivo General

Extender mediante una librería de C++ las funciones existentes de los NAOs relacionadas con los sensores visuales, como lo son las cámaras integradas.

3.2. Objetivos Específicos

Los objetivos específicos son:

1. Estudiar los API de reconocimiento de objetos y otros que incluyan funciones relacionadas con las cámaras de los NAOs.
2. Extender el reconocimiento de objetos de los robots NAOs utilizando las herramientas existentes.
3. Ejemplificar los logros alcanzados con un programa básico.

4. Justificación

En un robot humanoide existen diversas características importantes a tener en cuenta para un funcionamiento útil y eficaz del mismo. Una de las principales características es la capacidad de reconocimiento de objetos y personas mediante las cámaras integradas del mismo. Esta característica es muy importante, ya que brinda una mayor autonomía y utilidad al robot.

Los robots NAO que se encuentran actualmente en el PRIS-Lab tienen una capacidad muy limitada en cuanto a esta característica, ya que en el caso de reconocimiento de objetos, solo cuentan con un programa para reconocer una bola de color rojo, por este motivo es de gran importancia expandir ese reconocimiento a más objetos y optimizar esta característica a fin de poder utilizar al robot para tareas complejas. Como por ejemplo recojer objetos del suelo, reconocerlos como basura y arrojarlos al basurero.

Al construir sobre el NAO, una plataforma de mayor nivel para los sensores se garantiza que la persona que lo supervisa

podrá invertir menos esfuerzo y tiempo para desempeñar con el NAO las tares que desee.

5. Algoritmos de detección de características visuales

Según Arancil (2012) los algoritmos SIFT y SURF permiten caracterizar la información que se obtiene de una imagen para que permanezca invariante a cambios en la escala, rotación y alteraciones parciales de iluminación. Se procederá a explicar ambos algoritmos así como una comparación entre ellos.

5.1. SIFT (Scale Invariant Features Transform)

El autor anterior menciona que el algoritmo SIFT fue desarrollado en 1999 por David Lowe y consiste en una transformación a las características de la imagen para que pueda hallarse una correspondencia entre ella y la imagen buscada, para obtener las características de la imagen se deben llevar a cabo cuatro etapas:

5.1.1. Detección de máximos y mínimos de espacio-escala

En esta etapa del algoritmo se aplica la función escala-espacio $L(x, y, \sigma)$ para la búsqueda de los keypoints (puntos características) en todas las locaciones de las diferentes escalas, esta función se define como la convolución de la Gaussiana $G(x, y, \sigma)$ con la imagen $I(x, y)$. Posteriormente se utiliza la diferencia de Gaussianas convolucionada con la imagen $D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y)$ para eliminar los keypoints inestables y así hacer más rápida la búsqueda (Romero & Cazorla, 2009; Plaza & Zambrano, 2012).

5.1.2. Localización de los keypoints

Para detectar los máximos y mínimos locales se compara cada punto obtenido con sus ocho vecinos de la imagen actual y con sus nueve vecinos de la imagen superior e inferior, de estos puntos se debe guardar su escala, y la fila y columna donde se encuentran, ya que esto permite posteriormente identificar los keypoints para realizar la comparación (Romero & Cazorla, 2009; Plaza & Zambrano, 2012).

5.1.3. Asignación de orientación

Para calcular la orientación del keypoint se calcula la magnitud del gradiente como

$$m(x, y) = \sqrt{(L(x + 1, y) - L(x - 1, y))^2 + (L(x, y + 1) - L(x, y - 1))^2}$$

y su orientación mediante $\Theta = \arctan \dfrac{L(x, y + 1) - L(x, y - 1)}{L(x + 1, y) - L(x - 1, y)}$ (Romero & Cazorla, 2009). Para cada keypoitn se debe crear un histograma de 36 orientaciones donde se almacenan los valores ponderados de la magnitud del gradiente al pasar por la ventana circular gaussiana $\sigma = 1{,}5\times$ escala (Plaza & Zambrano, 2012).

5.1.4. Descripción del punto de interés

Para describir cada keypoint se calculan los gradientes de los vecinos del punto de interés y después se rotan las coordenadas del descriptor de forma relativa a la orientación del keypoint de la tercera etapa para que permanezca invariante (Romero & Cazorla, 2009). Al cumplir estas etapas se obtienen descriptores altamente distintivos que permiten realizar una correspondencia de características entre dos imágenes (Plaza & Zambrano, 2012).

5.2. SURF (Speeded Up Robust Features)

Este algoritmo utiliza muchas de las funciones de SIFT pero presenta algunas mejoras, como lo es un considerable tiempo de computación menor sin perder precisión y mayor robustez ante las variaciones en la imagen (Arancil, 2012). Fue desarrollado por Herbet Bay en el 2006, y es más rápido que el SIFT porque los keypoints poseen menos descriptores, ya que la mayoría poseen un valor de cero (Plaza & Zambrano, 2012).

El algoritmo SURF sigue en esencia las mismas cuatro etapas del SIFT antes mencionadas, pero para mejorar su velocidad de computación utiliza la matriz Hessiana $H(p,\sigma)$ en la primera etapa, donde $p = (x,y)$ comformada por las derivadas de segundo orden de L, para calcular el determinante de esta matriz se emplean aproximaciones de estas derivadas, así: $det(H) \approx D_{xx}D_{yy} - 0{,}9(D_{xy})^2$. Otra rasgo de los puntos característicos obtenidos mediante el SURF es su repetitibilidad, ya que si el keypoint resulta estable y confiable entonces será obtenido sin problema al variar la escala y la rotación, por lo que las alteraciones fotométricas y otros tipos de ruidos no suelen afectar (Romero & Cazorla, 2009).

Es importante mencionar que estos algoritmos funcionan mejor con imágenes en escala de grises, debido a esto a las imágenes a comparar se les aplicó este filtro y además en el código se indicó que el video capturado por el NAO se transmitiera en escala de grises.

6. Desarrollo del Proyecto

El primer punto para llevar a cabo el proyecto fue la implementación de los algoritmos descritos en la sección anterior, inicialmente la estuctura a seguir para que el programa compilara y corriera correctamente fue un obstáculo, pero se tomó un ejemplo proporcionado por aldebaran como esqueleto para el desarrollo del código. Se implementó el código del ejemplo getimage para obtener acceso a la cámara superior del NAO y así llevar a cabo la comparación con la imagen guardada en la base de datos. Mediante los algoritmos de detección de características visuales se identificó las esquinas del objeto a buscar en las imágenes vistas por el NAO, esto permitió encontrar el centro del objeto. Seguidamente se calculó el desplazamiento del centro del objeto respecto al centro del imagen y mediante el movimiento de la cabeza, girando entorno al eje Z y al eje Y, se logró que ambos centros coincidieran. Para saber cuantos radianes debía girar la cabeza el NAO según la separación se creó una escala a base de prueba y error, donde se obtuvo que al girar la cabeza en torno al eje Z cerca de 0.42 rad el centro de la imagen se desplazaba la mitad del ancho de la imagen (en calidad QVGA).

7. Metodología

Para lo anterior fue necesario utilizar las herramientas existentes para facilitar el desarrollo del proyecto, por ejemplo, se utilizaron los API de reconocimiento de objetos y otras herramientas y algoritmos existentes. Una vez logrado el objetivo anterior, será posible trabajar con los objetos reconocidos en distintas maneras.

Todo esto se sintetizó en una librería de C++ donde se incluyeron todas las funciones anteriores y otras funciones imprevistas que se implementaron durante el desarrollo del proyecto. Finalmente, se deseó ejemplificar con un programa sencillo todo lo anterior, por ejemplo, que el NAO reconozca un objeto con bastantes características y que haga alguna acción simple después de reconocer el objeto. El código creado se puede recrear en el simulador de ALdebaran o se puede modificar para que corra localmente en alguno de los robots NAO que este a disposición por el PRIS-Lab.

Antes de comenzar a programar, se planificó la estructura de la librería para que el trabajo fuera más eficaz, sin embargo esta estructura sufrió mdificaciones durante el desarrollo del proyecto. Una vez hecho esto, se estudió el funcionamiento de los API de reconocimiento de objetos así como algunos algoritmos que ayuden a cumplir con el desarrollo del proyecto y con los objetivos. Por ejemplo, se analizaron los ejemplos y se investigó en la teoría que proporciona el libro An introduction to robotics with NAO de Aldebaran Robotics así como la documentación disponible en línea, para comprender mejor los problemas enfrentados. También, se consultaron diferentes referencias como tesis, artículos de revistas y páginas web que contenían información escencial. Finalmente, se creó un video donde se expuso el desarrollo del trabajo, así como los resultados obtenidos y sus implicaciones, así como una muestra del programa en acción.

8. Conclusiones

En este proyecto se logró expandir las funciones de detección de objetos del robot NAO, haciendo posible el reconocimiento casi de cualquier elemento con solo ser agregado previamente a una base de datos, con excepción de algunos que sean tan simples que no tengan casi irregularidades o características para realizar la comparación.

En cuanto a las limitaciones que se presentaron, una es que al ejecutar el programa de forma remota, la transmisión de datos es muy lenta ya que es limitada por la velocidad del Wi-Fi, se disminuyó la calidad de la imagen lo posible, pero aun así la velocidad de transmisión fue muy lenta y por lo tanto su ejecución no es tan eficiente. Esto se podría evitar mediante una compilación cruzada utilizando la herramienta Nao Atom Cross Tool Chain proporcionada por el fabricante, y de esta forma ejecutar el programa internamente en el sistema operativo del robot, consiguiendo un mejor desempeño.

9. Anexos

9.1. main.cpp

```
/**
 * #NoobTeam
 */

///Include libraries to comunicate using the Broker.
///To connect to a desire IP, run ./naocam --pip <adress> from ./toolchain/sdk/bin
#include <signal.h>
#include <boost/shared_ptr.hpp>
#include <alcommon/albroker.h>
#include <alcommon/almodule.h>
#include <alcommon/albrokermanager.h>
#include <alcommon/altoolsmain.h>

///Include the header file
#include "naocam.h"

///If the event is remote...
#ifdef NAOCAM_IS_REMOTE
# define ALCALL
#else
# ifdef _WIN32
#  define ALCALL __declspec(dllexport)
# else
#  define ALCALL
#  endif
#endif

extern "C"
{

///Create and run the module.
  ALCALL int _createModule(boost::shared_ptr<AL::ALBroker> pBroker)
  {
    // init broker with the main broker instance
    // from the parent executable
    AL::ALBrokerManager::setInstance(pBroker->fBrokerManager.lock());
    AL::ALBrokerManager::getInstance()->addBroker(pBroker);
    AL::ALModule::createModule<NaoCam>( pBroker, "NaoCam" );

    return 0;
  }

///Close the module.
  ALCALL int _closeModule()
  {
    return 0;
  }
}

#ifdef NAOCAM_IS_REMOTE
  int main(int argc, char *argv[])
  {
    /// Pointer to createModule
    TMainType sig;
```

```
    sig = &_createModule;
    /// Call main function.
    ALTools::mainFunction("naocam", argc, argv, sig);
  }
#endif
```

9.2. naocam.h

```
/**
 * #NoobTeam
 */

///Define the class name.
#ifndef NAOCAM_NAOCAM_H
#define NAOCAM_NAOCAM_H

#include <boost/shared_ptr.hpp>
#include <alcommon/almodule.h>
#include <alvision/alimage.h>
#include <string>

///Include Aldebaran libraries to create a proxy between Nao and PC.
#include <alproxies/alvideodeviceproxy.h>
#include <alproxies/altexttospeechproxy.h>
#include <alproxies/almotionproxy.h>
#include <alproxies/alledsproxy.h>
#include <alproxies/alrobotpostureproxy.h>


///Include opencv libraries for image processing.
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/core/core.hpp"
#include "opencv2/calib3d/calib3d.hpp"

///Define namespace cv and std.
namespace cv{}
using namespace std;
using namespace cv;
namespace AL
{
  class ALBroker;
}

class NaoCam : public AL::ALModule
{
///Define public methods.
///init() should be always be present.
  public:
    NaoCam(boost::shared_ptr<AL::ALBroker> broker, const string& name);
    virtual ~NaoCam();
```

```
    void init();
    void trackingObject();
    void moveHead(float move, string joint);
    void usingLEDS(float duration);
    void move_to(float theta);

///Define private variables.
  private:
    AL::ALVideoDeviceProxy fVideoProxy;
    std::string fGVMId;
    AL::ALMotionProxy mProxy;
    AL::ALTextToSpeechProxy aProxy;
    AL::ALLedsProxy leds;
    AL::ALRobotPostureProxy posture;
    AL::ALImage* fImagePointer;
};

#endif  // NAOCAM_NAOCAM_H
```

9.3. naocam.cpp

```
/**
 * #Noobs
 */

///Include the header file
#include "naocam.h"

///Include Aldebaran libraries to create a proxy between Nao and PC.
#include <alvalue/alvalue.h>
#include <alcommon/alproxy.h>
#include <alcommon/albroker.h>

///Include opencv libraries for image processing.
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/core/core.hpp"
#include "opencv2/calib3d/calib3d.hpp"

///Define namespace cv and std.
namespace cv{}
using namespace std;
using namespace cv;

///Include Aldebaran libraries to suscribe the desire target.
#include <alvision/alvisiondefinitions.h>
#include <alproxies/almotionproxy.h>
#include <alproxies/altexttospeechproxy.h>
#include <alproxies/alledsproxy.h>
#include <alproxies/alrobotpostureproxy.h>
```

```
///The constructor initialize the broker and the objects.
NaoCam::NaoCam( boost::shared_ptr<AL::ALBroker> broker, const string& name):
        AL::ALModule(broker, name),
    fVideoProxy(AL::ALVideoDeviceProxy(broker)),
    mProxy(AL::ALMotionProxy(broker)),
    aProxy(AL::ALTextToSpeechProxy(broker)),
    leds(AL::ALLedsProxy(broker)),
    posture(AL::ALRobotPostureProxy(broker)),
    fGVMId("GVM")
{
  setModuleDescription("This module identify and tracks an 2D object using SURF and SIFT algorithms.");
}

///The destructor unsubscribes the proxies.
NaoCam::~NaoCam()
{
        fVideoProxy.unsubscribe(fGVMId);
        mProxy.rest();
        delete fImagePointer;
}


void NaoCam::init()
{
///Run the desire methods
        try
        {
/*
Parameters of .suscribe:
        *Name     Name of the subscribing module.
        *resolution     Resolution requested.
                **AL::kQQVGA image of 160*120px.
                **AL::kQVGA image of 320*240px.
                **AL::kVGA image of 640*480px
                **AL::k4VGA image of 1280*960px.
        *colorSpace     Colorspace requested.
        *fps     Fps (frames per second) requested to the video source. The OV7670 VGA camera can only
*/
        fGVMId = fVideoProxy.subscribe(fGVMId, AL::kQVGA, AL::kRGBColorSpace, 30);
        usingLEDS(3.0f);
        posture.goToPosture("StandInit", 0.5f);
        trackingObject();
        }

///If the fVideoProxy can't suscribe, then exit with status 1.
        catch (const AL::ALError& e)
        {
    cerr << "Caught exception: \n" << e.what() << endl;
        ::exit(1);
        }
}

///Using LEDs.
```

```
void NaoCam::usingLEDS(float duration)
{
        leds.rasta(duration);
}

///Method for moving NAO's head, receives an angle (float) and the direction of movement (string).
///The string to receive can be either "HeadYaw" for X movement or "HeadPitch" for Y movement.
void NaoCam::moveHead(float move, string joint)
{

///The name of the joint to be moved.
  const AL::ALValue jointName = joint;

///Make sure the head is stiff to be able to move it. To do so, make the stiffness go to the maximum in
///Target stiffness.
        AL::ALValue stiffness = 1.0f;
///Time (in seconds) to reach the target.
    AL::ALValue time = 1.0f;
///Call the stiffness interpolation method.
    mProxy.stiffnessInterpolation(jointName, stiffness, time);

///Set the target angle list, in radians.
    AL::ALValue targetAngles = AL::ALValue::array(move);
///Set the corresponding time lists, in seconds.
    AL::ALValue targetTimes = AL::ALValue::array(1.5f);
///Specify that the desired angles are absolute.
    bool isAbsolute = false;

///Call the angle interpolation method. The joint will reach the desired angles at the desired times.
    mProxy.angleInterpolation(jointName, targetAngles, targetTimes, isAbsolute);

///Remove the stiffness on the head.
    stiffness = 0.0f;
    time = 1.0f;
    mProxy.stiffnessInterpolation(jointName, stiffness, time);

}

void NaoCam::trackingObject()
{
///Load image of the object to be recognized and load to grayscale.
        const Mat img_object = imread("img.jpg", CV_LOAD_IMAGE_GRAYSCALE);
///Define an Mat objet with the dimensions of the NAO's camera requested resolution.
        Mat img_scene_color = Mat(Size(320, 240), CV_8UC3);
///Define Mat objet to contain scene image.
        Mat img_scene;
///Create a OpenCV window to display the images.
        namedWindow("images");

///Main loop. Exit when pressing ESC.*/
while ((char) waitKey(30) != 27){
/* Retrieve an image from the camera.
    * The image is returned in the form of a container object, with the
    * following fields:
```

```
    * 0 = width
    * 1 = height
    * 2 = number of layers
    * 3 = colors space index (see alvisiondefinitions.h)
    * 4 = time stamp (seconds)
    * 5 = time stamp (micro seconds)
    * 6 = image buffer (size of width * height * number of layers)
*/
        AL::ALValue img = fVideoProxy.getImageRemote(fGVMId);

/// Access the image buffer (6th field) and assign it to the opencv image container.
    img_scene_color.data = (uchar*) img[6].GetBinary();
/* Convert coloured scene image to grayscale scene image.
        This is performed to have better matches*/
    cvtColor(img_scene_color, img_scene, CV_BGR2GRAY);

///*Detect the keypoints using SURF Detector
/// Set minHessian for the SurfFeatureDetector.
        int minHessian = 400;
        SurfFeatureDetector detector( minHessian );
/// Set keypoint vectors for the image and the scene.
        vector<KeyPoint> keypoints_object, keypoints_scene;
        detector.detect( img_object, keypoints_object );
        detector.detect( img_scene, keypoints_scene );

///*Calculate descriptors (feature vectors).
        SurfDescriptorExtractor extractor;
///Set descriptors vectors for the image and the scene.
        Mat descriptors_object, descriptors_scene;
        extractor.compute( img_object, keypoints_object, descriptors_object );
        extractor.compute( img_scene, keypoints_scene, descriptors_scene );

///*Matching descriptor vectors using FLANN matcher.
        FlannBasedMatcher matcher;
        vector< DMatch > matches;
        matcher.match( descriptors_object, descriptors_scene, matches );

///Calculate max and min distances between keypoints
        double max_distance = 0;
        double min_distance = 100;

        for( int i = 0; i < descriptors_object.rows; i++ )
        { double distance = matches[i].distance;
        if( distance < min_distance ) min_distance = distance;
                if( distance > max_distance ) max_distance = distance;
        }

///Print max and min distances.
        cout<<"Max distance: "<<max_distance<<endl;
        cout<<"Max distance: "<<min_distance<<endl;

///Draw only matches whose distance is less than 3*min_dist.
        vector<DMatch> good_matches;
```

```
        for( int i = 0; i < descriptors_object.rows; i++ )
        {
                if( matches[i].distance < 3*min_distance )
        {
                        good_matches.push_back(matches[i]);
                }
        }

///Print how many good matches where found.
    cout<<"--Good Matches: "<<good_matches.size()<<endl;

        Mat img_matches;
///Draw a line between matches.
        drawMatches( img_object, keypoints_object, img_scene, keypoints_scene,
                                good_matches, img_matches, Scalar::all(-1), Scalar::all(-1),
                                vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS );

///Localize the object.
        vector<Point2f> object;
        vector<Point2f> scene;

        for(int i=0; i<good_matches.size(); i++)
        {
///Get the keypoints from the good matches
        object.push_back(keypoints_object[good_matches[i].queryIdx].pt );
        scene.push_back(keypoints_scene[good_matches[i].trainIdx].pt );
        }

        Mat H = findHomography(object, scene, CV_RANSAC);

///Get the corners from the object to be detected and the center of the object.
        vector<Point2f> obj_corners(5);
        obj_corners[0] = cvPoint(0,0);
        obj_corners[1] = cvPoint(img_object.cols, 0);
        obj_corners[2] = cvPoint(img_object.cols, img_object.rows);
        obj_corners[3] = cvPoint(0, img_object.rows);
///Get the center of the object
        obj_corners[4] = cvPoint(img_object.cols/2,img_object.rows/2 );
        vector<Point2f> scene_corners(5);

///Adapt the image from the object to the scene.
        perspectiveTransform( obj_corners, scene_corners, H);

///Draw lines between the corners (the mapped object in the scene) and draw a circle in the center of t
        line( img_matches, scene_corners[0] + Point2f( img_object.cols, 0), scene_corners[1] + Point2f(
        line( img_matches, scene_corners[1] + Point2f( img_object.cols, 0), scene_corners[2] + Point2f(
        line( img_matches, scene_corners[2] + Point2f( img_object.cols, 0), scene_corners[3] + Point2f(
        line( img_matches, scene_corners[3] + Point2f( img_object.cols, 0), scene_corners[0] + Point2f(
///Draw the center of the object.
        circle( img_matches, scene_corners[4] + Point2f( img_object.cols, 0), 3, Scalar( 255, 0, 0), -1
///Draw the center of the scene.
        circle( img_matches, Point2f(img_scene.cols/2, img_scene.rows/2) + Point2f( img_object.cols, 0)

///Print position of the center of the object.
```

```
        cout<<"--X position: "<<scene_corners[4].x<<endl;
        cout<<"--Y position: "<<scene_corners[4].y<<endl;

///Show detected matches
        imshow( "Object Detection", img_matches);

///Getting distance froms object to center.
        float movex = img_scene.cols/2-scene_corners[4].x;
        float movey = -img_scene.rows/2+scene_corners[4].y;

/*Moving head to object
        0.85f defines the transformation between distance between pixels and radians.
        Also works with 0.9f.
*/
        if(abs(movex)>1) {
                cout<<"--Moving in x: "<<movex<<endl;
                moveHead(movex*0.85f/img_scene.cols, "HeadYaw");
        }
        if(abs(movey)>1) {
                cout<<"--Moving in y: "<<movey<<endl;
                moveHead(movey*0.85f/img_scene.rows, "HeadPitch");
        }

/*If the distance between the center of the object and the center of the scene is less than
        30 pixels, then the object is detected, so say "I found the object"
*/

        if((abs(movex)<30)&&(abs(movey)<30))
        {
                usingLEDS(3.0f);
                aProxy.say("I found the object");
                cout<<"--Moving: "<<movex<<endl;
        }



/*Tells to ALVideoDevice that it can give back the image buffer to the
driver. Optional after a getImageRemote but MANDATORY after a getImageLocal.*/
        fVideoProxy.releaseImage(fGVMId);
  }
  destroyAllWindows();
///Do not forget to release the image.
}
```

10. Referencias

1. Aldebaran (2014). *NAO Robot: intelligent and friendly companion.*Recuperado de $http://www.aldebaran.com/en/humanoid-robot/naorobot$
2. Aldebaran (2014). *NAO Documentation.*Recuperado de $http://doc.aldebaran.com/1-14/$
3. Aracil, R. (2012). *Desarrollo de un sistema cognitivo de visión para la navegación robótica*. Tesis de licenciatura no publicada, Universidad Politècnica de València, Valencia, España.
4. Beiter, M., Coltin B., Liemhetcharat, S., *An Introduction To Robotics With NAO*, Aldebaran Robotics, 2012.

5. Plaza, A. & J. Zambrano. *Estudio y Selección de las Técnicas SIFT, SURF y ASIFT de Reconocimiento de Imágenes para el Diseño de un Prototipo en Dispositivos Móviles*. Tesis de licenciatura no publicada, Universidad Politéctica Salesiana, Cuenca, Ecuador.
6. Romero, A. & M. Cazorla. *Comparativa de detectores de características visuales y su aplicación al SLAM*. Actas del X Workshop de Agentes Físicos. Cáceres, España. pp. 55-62.
7. Solano, A. (2014). *Desarrollo de funciones de movimiento y control de los sensores para una plataforma robótica NAO*. Tesis de licenciatura no publicada, Universidad de Costa Rica, Ciudad Universitaria Rodrigo Facio", Costa Rica.