ANNEXES

1. Détection de NaoMarks [C++]

1.1.Main

```
#include <iostream>
#include <string>
#include <alerror/alerror.h>
#include <alproxies/almotionproxy.h>
#include <alproxies/allandmarkdetectionproxy.h>
#include <alproxies/almemoryproxy.h>
#include <alproxies/allandmarkdetectionproxy.h>
#include <alproxies/altexttospeechproxy.h>
#include <alproxies/alspeechrecognitionproxy.h>
#include <alvalue/alvalue.h>
#include <signal.h>
int main(int argc, char* argv[]) {
  if(argc != 2)
  {
    std::cerr << "Wrong number of arguments!" << std::endl;
    std::cerr << "Usage: naomark NAO_IP" << std::endl;
    exit(2);
  }
int PORT = 9559;
  AL::ALTextToSpeechProxy tts(argv[1], 9559);
  //void ALSpeechRecognitionProxy::setLanguage(const std::string&
french);
  tts.say("bonjour,montre moi une naomark");
//Create a proxy to ALLandMarkDetection
AL::ALLandMarkDetectionProxy markProxy(argv[1], 9559);
```

```
char buffer [50];
//Subscribe to the ALLandMarkDetection extractor
int period = 500;
markProxy.subscribe("Test_Mark", period, 0.0 );
  tts.say("souscription test mark");
AL::ALMemoryProxy memProxy(argv[1], 9559);
 tts.say("memoire");
//AL::ALLandMarkDetectionProxy data;
       try {
      AL::ALValue data=memProxy.getData("LandmarkDetected");
      //data=memProxy.getData("LandmarkDetected");
       tts.say("detection et impression");
       if (data.getSize()==0)
             {
              tts.say("naomark vue ok");
             }
               while(data.getSize()==0)
              {
                    markProxy.subscribe("Test_Mark", period, 0.0 );
                    AL::ALValue
data=memProxy.getData("LandmarkDetected");
                    tts.say("en attente");
                   // char buffer [50];
                    //sprintf (buffer,"%d naomark
detecte",data[1].getSize()-1);
              }
              tts.say("naomark vue");
       }
```

```
        catch (const AL::ALError& e) {
            std::cerr << "Caught exception: " << e.what() <<
std::endl;
            exit(1);
        }
  exit(0);
}
```

1.2.CMakeLists

```
cmake_minimum_required(VERSION 2.8)

project(proyect1)

find_package(qibuild)

# Create a executable named proyect1

# with the source file: main.cpp

qi_create_bin(proyect1 "main.cpp")

# Add a simple test:

enable_testing()

qi_create_test(test_proyect1 "test.cpp")

qi_use_lib (proyect1 ALCOMMON)
```

2. Prendre une image avec Nao [C++]

2.1.Sur la version 1.12

2.1.1. MAIN

```
// Aldebaran includes.
#include <alproxies/alvideodeviceproxy.h>
#include <alvision/alimage.h>
#include <alvision/alvisiondefinitions.h>
#include <alerror/alerror.h>
#include <alproxies/alvisiontoolboxproxy.h>
// Opencv includes.
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <iostream>
#include <string>
using namespace AL;
/**
* \brief Shows images retrieved from the robot.
* \param robotIp the IP adress of the robot
*/
void showPictures(const std::string& robotIp)
{
  /** Create a proxy to ALVideoDevice on the robot.*/
  ALVideoDeviceProxy camProxy(robotIp, 9559);
  /** Subscribe a client image requiring 320*240 and BGR colorspace.*/
  const std::string clientName = camProxy.subscribe("test", kVGA,
kBGRColorSpace, 30);
```

```
  /** Create an iplimage header to wrap into an opencv image.*/
  IplImage* imgHeader = cvCreateImageHeader(cvSize(640, 480), 8, 3);
  /** Create a OpenCV window to display the images. */
  cvNamedWindow("images");
  /** 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)
    */
  ALValue img = camProxy.getImageRemote(clientName);
      /** Access the image buffer (6th field) and assign it to the
opencv image
    * container. */
  imgHeader->imageData = (char*)img[6].GetBinary();
      /** Display the iplImage on screen.*/
  cvShowImage("images", imgHeader);
  /** Main loop. Exit when pressing ESC.*/
  bool loop=true;
  while (loop==true)
  {
        int select =(char) cvWaitKey(1000);
         switch(select)
```

```
                {
                case 27:
                      loop=false;
                break;
                case 32:
                          /** Tells to ALVideoDevice that it can
give back the image buffer to the
                            * driver. Optional after a getImageRemote
but MANDATORY after a getImageLocal.*/
                      camProxy.releaseImage(clientName);
                      img = camProxy.getImageRemote(clientName);
                      imgHeader->imageData =
(char*)img[6].GetBinary();
                      cvShowImage("images", imgHeader);
                      printf("done \n");
                break;
                default:
                      ;
                };
  }
  /** Cleanup.*/
  camProxy.releaseImage(clientName);
  camProxy.unsubscribe(clientName);
  cvReleaseImageHeader(&imgHeader);
}
int main(int argc, char* argv[])
{
  if (argc < 2)
  {
```

```
    std::cerr << "Usage 'getimages robotIp'" << std::endl;
    return 1;
  }
  const std::string robotIp(argv[1]);
  try
  {
    showPictures(robotIp);
  }
  catch (const AL::ALError& e)
  {
    std::cerr << "Caught exception " << e.what() << std::endl;
  }  return 0;}
```

2.1.2. CMakeLists

```
cmake_minimum_required(VERSION 2.8)
project(sacarfoto)
find_package(qibuild)
# Create a executable named sacarfoto
# with the source file: main.cpp
qi_create_bin(sacarfoto "main.cpp")
# Add a simple test:
enable_testing()
qi_create_test(test_sacarfoto "test.cpp")
qi_use_lib (sacarfoto ALCOMMON)
qi_use_lib(sacarfoto ALCOMMON ALVISION OPENCV)
qi_use_lib(sacarfoto ALCOMMON ALVISION OPENCV2_CORE OPENCV2_HIGHGUI)
#qi_create_bin(sacarfoto sacarfoto.cpp)
```

2.2.Sur la version 1.14

2.2.1. MAIN

```
// Aldebaran includes.
#include <alproxies/alvideodeviceproxy.h>
#include <alvision/alimage.h>
#include <alvision/alvisiondefinitions.h>
#include <alerror/alerror.h>
#include <alproxies/alvisiontoolboxproxy.h>
// Opencv includes.
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <string>
using namespace AL;
/**
* \brief Shows images retrieved from the robot.
*
* \param robotIp the IP adress of the robot
*/
void showPictures(const std::string& robotIp)
{
  /** Create a proxy to ALVideoDevice on the robot.*/
  ALVideoDeviceProxy camProxy(robotIp, 9559);
  /** Subscribe a client image requiring 320*240 and BGR colorspace.*/
  const std::string clientName = camProxy.subscribe("test", kVGA,
kBGRColorSpace, 30);
  /** Create an cv::Mat header to wrap into an opencv image.*/
  cv::Mat imgHeader = cv::Mat(cv::Size(320, 240), CV_8UC3);
```

```
  /** Create a OpenCV window to display the images. */
  cv::namedWindow("images");
  /** 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)
    */
  ALValue img = camProxy.getImageRemote(clientName);
      /** Access the image buffer (6th field) and assign it to the
opencv image
    * container. */
    imgHeader.data = (uchar*) img[6].GetBinary();
    /** Tells to ALVideoDevice that it can give back the image buffer
to the
    * driver. Optional after a getImageRemote but MANDATORY after a
getImageLocal.*/
    camProxy.releaseImage(clientName);
    /** Display the iplImage on screen.*/
    cv::imshow("images", imgHeader);
   /** Main loop. Exit when pressing ESC.*/
 // printf("\n aqui \n");
  bool loop=true;
  while (loop==true)
```

```
  {
        int select =(char) cv::waitKey(1000);
         switch(select)
                  {
                  case 27:
                        loop=false;
                  break;
                  case 32:
                              /** Tells to ALVideoDevice that it can
give back the image buffer to the
                                * driver. Optional after a getImageRemote
but MANDATORY after a getImageLocal.*/
                        //cvReleaseImageHeader(&imgHeader);
                        //printf("\n bucle \n");
                        camProxy.releaseImage(clientName);
                        img = camProxy.getImageRemote(clientName);
                        imgHeader.data = (uchar*) img[6].GetBinary();
                        camProxy.releaseImage(clientName);
                        cv::imshow("images", imgHeader);
                        //cvSaveImage("imgnao.png", "images");
                        printf("done \n");
                  break;
                  default:
                        ;
                  };
  }
  /** Cleanup.*/
  camProxy.releaseImage(clientName);
  camProxy.unsubscribe(clientName);
```

```
  //cvReleaseImageHeader(&imgHeader);
}
int main(int argc, char* argv[])
{
  if (argc < 2)
  {
    std::cerr << "Usage 'getimages robotIp'" << std::endl;
    return 1;
  }
  const std::string robotIp(argv[1]);
  try
  {
    showPictures(robotIp);
  }
  catch (const AL::ALError& e)
  {
    std::cerr << "Caught exception " << e.what() << std::endl;
  }
  return 0;
}
```

2.2.2. CMakeLists

```
cmake_minimum_required(VERSION 2.8)

project(newfoto)

find_package(qibuild)

# Create a executable named newfoto

# with the source file: main.cpp

qi_create_bin(newfoto "main.cpp")


# Add a simple test:

enable_testing()

qi_create_test(test_newfoto "test.cpp")

qi_use_lib (newfoto ALCOMMON)

#qi_use_lib(newfoto ALCOMMON ALVISION OPENCV)

qi_use_lib(newfoto ALCOMMON ALVISION OPENCV2_CORE OPENCV2_HIGHGUI)
```

3. qrdecoder (Code déjà existant) [Python]

3.1.MAIN

```
#!/usr/bin/env python

# Import Stuff

import sys, os, zbar, Image, sip, string

import PcxImagePlugin, TgaImagePlugin, TiffImagePlugin

sip.setapi("QVariant", 2)

from PyQt4 import QtGui, QtCore

from PyQt4.QtCore import *

from PyQt4.QtGui import *

from src.ui.ui_mainwindow import Ui_MainWindow

from src.ui.ui_aboutwindow import Ui_aboutDialog

import src.ui.images_rc


# App Name

__appName__ = "qrDecoder"

__verFileName__ = 'VERSION'

__configFileName__ = 'settings'


# Determine if application is a script file or frozen exe and get
appPath

# http://stackoverflow.com/a/404750/1061279

if hasattr(sys, 'frozen'):

    __appPath__ = os.path.dirname(sys.executable)

elif __file__:

    __appPath__ = os.path.dirname(__file__)

__verPath__ = os.path.join(__appPath__, __verFileName__)
```

```
# Get qrdecoder version from VERSION file

f = open(__verPath__, 'r')

__version__ = f.readline()

f.close()

#=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=- ABOUT WINDOW =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

# About window properties

class AboutDialog(QDialog, Ui_aboutDialog):

    def __init__(self, parent=None):

        super(QDialog, self).__init__(parent)

        self.setupUi(self)

        self.setWindowTitle("About")

        # Info displayed in the About window

        __info__ = ("<b>"+__appName__+" v" + __version__ + "</b>"

            "<p>Author: Nicholas Wilde"

            "<p>E-mail: <a
href='mailto:ncwilde43@gmail.com'>ncwilde43@gmail.com</a>"

            "<p>Home page: <a
href='http://code.google.com/p/qrdecoder/'>http://qrdecoder.googlecode
.com</a>"

            "<p>License: <a href='http://www.gnu.org/licenses/gpl-
3.0.txt'>GNU GPLv3</a>"

            )

        self.label.setText(__info__)

        self.connect(QtGui.QShortcut(QKeySequence("Ctrl+W"),self),

            SIGNAL("activated()"), SLOT("accept()"))

        self.connect(self.pushButton_OK, SIGNAL("clicked()"),
SLOT("accept()"))

        del __info__

#=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=- MAIN WINDOW =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

```
# Main Window properties

class qrDecoder(QMainWindow, Ui_MainWindow):

    def __init__(self):

        super(QMainWindow, self).__init__()

        self.setupUi(self)

        # Read settings from settings.ini file

        self.settingsFile()

        self.readSettings()

        # Have window accept drops

        self.setAcceptDrops(True)

        # Fix window size

        self.layout().setSizeConstraint(QtGui.QLayout.SetFixedSize)

        # Set window title

        self.setWindowTitle(__appName__+" - v"+__version__)

        # Set qlineedit to read only so user can copy but not edit.

        self.lineEdit_code.setReadOnly(1)

        # Disable copy push button

        self.pushButton_copy.setEnabled(0)

        # Button commands

        self.connect(self.pushButton_browse, SIGNAL("clicked()"),
self.selectImage)

        self.connect(self.pushButton_about, SIGNAL("clicked()"),
self.about)

        self.connect(self.pushButton_decode, SIGNAL("clicked()"),
self.decodeImage)

        self.connect(self.pushButton_clear, SIGNAL("clicked()"),
self.clearFields)

        self.connect(self.pushButton_exit, SIGNAL("clicked()"),
self.exitApplication)

        self.connect(self.pushButton_copy, SIGNAL("clicked()"),
self.copy2Clipboard)
```

```
        # Keyboard Shortcuts
        self.connect(QtGui.QShortcut(QKeySequence("Ctrl+W"),self),
            SIGNAL("activated()"), self.exitApplication)
    # Call the About window
    def about(self):
        dialog = AboutDialog(self)
        dialog.exec_()
    # Bring up the select file dialog once the browse button has been
clicked
    def selectImage(self):
        fileName = QFileDialog.getOpenFileName(self, "Select Image",
"",
            "Image Files (*.png *.jpg *.jpeg *.bmp *.gif *.pcx *.tga
*.tif); \
                ;All Files (*.*)")
        if fileName:
            self.clearFields()
            self.lineEdit_location.setText(fileName)
            self.decodeImage()
    # Use zbar to decode image
    def decodeImage(self):
        # Taken from zbar\examples\scan_image.py
        # create a reader
        scanner = zbar.ImageScanner()
        # configure the reader
        scanner.parse_config('enable')
        # obtain image data
        pil =
Image.open(unicode(self.lineEdit_location.text())).convert('L')
        width, height = pil.size
```

```
        raw = pil.tostring()
        # wrap image data
        image = zbar.Image(width, height, 'Y800', raw)
        # scan the image for barcodes
        scanner.scan(image)
        # extract results
        for symbol in image:
            # do something useful with results
            if symbol.type is not symbol.QRCODE:
                self.statusBar().showMessage("Image does not contain a
QR code.",
                    2000)
                return
            self.lineEdit_code.clear()
            self.lineEdit_code.setText(symbol.data)
            if not self.pushButton_copy.isEnabled():
                self.pushButton_copy.setEnabled(1)
            self.statusBar().showMessage("QR code decoded.", 2000)
        # clean up
        del(image)
    # Clear all fields
    def clearFields(self):
        self.lineEdit_location.clear()
        self.lineEdit_code.clear()
        if self.pushButton_copy.isEnabled():
            self.pushButton_copy.setEnabled(0)
    # Determine if drag even should happen
    def dragEnterEvent(self, event):
        if event.mimeData().hasUrls():
```

```
            event.accept()
        else:
            event.ignore()
    # What to do when the drag event happens
    def dropEvent(self, event):
        data = event.mimeData()
        for url in data.urls():
            fileName = url.toLocalFile()
            fileInfo = QFileInfo(fileName)
            if fileInfo.isFile():
                self.lineEdit_location.clear()
                self.lineEdit_code.clear()
                self.lineEdit_location.setText(fileName)
                self.decodeImage()
            else:
                event.ignore()
    # Copy decoded text to clipboard
    def copy2Clipboard(self):
        if self.lineEdit_code.text():
            clipboard = QtGui.QApplication.clipboard()
            clipboard.setText(self.lineEdit_code.text())
            event = QtCore.QEvent(QtCore.QEvent.Clipboard)
            QtGui.QApplication.sendEvent(clipboard, event)
            self.statusBar().showMessage("Copied.", 2000)

    #=-=-=-=-=-=-=-=-=-=-=-=-=-=- WINDOW CONTROLS =-=-=-=-=-=-=-=-=-=-
=-=-=-=-=-
    # Center the window
    def centerWindow(self):
```

```
        screen = QtGui.QDesktopWidget().availableGeometry()
        size = self.geometry()
        self.move(int((screen.width()-size.width())/2),
            int((screen.height()-size.height())/2))
    # Move the window
    def moveWindow(self):
        screen = QtGui.QDesktopWidget().availableGeometry()
        size = self.geometry()
        self.move(int(self.settings.value("MainWindow/x",
(screen.width()-size.width())/2)),
            int(self.settings.value("MainWindow/y", (screen.height()-
size.height())/2)))
    #=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=- SETTINGS =-=-=-=-=-=-=-=-=-=-
=-=-=-=-=-=
    # Read settings on startup
    def readSettings(self):
        numScreens = QtGui.QDesktopWidget().numScreens()
        screen = QtGui.QDesktopWidget().availableGeometry(numScreens-
1)
        size = self.geometry()
        # Get edge locations of window
        if self.settings.value("MainWindow/x"):
            windowLeft = int(self.settings.value("MainWindow/x"))
            windowRight = int(self.settings.value("MainWindow/x")) +
size.width()
            windowTop = int(self.settings.value("MainWindow/y"))
            windowBottom = int(self.settings.value("MainWindow/y")) +
size.height()
            screenRight = int(screen.width()) + screen.x()
            screenLeft =
QtGui.QDesktopWidget().availableGeometry().x()
```

```
            screenBottom = int(screen.height()) + screen.y()
            screenTop = QtGui.QDesktopWidget().availableGeometry().y()
            # If part of the window is off the screen, center it.
            if (windowRight > screenRight) \
                or (windowBottom > screenBottom) \
                or (windowLeft < screenLeft) \
                or (windowTop < screenTop):
                self.centerWindow()
            else:
                self.moveWindow()
        else:
            self.centerWindow()
    # Determine where to save settings file
    def settingsFile(self):
        global __configFileName__
        if sys.platform.startswith('win'):
            __configFileName__ += '.ini'
            __configFilePath__ = os.path.join(__appPath__,
__configFileName__)
            self.settings = QSettings(__configFilePath__,
QSettings.IniFormat)
            if not self.settings.isWritable(): # Use %appdata% if
install dir is read only
                self.settings = QSettings(QSettings.IniFormat,
QSettings.UserScope,
                    __appName__, __configFileName__)
        elif sys.platform.startswith('lin'): # Read settings.conf file
            self.settings = QSettings(__appName__.lower(),
__configFileName__)
        else:
```

```
            self.settings = QSettings(__appName__.lower(),
__configFileName__)

    # Save application settings

    def writeSettings(self):

        # Save window location

        self.settings.beginGroup("MainWindow")

        self.settings.setValue("x", self.x())

        self.settings.setValue("y", self.y())

        self.settings.endGroup()

    #=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=- EXIT =-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=-=-=-=

    # Call exitApplication() when "x" close button is pressed.

    def closeEvent(self, event):

        self.exitApplication()

        event.accept()

    # Actions to perform on application close

    def exitApplication(self):

        self.writeSettings()

        self.close()

if __name__ == "__main__":

    application = QApplication(sys.argv)

    application.setOrganizationName('Nicholas Wilde')

    application.setOrganizationDomain('qrdecoder.googlecode.com')

    application.setApplicationName(__appName__)

    mainWindow = qrDecoder()

    mainWindow.show()

    sys.exit(application.exec_())
```

4. Decoder QR code avec ZBar [Python]

4.1.MAIN

```
# Import Stuff

import sys, os, zbar, Image, sip, string

import PcxImagePlugin, TgaImagePlugin, TiffImagePlugin

sip.setapi("QVariant", 2)

from PyQt4 import QtGui, QtCore

from PyQt4.QtCore import *

from PyQt4.QtGui import *

from src.ui.ui_mainwindow import Ui_MainWindow

from src.ui.ui_aboutwindow import Ui_aboutDialog

import src.ui.images_rc

# App Name

__appName__ = "qrDecoder"

__verFileName__ = 'VERSION'

__configFileName__ = 'settings'

# Determine if application is a script file or frozen exe and get
appPath

# http://stackoverflow.com/a/404750/1061279

if hasattr(sys, 'frozen'):

    __appPath__ = os.path.dirname(sys.executable)

elif __file__:

    __appPath__ = os.path.dirname(__file__)

__verPath__ = os.path.join(__appPath__, __verFileName__)

# Get qrdecoder version from VERSION file

f = open(__verPath__, 'r')

__version__ = f.readline()

f.close()
```

```
#=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= DEF  =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-

def decodeImage(img):

    #print "La imagen esta en", img, "\n"

    scanner = zbar.ImageScanner()

    # configure the reader

    scanner.parse_config('enable')

    # obtain image data

    #pil =
Image.open(unicode('/home/luis/PJE_linux/qrdecoder/qrdecoder-
0.1.2/test/prueba.jpg')).convert('L')

    pil = Image.open(img).convert('L')

    width, height = pil.size

    raw = pil.tostring()

    #pil.show()

    # wrap image data

    image = zbar.Image(width, height, 'Y800', raw)

    print image

    # scan the image for barcodes

    scanner.scan(image)

    # extract results

    for symbol in image:

      if symbol:

          print "Image contain a code."

      else:

          print "Image does not contain a QR code."

    for symbol in image:

      print symbol.type

      # do something useful with results
```

```
      if symbol.type is not symbol.QRCODE:
          print "Image does not contain a QR code."
          return
      #print la informacion del codigo:
      print "La informacion del codigo es: \n ", symbol.data, "\n"
      #return symbol.data
    # clean up
    del(image)
#=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=- MAIN  =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
print "\nQue imagen quieres que decodifique? \n"
#imagen = raw_input (">")
#/home/luis/PJE_linux/qrdecoder/qrdecoder-0.1.2/test/qrcode.jpg
#imagen = "/home/luis/PJE_linux/qrdecoder/qrdecoder-
0.1.2/test/barcode.png"
print "\n"
imagen = '/home/luis/PJE_linux/qrdecoder/qrdecoder-
0.1.2/test/baboon.jpg'
#imagen = '/home/luis/PJE_linux/qrdecoder/qrdecoder-
0.1.2/test/prueba.jpg'
#/home/luis/PJE_linux/qrdecoder/qrdecoder-0.1.2/test/barcode.png
decodeImage(imagen)
#if __name__ == "__main__":
#    mainWindow = qrDecoder()
#    mainWindow.show()
#    sys.exit(application.exec_())
```

5. Prendre une image avec Nao puis decoder QR code avec ZBar [Python]

5.1.MAIN

```
# Import Stuff

import sys, os, zbar, Image, sip, string

import PcxImagePlugin, TgaImagePlugin, TiffImagePlugin

sip.setapi("QVariant", 2)

from PyQt4 import QtGui, QtCore

from PyQt4.QtCore import *

from PyQt4.QtGui import *

from src.ui.ui_mainwindow import Ui_MainWindow

from src.ui.ui_aboutwindow import Ui_aboutDialog

import src.ui.images_rc

from naoqi import ALProxy

# App Name

__appName__ = "qrDecoder"

__verFileName__ = 'VERSION'

__configFileName__ = 'settings'

# Determine if application is a script file or frozen exe and get
appPath

# http://stackoverflow.com/a/404750/1061279

if hasattr(sys, 'frozen'):

    __appPath__ = os.path.dirname(sys.executable)

elif __file__:

    __appPath__ = os.path.dirname(__file__)

__verPath__ = os.path.join(__appPath__, __verFileName__)


# Get qrdecoder version from VERSION file

f = open(__verPath__, 'r')
```

```
__version__ = f.readline()

f.close()

#=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=- DEF  =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-

def showNaoImage(IP, PORT):

  """

  First get an image from Nao, then show it on the screen with PIL.

  """

  camProxy = ALProxy("ALVideoDevice", IP, PORT)

  resolution = 2    # VGA

  colorSpace = 11   # RGB

  videoClient = camProxy.subscribe("python_client", resolution,
colorSpace, 5)

  t0 = time.time()

  # Get a camera image.

  # image[6] contains the image data passed as an array of ASCII
chars.

  naoImage = camProxy.getImageRemote(videoClient)

  t1 = time.time()

  # Time the image transfer.

  print "acquisition delay ", t1 - t0

  camProxy.unsubscribe(videoClient)

  decodeImage(naoImage)

def decodeImage(img):

    #print "La imagen esta en", img, "\n"

    scanner = zbar.ImageScanner()

    # configure the reader

    scanner.parse_config('enable')

    # obtain image data
```

```
    #pil =
Image.open(unicode('/home/luis/PJE_linux/qrdecoder/qrdecoder-
0.1.2/test/prueba.jpg')).convert('L')

    pil = Image.open(img).convert('L')

    width, height = pil.size

    raw = pil.tostring()

    #pil.show()

    # wrap image data

    image = zbar.Image(width, height, 'Y800', raw)

    print image

    # scan the image for barcodes

    scanner.scan(image)

    # extract results

    for symbol in image:

      if symbol:

          print "Image contain a code."

      else:

          print "Image does not contain a QR code."

    for symbol in image:

      print symbol.type

      # do something useful with results

      if symbol.type is not symbol.QRCODE:

          print "Image does not contain a QR code."

          return

            print "La informacion del codigo es: \n ", symbol.data,
"\n"

      #return symbol.data


    # clean up
```

```
    del(image)

#=-=-=-=-=-=-=-=-=-=-=-=-=-=-=- MAIN  =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-

if __name__ == '__main__':

  IP = "nao.local"  # Replace here with your NaoQi's IP address.

  PORT = 9559

  # Read IP address from first argument if any.

  if len(sys.argv) > 1:

    IP = sys.argv[1]

  naoImage = showNaoImage(IP, PORT)

#

#if __name__ == "__main__":

#    mainWindow = qrDecoder()

#    mainWindow.show()

#    sys.exit(application.exec_())
```

6. Datamatrix (Box Choregraphe Aldebaran) [Python]

```
from vision_definitions import*

class MyClass(GeneratedClass):
def __init__(self):
GeneratedClass.__init__(self)
self.cam = ALProxy("ALVideoDevice");

def onLoad(self):
from ctypes import *;
cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId)
+ '/lib/libdmtx.so');
try:
from pydmtx import DataMAtrix;
except:
import sys;
sys.path.append(ALFrameManager.getBehaviorPath(self.behaviorId)
+ '/lib');
from pydmtx import DataMatrix;
self.dm = DataMatrix();

def onInput_onStart(self):
# Get image from Nao's camera
self.cam.subscribe("dmtx", kQQVGA, kRGBColorSpace, 5); #
Warning, use QQVGA (160x120px) otherwise it's horribly long
img = self.cam.getImageRemote("dmtx");
self.cam.unsubscribe("dmtx");
# Decode the image
self.dm.decode(img[0], img[1], buffer(img[6]));
# Parse results
result = list(self.dm.message(i) for i in range(1,
self.dm.count()+1));
self.onStopped(result);
```

7. QR (modification Box Choregraphe Datamatrix Aldebaran) [Python]

```
from vision_definitions import*


class MyClass(GeneratedClass):
def __init__(self):
GeneratedClass.__init__(self)
self.cam = ALProxy("ALVideoDevice");

def onLoad(self):
import sys, os, zbar, Image, sip, string;
from ctypes import *;
cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId)
+ '/lib/libdmtx.so');
cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId)
+ '/lib/CORE_RL_magick_.lib');
cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId)
+ '/lib/CORE_RL_Magick++_.lib');
cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId)
+ '/lib/CORE_RL_wand_.lib');
cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId)
+ '/lib/libzbar-0.def');
cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId)
+ '/lib/libzbar-0.lib');
cdll.LoadLibrary(ALFrameManager.getBehaviorPath(self.behaviorId)
+ '/lib/X11.lib');
try:
from pydmtx import DataMAtrix;
except:
import sys;
sys.path.append(ALFrameManager.getBehaviorPath(self.behaviorId)
+ '/lib');
from pydmtx import DataMatrix;
self.dm = DataMatrix();

def onInput_onStart(self):
# Get image from Nao's camera
self.cam.subscribe("dmtx", kQQVGA, kRGBColorSpace, 5); #
Warning, use QQVGA (160x120px) otherwise it's horribly long
img = self.cam.getImageRemote("dmtx");
self.cam.unsubscribe("dmtx");
# Decode the image
##self.dm.decode(img[0], img[1], buffer(img[6]));
decodeImage(img)
# Parse results
##result = list(self.dm.message(i) for i in range(1,
self.dm.count()+1));
```

```
result = list(symbol.data);
self.onStopped(result);
########### decode zbar ###########
def decodeImage(img):
#print "La imagen esta en", img, "\n"
scanner = zbar.ImageScanner()

# configure the reader
scanner.parse_config('enable')

# obtain image data
#pil =
Image.open(unicode('/home/luis/PJE_linux/qrdecoder/qrdecoder-
0.1.2/test/prueba.jpg')).convert('L')
pil = Image.open(img).convert('L')
width, height = pil.size
raw = pil.tostring()
# wrap image data
image = zbar.Image(width, height, 'Y800', raw)
print image

# scan the image for barcodes
scanner.scan(image)
return symbol.data
```