Katana 450

Quickstart Guide

Programming with KNI

Document No: 233551
Version 1.0.0

Neuronics AG, Technoparkstrasse 1, CH-8005 Zürich
Tel: +41 44 445 16 40, Fax: +41 44 445 16 44

www.neuronics.ch
info@neuronics.ch

Contents

1 Overview

The Katana Native Interface KNI is an open source software library for controlling the Katana robot. KNI is written in C++ and structured so that it can easily be ported to other languages and frameworks. The code is non-platform-specific and can be compiled under both Windows (with the MS Visual C++ Compiler) and Linux (with the GNU Compiler Toolchain).

Since the KNI abstracts the underlying layers, applications can be written for the Katana without having to become involved in the details of the system. It takes just a few function calls to connect and initialise the robot. The protocol for controlling the robot from the PC is abstracted in its entirety. The KNI features an implementation of robot kinematics and path calculation routines for the synchronous control of all axes and the traversing of paths in space with the end effector.

The openness of the common sources also makes the KNI the ideal tool for research and training, since the entire implementation can be traced, as well as modified and adapted at will.

2 Build Environment

2.1 KNI Sources

2.1.1 Reference Sources

Up-to-date sources for the KNI can be downloaded free of charge from the Neuronics website

```
http://www.neuronics.ch/
```

(visit the Download area). Existing customers are informed automatically whenever a new version is made available.

2.1.2 Installation

2.1.2.1 Linux

In a shell, switch to the folder to which the KNI archive was downloaded. Decompress the archive with

```
tar -jxvf KatanaNativeInterface-x.x.x.tar.bz2
```

and switch to the KNI folder with

```
cd KatanaNativeInterface-x.x.x/
```

Once the dependencies and tools have been installed as described in Chapter 2.2, the libraries and demos can be compiled with the

```
make
```

command.

2.1.2.2 Windows

Run the installer and follow the instructions. The installation folder is

```
C:\Program Files\Neuronics AG\KatanaNativeInterface\
```

The installer creates a program group ‘Katana Native Interface’ in the start menu with links to ‘KNI Visual C++ Solution’ and the ‘Demo Applications’ folder.

Once the dependencies and tools have been installed as described in Chapter 2.2, the ’.sln’ file in the installation folder under ’win32’ can be opened with Visual Studio (double-click will open Visual Studio automatically).

On the left side in Visual Studio the Projects (parts of KNI and demos) are listed. To compile the control demo, right-click on the ’Demo-Control’ and choose ’Build’. The program is now in the installation folder under `demo\control\control.exe`

2.1.3 Description of the Source Tree

Following installation, the KNI looks like a simple source tree:

```
-rw-r--r-- 1 user user      359 2008-09-05 11:24 AUTHORS.txt
-rw-r--r-- 1 user user     3780 2007-11-21 12:30 changelog.txt
drwxr-xr-x 3 user user     4096 2008-06-25 14:48 configfiles400
drwxr-xr-x 3 user user     4096 2008-07-29 06:08 configfiles450
drwxr-xr-x 8 user user     4096 2008-09-09 08:41 demo
drwxr-xr-x 6 user user     4096 2008-06-06 11:12 doc
-rw-r--r-- 1 user user     9937 2008-06-06 11:10 Doxyfile
drwxr-xr-x 3 user user     4096 2007-07-24 08:48 drivers
drwxr-xr-x 8 user user     4096 2008-09-05 11:32 include
-rw-r--r-- 1 user user      426 2007-05-22 12:11 INSTALL.txt
drwxr-xr-x 3 user user     4096 2007-09-27 10:38 KNI.net
drwxr-xr-x 5 user user     4096 2007-05-22 12:11 lib
-rw-r--r-- 1 user user    15149 2007-05-22 12:11 LICENSE.txt
-rw-r--r-- 1 user user     1420 2008-09-05 11:37 Makefile
drwxr-xr-x 4 user user     4096 2008-09-05 11:28 py
-rw-r--r-- 1 user user     1553 2008-06-06 10:54 readme.txt
drwxr-xr-x 8 user user     4096 2008-09-04 14:51 src
```

The folders in the root directory contain:

Name	Description
configfiles400	Configuration files for the Katana400
configfiles450	Configuration files for the Katana450
demo	KNI demo programs and examples
doc	KNI Documentation
drivers	Drivers for the USB connection to the Katana
include	The KNI's include files
KNI.net	A KNI wrapper for .NET
lib	The static and dynamic libraries for Linux and Windows
py	Python bindings for the KNI
src	The KNI sources in C++

The files in the root directory have the following functions:

Name	Description
AUTHORS.txt	KNI software authors
changelog.txt	Changes affecting versions
Doxyfile	Descriptor for generating documentation
INSTALL.txt	Installations-Anleitung
KatanaNativeInterface.*	VisualStudio files (Windows)
LICENSE.txt	Licence GNU GPL Version 2
Makefile	Descriptor for compilations with make under Linux
readme.txt	Brief information
WindowsInstaller.iss	Descriptor for generating the Windows installer

2.1.4 The KNI Software Architecture

Figure 2.1 illustrates the basic software architecture of the KNI and the various ways in which it can be integrated into user-specific applications and frameworks.

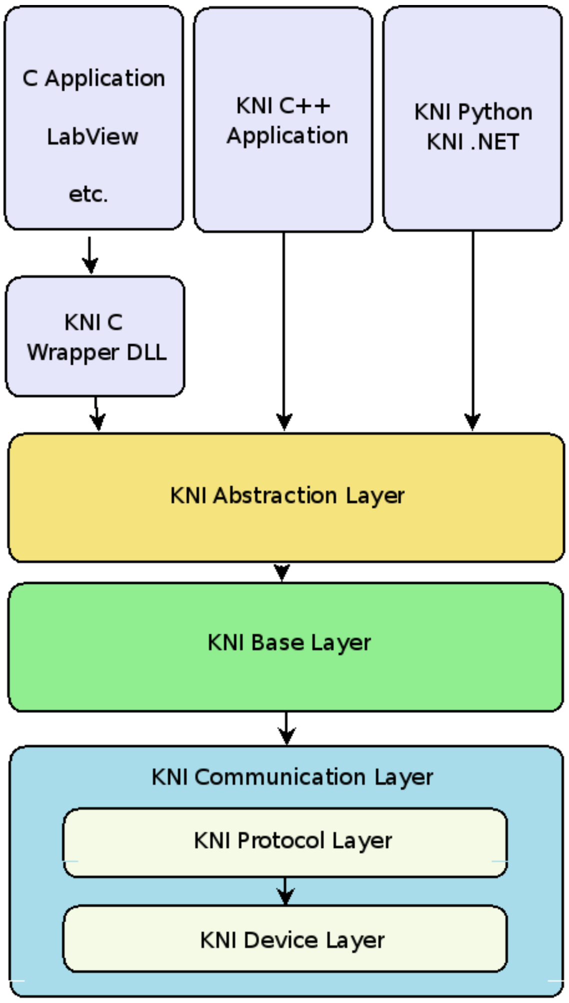

Figure 2.1: The KNI software architecture

2.2 Build Dependencies and Required Tools

2.2.1 Linux und Mac

To compile the KNI, you need the GNU C and C++ Compiler with the standard libraries and the GNU ‘make’ tool.

2.2.2 Windows

The MS Visual C++ Compiler is used to compile the KNI library and C++ programs which use the KNI. Information about Microsoft VisualStudio is available at

```
http://www.microsoft.com/
```

2.3 The Katana Configuration Files

The configuration files are located in the ’configfiles400’ folder for the Katana400 and in the ‘configfiles450’ folder for the Katana450. There is a separate file named accordingly for each robot model.

The configuration values in the files are correct and reliable and should only be changed if you know exactly what you are doing. Making erroneous changes to the configuration can lead to the robot sustaining damage during operation!

3 Programming in C++

The following sections of code shows the use of the KNI for the following purposes: connect, initialise and move.

3.1 Connection

Listing 3.1 shows a section of code for establishing a connection with the Katana. First the network socket is opened, the protocol is instantiated and initialised with the socket.

Listing 3.1: Connection

```
int port = 5566;
char* ip = "192.168.168.232";
std::auto_ptr<CCdlSocket> device;
std::auto_ptr<CCplSerialCRC> protocol;
try {
        // set device
        device.reset(new CCdlSocket(ip, port));
        // set protocol
        protocol.reset(new CCplSerialCRC());
        // initialize protocol
        protocol->init(device.get()); //fails if no response from Katana
} catch(Exception &e) {
        // handle exception
}
```

3.2 Initialisation

Listing 3.2 shows a section of code for initialising the Katana. Once the CLMBase class, which implements the Katana with linear movement, has been instanced, it is initialised with the configuration files and the protocol instance.

Listing 3.2: Initialisation

```
char* configfile = "configfiles450/katana6M90T.cfg";
std::auto_ptr<CLMBase> katana;
try {
        // create and initialize katana object
        katana.reset(new CLMBase());
        katana->create(configfile, protocol.get());
} catch(Exception &e) {
        // handle exception
}
```

3.3 Movements

Listing 3.3 shows examples of point-to-point and linear movements. The Katana always has to be calibrated first. Then the actual pose (in other words, the position and orientation of the robot in space) is read out and a second pose is then constructed. The first movement with 'moveRobotTo()' moves from the current pose to the constructed pose without any specific importance being attached to the path. The second movement with 'moveRobotLinearTo()' takes a linear path back to the original pose.

Listing 3.3: Movements

```
// calibrate katana!
katana->calibrate();
// get actual position
double x1, y1, z1, phi1, theta1, psi1;
katana->getCoordinates(x1, y1, z1, phi1, theta1, psi1);
// create second position
double x2 = x1 + 100.0;
double y2 = y1 - 50.0;
double z2 = z1 + 80.0;
double phi2 = phi1;
double theta2 = theta1;
double psi2 = psi1;
// move to position 2
katana->moveRobotTo(x2, y2, z2, phi2, theta2, psi2);
// move linear back to position 1
katana->moveRobotLinearTo(x1, y1, z1, phi1, theta1, psi1);
```

3.4 Demo Programs

The KNI sources also include some demo programs demonstrating the use of the KNI library. These can be found in the `demo` directory and are included in the compilation automatically when 'make' is called in the root directory. The various example (demo) programs are described briefly in the following sections.

3.4.1 Control

This example is the one recommended by Neuronics for users who are programming with the KNI for the first time. Following start-up with the command below, various inputs can be used to enable direct communication with the Katana, poll data and teach-in programs.

Listing 3.4: The control example program

```
./control ../../configfiles450/(katanatype) IP-address
success: katana initialized
-------------------------------------------
?: Display this help
c: Calibrate the Katana
e: Read the current encoder values
o: Switch motors off/on (Default: On)
r: Switch angle format: Radian/Degree (Default: Rad)
x: Read the current position
v: Set the velocity limits for all motors seperately
V: Set the velocity limits for all motors (or for the TCP if in linear movement mode)
a: Set the acceleration limits for all motors seperately
A: Set the acceleration limits for all motors
w: Read the velocity limits of all motors
W: Read the acceleration limits of all motors
q: Read the Sensors
```

```
y: Set a new position using IK
l: Switch on/off linear movements
<: Add a point to the point list
>: Move to a specific point
 : (space) Move to the next point in the point list
=: write pointlist to file
f: read pointlist from file
g: Open Gripper
h: Close Gripper
n: Set the speed collision limit for all motors seperately
N: Set the speed collision limit for all motors
s: Set the position collision limit for all motors seperately
S: Set the position collision limit for all motors
t: Switch collision limit on
T: Switch collision limit off
u: Unblock motors after crash
d: Move motor to degrees
z: Set TCP offset

1: Move motor1 left
2: Move motor1 right
3: Move motor2 left
4: Move motor2 right
5: Move motor3 left
6: Move motor3 right
7: Move motor4 left
8: Move motor4 right
9: Move motor5 left
0: Move motor5 right
/: Move motor6 left
*: Move motor6 right
.: Toggle Step mode
+: Increase step size
-: Decrease step size

$: Start/Stop Program
p: Start/Stop movement through points list
```

3.4.2 Commands

This example demonstrates communication with the Katana based on the individual commands from the Katana firmware protocol. Following start-up with

Listing 3.5: Commands example program

```
./commands ../../configfiles450/(katanatype) IP-address
```

the input options listed can be used to configure and send the individual commands.

3.4.3 Csharp

This short example demonstrates the integration of the KNI .NET wrapper with csharp for Microsoft Visual Studio.

3.4.4 kni_wrapper

This example demonstrates the use of the kni_wrapper library, which permits calls from non-object-oriented programming languages (C) and development environments (LabView, Matlab).

4 Kinematics

In the Katana configuration file, two different kinematics can be selected under [KATANA] [GENERAL] kinematics. The first is an integrated analytical kinematics model and the second is an external numerical kinematics model which is also used in the Katana4D control software.

4.1 Integrated Kinematics Library

Make the configuration setting ‘kinematics = Analytical’ to use the integrated analytical kinematics model. Although this implementation calculates the inverse kinematics very quickly, it has problems conceptually affecting numerical calculation, meaning that under certain circumstances solutions might not be correct.

4.2 Roboop Kinematics Library

Make the configuration setting ‘kinematics = RobAnaGuess’ to use the external numerical kinematics model. Although calculations take longer, this option is much more reliable. The weak point of the numerical method (namely that in order to calculate the inverse kinematics a ‘good’ starting value needs to be specified approximating the solution) is mitigated by means of combination with the less stable analytical implementation and a few tricks. This combined approach enables results to be achieved which are far better than is the case with the purely analytical model.

5 Integration in Other Languages and Frameworks

5.1 .NET

The KNI.net directory contains a .NET wrapper which permits the KNI to be integrated under .NET. This wrapper can be compiled with the ‘cli’ .NET compiler option. An example Visual Studio project file shows the necessary compiler settings.

5.2 C-based Interfaces

The kni_wrapper KNI library contains a non-object-oriented interface to the KNI which permits calls from ‘C’-based environments. This provides a means of integrating the KNI into measuring and control environments such as LabView or Matlab. An example program demonstrating this wrapper is available under `demo/kni_wrapper`.

5.3 Python

The kni_wrapper and a Python wrapper generated automatically with SWIG enable the KNI to be used in the Python script language. The corresponding sources and files are located in the `py` directory. Calling ‘make’ in this directory will generate the KNI.py and KNI.so files accordingly. These files contain the entire set of KNI libraries and as such can be imported and used directly in a Python shell or a Python script. This also provides a means of achieving script-based KNI integration.

In general terms, the interface is the same as in the case of the kni_wrapper. Only a small number of functions differ with regard to the signature (on account of conceptual differences between C and Python). More detailed information about this appears in the `readme.txt` file which you will find in the `py` directory.

Listing 5.1 shows KNI being used from a Python shell.

Listing 5.1: Using the KNI from Python

```
>>> import KNI
>>> KNI.initKatana("../configfiles450/katana6M90T.cfg", "192.168.1.1")
1
>>> KNI.calibrate(0)
Katana4xx calibration started
...done with calibration.
1
>>> KNI.getEncoder(1)
30500
>>> KNI.moveMot(1, 20000, 50, 2)
1
>>> from KNI import TMovement
>>> home = TMovement()
>>> KNI.getPosition(home.pos)
1
>>> home.transition = KNI.PTP
>>> home.velocity = 50
>>> home.acceleration = 2
>>> KNI.allMotorsOff()
1
>>> """move robot by hand to an other position"""
'move robot by hand to an other position'
>>> KNI.allMotorsOn()
1
>>> KNI.executeMovement(home)
1
>>> KNI.allMotorsOff()
1
>>>
```

5.3.1 LabView

The kni_wrapper can be used to integrate the KNI into Labview. You will find the `kni_labview.vi` example program for integrating the KNI into LabView in the `demo/kni_labview` directory.

5.3.2 Matlab

The kni_wrapper can be used to integrate the KNI into Matlab. You will find the `kni_matlab.m` example program for integrating the KNI into Matlab in the `demo/kni_matlab` directory.