The KUKA Control Toolbox: motion control of KUKA robot manipulators with MATLAB

Francesco Chinello, Stefano Scheggi, Fabio Morbidi, Domenico Prattichizzo

Abstract—The KUKA Control Toolbox (KCT) is a collection of MATLAB functions developed at the University of Siena, for motion control of KUKA robot manipulators. The toolbox, which is compatible with all 6 DOF small and low-payload KUKA robots that use the Eth.RSIXML, runs on a remote computer connected with the KUKA controller via TCP/IP. KCT includes more than 40 functions, spanning operations such as forward and inverse kinematics computation, point-to-point joint and Cartesian control, trajectory generation, graphical display, 3-D animation and diagnostics. Applicative examples show the flexibility of KCT and its easy interfacing with other toolboxes and external devices.

I. INTRODUCTION

A. Motivation and related work

MATLAB [1] is a powerful and widely used commercial software environment for numerical computation, statistical analysis and graphical presentation, available for a large number of platforms. Specific toolboxes (i.e., collections of dedicated MATLAB functions) have been developed in the past few years as supports for research and teaching in almost every branch of engineering, such as, telecommunications, electronics, aerospace, mechanics and control. As far as robotics is concerned, several toolboxes have been recently presented for the modeling of robot systems [2]–[7]. These simulation tools have been inspired by various applicative scenarios, such as, e.g., robot vision [5], [6] and space robotics [3], and have addressed different targets ranging from industrial [4] to academic/educational [2], [5]–[7].

A more challenging problem is to design MATLAB toolkits, offering intuitive programming environments, for motion control of real robots. Some work has been done in this field for the Puma 560 [8], [9]: however this robot is known to have intrinsic software limitations, especially in real-time applications, which have been overcome by more recent manipulators.

In this paper we will focus on the manipulators produced by KUKA [10], one of the world’s leading manufacturers of industrial robots. KUKA manipulators are designed to cover a large variety of applications in industrial settings, such as, for example, assembly, machine loading, dispensing, palletizing and deburring tasks. A specific Pascal-like programming language, called KRL (KUKA Robot Language), has been developed by KUKA for robot motion control. This language is simple and allows comfortable programming [11]. However, it does not support graphical interfaces and advanced mathematical tools (such as, matrix operations, optimization and filtering tasks, etc.), and it does not allow an easy integration of external modules and hardware (e.g., cameras or embedded devices that use standard protocols: USB, Firewire, PCI, etc.). A possible way to overcome these drawbacks is to build a MATLAB abstraction layer upon the KRL. A first step towards this direction has been taken by a MATLAB toolbox, called Kuka-KRL-Tbx, recently developed at the University of Wismar [12]. The authors use a serial interface to connect the KUKA Robot Controller (KRC) with a remote computer where MATLAB is installed. A KRL interpreter running on the KRC, realizes a bidirectional communication between the robot and the remote computer and it is responsible for the identification and execution of all the instructions that are transmitted via the serial interface. Kuka-KRL-Tbx offers a homogeneous environment from the early design to the operation phase, and an easy integration of external hardware components. In addition, it preserves the security standards guaranteed by the KRL (workspace supervision, check of the final position switches of every robot’s axis, etc.), and it benefits from the efficient mathematical tools of MATLAB.

However, Kuka-KRL-Tbx suffers from some limitations:

• The MATLAB commands of the toolbox are in one-to-one correspondence with the KRL functions: this results in an undesirable lack of abstraction that may hinder the user from designing advanced control applications.

• The serial interface does not allow high transmission speeds, and this may represent a serious limitation in real-world tasks.

• The toolbox does not include specific routines for graphical display.

Quanser is currently developing a seamless real-time open-architecture interface to KUKA small robots based on QuaRC [13]. A control prototyping tool generates code from a Simulink diagram and runs it in real-time in Windows, QNX, or Linux. Full Simulink external mode is supported, which means that the control scheme’s parameters can be tuned on the fly and that the feedback data from the robot can be monitored at run-time. However, the introduction of the additional QuaRC layer between the robot and the MATLAB environment appears problematic: in fact, it makes the overall architecture more complex and difficult to supervise.

As concerns “real-time” motion control of KUKA robots, it is finally worth mentioning the OROCONS open source software project [14], developed at the University of Leuven. It provides a general-purpose, modular framework for complex sensor-driven robotic tasks. However, even though a toolbox for creating OROCONS components in Simulink has been recently

The authors are with the Department of Information Engineering, University of Siena, 53100 Siena, Italy.
Email of the corresponding author: scheggi@dii.unisi.it
released [15], the project is not MATLAB native and it relies on four C++ libraries for the real-time, kinematics and dynamics, Bayesian filtering and component parts.

B. Original contributions and organization

This paper presents a new MATLAB toolbox, called KUKA Control Toolbox (KCT), for motion control of KUKA robot manipulators. The toolbox, designed both for academic/educational and industrial purposes, includes a broad set of functions divided into 6 categories, spanning operations such as, forward and inverse kinematics computation, point-to-point joint and Cartesian control, trajectory generation, graphical display, 3-D animation and diagnostics.

KCT shares with Kuka-KRL-Tbx the same advantages and improves it in several directions:

- The functions of KCT are not a MATLAB counterpart of the corresponding KRL commands. This makes the toolbox extremely versatile and easy to use.
- KCT runs on a remote computer connected with the KRC via TCP/IP: this protocol guarantees a higher transmission speed than RS-232, and a time determination comparable to that of the Kuka-KRL-Tbx (in fact, although the TCP/IP connection is more sensitive to retransmissions than a serial one, our communication scheme is not affected by the non-real-time behavior of a KRL interpreter, as that in [12]). A multi-thread server runs on the KRC and communicates via Eth.RSIXML (Ethernet Robot Sensor Interface XML) with a client managing the information exchange with the robot. To the best of our knowledge, KCT is the first software tool presented in the literature that allows easy access to the Eth.RSIXML.
- KCT has several dedicated functions for graphics and animation (plot of the trajectory of the end-effector, plot of the time history of the joint angles, 3-D display of the manipulator, etc.), and includes a graphical user interface (GUI).
- KCT can be easily interfaced with external toolkits, such as, e.g., MATLAB Image Acquisition Toolbox, the Epipolar Geometry Toolbox [5], the Machine Vision Toolbox [6], the Haptik Library [16] or MATLAB routines from the OpenCV Library [17], to perform complex motion control and robot vision tasks.

KCT is fully compatible with all small and low-payload 6 DOF KUKA robot manipulators which use the Eth.RSIXML with 5.4, 5.5 or 7.0 KUKA System Software: the controllers KR C2, KR C2 ed05 and KR C3 (equipped with a real-time 10/100 card) are currently supported by the toolbox. KCT has been successfully tested on multiple platforms, including Windows, Mac and Linux. The toolbox is released under the GNU GPL version 3 and it can be freely downloaded from the web page: http://sirslab.dii.unisi.it/software/kct/

This article is the outgrowth of [18], compared to which we present herein several new functionalities of KCT, as well as a more accurate experimental validation.

The rest of the paper is organized as follows. Sect. II illustrates the main functionalities of KCT. Three applicative examples are reported in Sect. III to show the flexibility of KCT in real scenarios and its easy integration with other toolboxes. Finally, in Sect. IV, conclusions are drawn and possible future research directions are highlighted.

II. THE KUKA CONTROL TOOLBOX

The 6 DOF robot manipulator shown in Fig. 1 will be used as a reference throughout this paper [19], [20]: vector \( \mathbf{q} = [\theta_1, \theta_2, \ldots, \theta_6]^T \) denotes the collection of the joint angles of the manipulator, and \( \mathbf{d}_j^{\mathbf{0}} \in \mathbb{R}^3, j \in \{1, 2, \ldots, 6\} \) the displacement between the center of the \( (j-1) \)-th and \( j \)-th joint of the robot (note that \( \mathbf{d}_1^{\mathbf{0}} \equiv \mathbf{0} \)). The homogeneous matrix \( \mathbf{H}_0^{\mathbf{b}} \in \text{SE}(3) \) relates the coordinates of a 3-D point written in the base reference frame \( (x_0, y_0, z_0) \), with the coordinates of the same point written in the end-effector frame \( (x_6, y_6, z_6) \).

In the next subsections the main functionalities of KCT will be illustrated. In the interest of clarity, the commands of toolbox have been subdivided into 6 categories: Initialization, Networking, Kinematics, Motion control, Graphics and Homogeneous transforms (see Table II). Note that only the Networking and Motion control functions, which rely on the TCP/IP and Eth.RSIXML communication protocols, depend on the peculiar features of the manipulators produced by KUKA (see Fig. 2(a)). The KUKA robot models currently supported by KCT are listed in Table I: to date, the toolbox has been successfully tested on the KR3, KR16-2 and KR5ixxR850 robots.

Fig. 2(b) illustrates the communication scheme between KCT and the robot manipulator. It consists of three parts:

![Fig. 1. Reference robot: 6 DOF elbow manipulator with spherical wrist.](image)
1) A remote computer running KCT under MATLAB,
2) The KUKA Robot Controller (KRC),
3) The robot manipulator.

To establish a connection between the remote computer and the robot controller, KCT provides `kctserver.exe`, a C++ multi-thread server running on the KRC. `kctserver.exe` communicates via Eth.RSIXML (a KUKA software package for TCP/IP-robot interface) with `kctrsiclient.src`, a KRL script which runs the Eth.RSIXML client on the KRC and manages the information exchange with the robot. The server sends the robot’s current state to the remote computer and the velocity commands to the manipulator via `kctrsiclient.src`, in a time loop of 12 ms. `kctrsiclient.src` is also used to define a HOME position (initial position) for the robot arm.

Two classes of constraints affect robot’s motion. The hardware constraints depend on manipulator’s physics and cannot be modified by the user. Conversely, the software constraints (established by the Eth.RSIXML) can be configured at the beginning of each working session via the functions `kctrsiclient.src` or `kctsetbound` (see Sect. III-A for more details). Every time the robot accidentally stops because of the hardware bounds, KCT must be re-initialized. This is not necessary, instead, when the robot halts because of the software constraints.

In what follows, all the angles will be in degrees and all the distances in millimeters.

A. Initialization and networking

The connection between the remote computer and the KRC can be established in a very intuitive way. The information relative to the KUKA robots supported by KCT is stored in the MATLAB file `kctrobotdata.mat` (see Fig. 3 and Table III) and can be accessed by typing:

```matlab
>> kctrobot();
```

The functions `kctinsertrobot`, `kctfindrobot`, `kctdeleterobot` allow to insert, search for and delete robot data. To initialize a robot (for example the model KR3), it is sufficient to write,

```matlab
>> kctinit('KR3');
```

where the argument is a string containing the name of the selected robot, as specified in the file `kctrobotdata.mat`. The TCP/IP connection can then be established by typing,

```matlab
>> kctclient('193.155.1.0', 0.012);
```

where 193.155.1.0 is the IP address of the KRC real-time network card and 0.012 (seconds) is the sampling time.

Fig. 2. (a) Architecture of the KUKA Control Toolbox; (b) Communication scheme between KCT and the manipulator.

Fig. 3. Working envelope and links of robot KR5sixxR850 (the image is drawn from robot’s manual, courtesy of KUKA Robot Group).
By default, KCT communicates with the server using MEX-files (see Table II). However, it also supports MATLAB Instrument Control Toolbox (ICT). To switch between the two modalities, it is sufficient to write kctsettcpip(‘ICT’) or kctsettcpip(‘MEX’). Finally, to close the communication, the user should simply type:

`>> kctcloseclient();`

**Remark 1:** By writing kctclient(‘offline’), the TCP/IP connection is not established. However, this enables the off-line use of all the KCT functions (except those in the Motion control category, c.f. Fig. 2(a) and Sect. II-C).

### B. Kinematics

The state of the manipulator is stored in a 2 x 6 matrix, called `robotstate`, containing the current position and roll-pitch-yaw orientation of the end-effector (first row) and the current joint angles of the robot (second row). This matrix can be accessed using the function:

`>> robotstate = kctreadstate();`

To compute the matrix \(H_0^6\) of the forward kinematics, and the inverse kinematics solution expressed as a joint angles’ vector \(q\), KCT provides the following two functions:

`>> q = [13, 32, -43, 12, 54, 15];`

`>> H06 = kctfkin(q);`

`>> q' = kctikine(H06);`

The function \(p = kctkinerpy(q)\) is analogous to \(kctkine\) but returns the position and roll-pitch-yaw orientation of the end-effector of the robot arm as a vector \(p = [X, Y, Z, \phi, \gamma, \psi]^T\). Likewise, the function \(q = kctikinerpy(p)\) computes the inverse kinematics solution from the vector \(p\).

### C. Motion control

KCT provides several functions for point-to-point motion and trajectory planning. For these operations, the toolbox directly relies on the KUKA robot controller, and the joint or Cartesian information is sent to the KRC in open-loop. Although more sophisticated closed-loop control schemes can be devised, the open-loop solution offers a good compromise between execution time and accuracy of the motion tasks.

The simplest operation one could require is to move the manipulator from an initial to a final desired joint configuration defined by robot’s joint angles or by end-effector’s poses. Let \(q_f = [\theta_1, \theta_2, \ldots, \theta_6]^T\) be the final desired joint configuration of the robot. The function,

`>> qf = [23, 35, 12, -21, 54, 60];`

`>> vp = 20;`

`>> [robotinfo, warn] = kctsetjoint(qf,vp);`
Fig. 4. (a) The interface loaded by the function kctdrivegui; (b) 3-D animation of the robot arm.

moves the robot from the current to the desired configuration. vp is a parameter that varies between 0 and 100 (percentage of the maximum velocity supported by the Eth.RSIXML), the matrix robotinfo contains the time history of the joint angles and warn is a Boolean variable that is set to 1 when an error occurs during robot’s motion. Let now \( p_f = [X, Y, Z, \phi, \gamma, \psi]^T \) be the final desired pose of the end-effector. The function,

```matlab
>> pf = [412, -2, 350, 20, 12, 15];
>> [robotinfo, warn] = kctsetxyz(pf,vp);
```

moves the end-effector of the robot through the three points \( p_1, p_2 \) and \( p_3 \) with velocity vp. The third argument of kctpathxyz is a Boolean variable enabling (when set to 1) the visualization of the 3-D trajectory of the end-effector and the time history of the joint angles at the end of the task.

The function kctpathjoint is analogous to kctpathxyz: the only difference is that the trajectory is defined here in the joint space instead of the operational space. The argument of kctpathjoint is an \( n \times 6 \) matrix \( Q \), whose rows are vectors of joint angles:

```matlab
>> Q = [23, 35, 12, -21, 54, 60;
>>   42, -10, 20, 14, -5, 21;
>>   -15, 31, 10, 12, 20, 80];
>> vp = 20;
>> [robotinfo, warn] = kctpathjoint(Q,vp,1);
```

To stop the robot in the current position, the user must first terminate the execution of the motion control functions using ctrl-c, and then type kctstop(). Finally, to drive the robot back to the HOME position, KCT provides the command kcthome().

A graphical user interface, inspired by Robotics Toolbox’s drivebot GUI [2] can be loaded by typing (see Fig. 4(a)),

```matlab
>> kctdrivegui();
```

It allows the user to easily regulate the joint angles of the robot via 6 sliders, and visualize the corresponding motion of the links through a 3-D animation (see Fig. 4(b)). The trajectory control panel on the right-hand side of the GUI, allows to intuitively plan robot’s point-to-point motion, thanks to the visual feedback of the 3-D animation.

D. Graphics

Several functions are available in KCT for graphical display. The function,

```matlab
>> kctdisptraj(robotinfo);
```
plots the 3-D trajectory of the end-effector, the base reference frame and the initial and final configuration of the robot. The function,

\begin{verbatim}
>> kctdispdyn(robotinfo);
\end{verbatim}
plots the time history of the reference (dashed) and actual robot joint angles (solid). Finally,

\begin{verbatim}
>> kctanimtraj(robotinfo);
\end{verbatim}
creates a 3-D animation of the robot performing the requested motion task.

**E. Homogeneous transforms**

KCT provides a set of transformation functions of frequent use in robotics. Let \( \mathbf{d} \in \mathbb{R}^3 \) be a translation vector and \( \alpha \) an angle. The functions,

\begin{verbatim}
>> d = [100, -23, 300];
>> alpha = 60;
>> Htr = kcttran(d);
>> Hx = kctrotox(alpha);
>> Hy = kctrotoy(alpha);
>> Hz = kctrotoz(alpha);
\end{verbatim}
provide the basic homogeneous transformations generating \( \text{SE}(3) \) for translation and rotation about the \( x, y, z \)-axes.

Suppose now that we wish to move the robot’s end-effector with respect to an external reference frame \( \langle x_w, y_w, z_w \rangle \) different from the base \( \langle x_0, y_0, z_0 \rangle \). This could be useful, for instance, in an eye-in-hand framework where robot’s motion should be referred with respect to the camera frame (see Sect. III-C). Let \( \mathbf{H}_0^w \) be the homogeneous matrix defining the rigid motion between \( \langle x_w, y_w, z_w \rangle \) and \( \langle x_0, y_0, z_0 \rangle \). The function,

\begin{verbatim}
>> H0w = kctrotoz(alpha)*kcttran(d);
>> kctchframe(H0w);
\end{verbatim}
fixes \( \langle x_w, y_w, z_w \rangle \) as new reference frame. All the operations specified by commands executed after `kctchframe` are thus automatically referred to \( \langle x_w, y_w, z_w \rangle \).

**III. ILLUSTRATIVE EXAMPLES**

This section presents three examples demonstrating the versatility and ease of use of KCT in real scenarios\(^1\). In the first example, we show an elementary application of the motion control functions (e.g., for painting or welding tasks in an industrial setting). The second and third example illustrate how to interface KCT with other toolboxes to perform more complex tasks. In particular, in the second example we couple KCT with the Haptic Library [16], in order to control the robot arm with a commercial haptic device (see `kctdemohaptik`). The third example shows the results of a visual servoing experiment realized by combining KCT, MATLAB Image Acquisition Toolbox, the Epipolar Geometry Toolbox [5] and MATLAB routines from the OpenCV library [17] (see `kctdemovision`). The experiments we will present in the next subsections, have been performed using the KUKA KR3 robot with the KR C3 controller.

**A. Drawing a circle**

Suppose we wish to draw the circle,

\[
\begin{aligned}
    x(k) &= 600, \\
    y(k) &= 150 \cos(k), \quad k \in [0, 2\pi], \\
    z(k) &= 150 \sin(k) + 310,
\end{aligned}
\]
on a whiteboard, with a pen mounted on the flange of the KUKA manipulator. To achieve this goal, we must first initialize the robot and establish the TCP/IP communication (recall Sect. II-A). It is then opportune to set the software bounds of the robot using the command,

\(^1\)The videos of the experiments are available at: http://sirslab.dii.unisi.it/software/kct/
where the first and second row of the matrix $B$ contain the lower and upper bounds on the position and orientation (limited to the joint angles $\theta_4$, $\theta_5$ and $\theta_6$) of the end-effector, respectively. Note that \texttt{kctsetbound} enables a MATLAB warning message in the Motion control functions when the workspace’s bounds are violated. To draw the circle with the robot arm, it is sufficient to execute the following lines of code:

\begin{verbatim}
>> k = [0:pi/50:2*pi];
>> x = 600*ones(1,length(k));
>> y = 150*cos(k);
>> z = 150*sin(k) + 310;
>> P = [x',y',z',repmat([0, 90, 0],length(k),1)];
>> kctpathxyz(P,20,1);
\end{verbatim}

where $P$ is the matrix of points defined in Sect. II-C. In our experiment the circle was drawn in 19 sec. with a maximum position error less than 1 mm. The sampling time was set to 15 ms, but because of MATLAB’s and TCP/IP’s communication delays the actual value was around 19 ms. Fig. 5(a) reports the trajectory of the end-effector and Fig. 5(b) the time history of the joints angles, as returned by \texttt{kctpathxyz} (1 sample coresponds to 19 ms). Fig. 6 shows three snapshots of the real robot during the experiment.

\section*{B. Control of the manipulator via a haptic device}

To demonstrate the flexibility and integration capabilities of KCT, we established a bidirectional coupling between a 3-DOF Novint Falcon haptic device (see Fig. 7(a)) and the KUKA robot. The current position of the haptic device is read using the Haptik Library [16], and the velocity of the manipulator is controlled with KCT (obviously, since the workspace of the haptic device is much smaller than that of the robot arm, the position information delivered by the Falcon needs to be suitably scaled). A force feedback proportional to the robot displacement is returned by the haptic interface. Note that the MATLAB environment does not support real-time haptic callback-based services because of the non-deterministic timers: therefore, the proposed setup is not suited for real-time remote manipulation tasks, but it is ideal for rapid prototyping or teaching purposes.

The working frequency of the manipulator, around 83 Hz, is much lower than that of the haptic device (of the order of kHz): to couple the two systems, we then lowered the haptic sample time.

Since the reference frames of the haptic device and of the robot are rotated of $R_h = R_z(-90^\circ)R_x(90^\circ)$ (see Fig. 7(b)), it is convenient to perform the following change of frame (recall Sect. II-E):

\begin{verbatim}
>> kctchframe(kctrotoz(-90)*kctrotox(90));
\end{verbatim}

In this way, all the subsequent robot commands are automatically referred to the frame $\{x_h,y_h,z_h\}$ of the haptic device. The following lines of code show how the Falcon and the robot manipulator interact:

\begin{verbatim}
>> h = haptikdevice;
>> for i=1:200
     tic;
     pos = read_position(h);
     write(h,-1*pos*2.5/30);
     while toc < 0.01
     end
     vel = (read_position(h)-pos)/0.01;
     kctmovexyz([vel(1,1), vel(1,2), vel(1,3),...]
                 [0, 0, 0]*0.015);
     end
>> close(h);
\end{verbatim}

The function \texttt{read_position(h)} reads the position of the haptic device $h$, \texttt{write(h,-1*pos*2.5/30)} returns the force feedback proportional to the robot displacement and \texttt{kctmovexyz} sends the velocity commands to the manipulator. Fig. 8 reports the time history of the $x$-, $y$-, $z$-position of the haptic device (HD, black) and of the end-effector of the robot arm (red). A maximum position error of about 7 mm is achieved.

\section*{C. Visual servoing}

When combined with MATLAB Image Acquisition Toolbox, the Epipolar Geometry Toolbox [5] and MATLAB routines from the OpenCV Library [17], KCT offers an intuitive and versatile environment to test visual servoing algorithms on
real robots. The classical visual servo control by Rives [21] has been chosen as a tutorial to illustrate the main features of KCT in robot vision. The visual control in [21] aims at driving a robot arm from an initial configuration to a desired one, using only the image information provided by a camera mounted on the end-effector. The idea behind the control is that of decoupling the camera/end-effector’s rotation and translation by using the hybrid vision-based task function approach proposed in [22]. To this end, a suitable error function is minimized in order to first rotate the camera until the desired orientation is reached and then translate it toward the desired position. Fig. 9(a) shows the initial and desired configuration of the robot, and the 3-D observed scene in our setup. The desired and initial reference frame $\langle x_d, y_d, z_d \rangle$ and $\langle x_i, y_i, z_i \rangle$ of the camera are rotated of $R = R_z(10^\circ)$ and translated of $t = [-551.38, -52.35, -200.06]^T$. The camera calibration matrix (the image size is $640 \times 480$ pixels), has been estimated using the Camera Calibration Toolbox [23].

In an initialization stage, we first brought the robot to the desired configuration, 

```
>> qD = [64.4, 45.6, -47.4, 89.1, 64.4, -178];
>> kctsetjoint(qD,10);
``` 

and took a picture of the 3-D scene using the Image Acquisition Toolbox. 13 features have been manually selected in this picture (see Fig. 9(b)), and collected in the $2 \times 13$ matrix $\text{feat}_D$. The robot was subsequently brought to the initial configuration, 

```
>> qI = [-6.6, 73.8, -79.8, -47.6, 9, -52.8];
>> kctsetjoint(qI,10);
``` 

where a corresponding set of 13 features ($\text{feat}_I$) has been chosen (see Fig. 9(c)). Before executing the visual servoing algorithm, we performed the following change of frame in order to refer the motion of the robot with respect to the camera frame, 

```
>> rotfr = kctrotoz(posI(4))*kctrotoy(posI(5))*...
        kctrotox(posI(6));
```
Fig. 9. Example 3: (a) Desired and initial configuration of the manipulator with respect to the 3-D scene; (b)-(c) Desired and initial image: the corresponding features and the epipolar lines are shown in red.

\[
\begin{align*}
&\text{>> kctchframe(kcttran(posI([1:3]))*rotfr);} \\
&\text{where posI denotes the pose of the end-effector in the initial configuration with respect to the base reference frame. During each of the 200 iterations of the visual servoing algorithm, the optical flow is calculated with the MATLAB routines from the OpenCV Library, using the pyramidal implementation of the iterative Lucas-Kanade method [24],} \\
&\text{>> type = 'opticalFlowPyrLK';} \\
&\text{>> featAn = cvlib_mex(type,frameP,frameC,featA');} \\
&\text{where frameP is the previous image acquired by the camera, frameC is the current image, featA is a matrix containing the previous features and featAn is a } 13 \times 2 \text{ matrix containing the current features. The fundamental matrix } F \text{ [25, Ch. 9.2] necessary for the implementation of the visual servoing algorithm, is estimated using the following function of the Epipolar Geometry Toolbox,} \\
&\text{>> F = f_esthlm(featAn',featD,4);} \\
&\text{where the third argument of the function indicates the estimation method selected. Once the camera translation vector } tA \text{ and the rotation angles } \omega_X, \omega_Y, \omega_Z \text{ have been computed by the visual servoing algorithm, the KCT function,} \\
&\text{\text{>> kctsetxyz([tA', omegaX, omegaY, omegaZ],10);}}
\end{align*}
\]

is called. Fig. 10(a) shows the migration of the features in the image plane from the initial to the desired configuration (blue: rotation only, red: translation only), and Fig. 10(b) the norm of the error function.

IV. CONCLUSIONS AND FUTURE WORK

In this paper we have presented an open-source MATLAB toolbox for motion control of KUKA robot manipulators. The KUKA control toolbox (KCT) runs on a remote computer connected with the KUKA controller via TCP/IP. It includes a heterogeneous set of functions, spanning operations such as forward and inverse kinematics computation, point-to-point joint and Cartesian control, trajectory generation, graphical display, 3-D animation and diagnostics. Special care has been devoted to keep these functions intuitive and easy to use. The versatility and effectiveness of the toolbox have been demonstrated through three applicative examples.

KCT is an ongoing software project: work is in progress to extend the compatibility of the toolbox to all KUKA industrial robots and to offer new functionalities in the Simulink environment. In order to enhance the prototyping capabilities of KCT, we also aim at creating a robot simulator for offline validation of motion control tasks. The proposed toolbox currently relies on the Eth.RSIXML: in future work, we plan to revise KCT in order to exploit the superior capabilities of the Fast Research Interface (FRI), recently developed for the KUKA lightweight robot [26]. The FRI provides a direct low-level access to the KRC at rates up to 1 kHz (the user can set the flexible cyclic time frame between 1 and 100 ms), while preserving all its industrial-strength features (such as, teaching/touchup, execution of motion primitives, fieldbus I/O and safety). In addition, the UDP socket communication used by the FRI ensures easy integration and portability across a wide range of operating systems.

ACKNOWLEDGEMENTS

The authors are grateful to Dr. François Touvet and to KUKA Roboter Italy for giving us the opportunity to test the toolbox on the robots KR16-2 and KRSiSixR850.

REFERENCES

Fig. 10. Example 3: (a) Migration of the features in the image plane from the initial (dot) to the desired configuration (cross); (b) Norm of the error function.