To Paint What Is Seen: A System Implementation of a Novel Conceptual Hyper-Redundant Chain Robot with Monocular Vision

KeJun Ning, and Florentin Wörgötter
Bernstein Center for Computational Neuroscience, Inst. of Physics III, University of Göttingen, 37077 Göttingen, Germany
Email: nkj@sjtu.org, {ning, worgott}@bccn-goettingen.de

Abstract

This paper presents our system-level implementation on operating a hyper-redundant chain robot (HRCR) with monocular vision. The task is to let our original HRCR prototype, 3D-Trunk, to paint what it sees. The involved processes, technical issues, and solutions for achieving this task are described in this paper. This work represents also an implementation experiment on vision-based human-machine interface, potentially useful for future research and applications.

1 Introduction

Bio-inspired robotic systems are designed to mimic their biological counterparts formally and functionally. Traditional engineering approaches are adapted to implement principles abstracted from the observation of some living creature [1].

In this paper, we present one application of our novel conceptual Hyper-Redundant Chain Robot (HRCR) system, 3D-Trunk [2, 3] (see Fig.1). The target of this work is to let this HRCR “learn to paint by vision-based teaching”.

This application has been inspired by the famous painting elephants from the Maesa Elephant Camp in northern Thailand (see youtube movie [4]). There was another interesting engineering work presented in [5]. An earlier painting experiment on our 3D-Trunk had been done by us via programming it to write Chinese characters (see youtube movie [6]). In the current paper, we will show an overview of the whole system implementation for the learning-by-vision task, as well as the low-level technical issues.

The remainder of this article is organized as follows. In Section 2 of this paper, the target scenario is described. The system-level implementation technology is exhibited in Section 3, with solution on the special kinematics properties of 3D-Trunk. In Section 4, experiments are introduced. Finally, conclusions and the future work are presented in Section 5.

2 The Target: “To Paint What Is Seen”

Different from classical teaching and playback devices of an industrial robot, teaching pendant (operation box),
we want to find a new way to lead a robot to do something easily, which could then also be potentially useful for some 2D applications in an industrial context.

By moving an appointed “object” along a trajectory, the system should recognize it and record its trajectory (or key points along this trajectory). After the required processing, the manipulator can then reproduce this trajectory automatically. This vision system becomes an input device.

In the application experiment described here, 3D-Trunk works with a vision subsystem, not depending on any special off the shelf goods. The actual setup is shown in Fig. 1 (b).

2.1 Overview of 3D-Trunk

3D-Trunk features that all its joints are passive and state controllable, while they share common motor inputs introduced by wire-driven control. The fundamental design, analysis issues, and parameters are presented in [2].

3D-Trunk is built as an independent mechatronics system, all electronics components and micro-controllers are embedded inside. With the supporting of the developed command set, we can access and operate the whole HRCR by connecting a supervisory level controller [2, 3]. As 3D-trunk is not a traditional HRCR design, there are some special characteristics of this new design which need to be taken into account [2]. Extending [2], we will here also describe some additional properties and the inverse kinematics positional control solution for this system.

2.2 Solution for the Target Application

In order to implement the task mentioned above, an Appointed Color Object (ACO) recognition method is employed. Color information can be extracted and processed easily. We use a regular shaped object, with a special color in a cluttered background, to let this system know the key points in the trajectory.

As shown in Fig. 2, the ACO can only be moved on the observation plane (P_Observation), which imposes a constraint on the 3D coordinates of points that lie on the plane P_Observation. As a consequence, the geometrical configuration of the vision sub-system shown in Fig. 2 is sufficient to allow Coordinate Mapping (CM) to produce unique coordinates for each plane pixel in the image.

Let us define \( X_{\text{Image}} = (x_i, y_i)^T \) and \( X_{\text{Motion}} = (x_M, y_M, z_M)^T \) to denote the pixel coordinate of an image and the coordinate of the end-effector on the motion plane (P_Motion) respectively, then the transformation of the CM can be given by,

\[
X_{\text{Motion}} = \eta X_{\text{Image}} \quad (1)
\]

where, \( \eta \) is a transformation matrix, which represents the relations between the coordinate frames shown in Fig. 2.

As shown in Fig. 2, our 3D-Trunk will be controlled to reach these mapped points on the motion plane P_Motion, one by one. During this course, different application tasks can be assigned, e.g., painting along these points on the working plane (P_Working) or just reaching the key points.

The overhead camera configuration shown in Figs. 1(b) and 2 has the advantage of matching pixel coordinates to motion coordinates directly.

3 System-Level Implementation Technology for this Task

In this section, we present the key implementation technology.

Figure 3 shows the organization flow of the software realization. OpenCV [7] is employed to implement the vision processing; and the Robotics Toolbox [8] is used for calculating 3D-Trunk’s kinematics and dynamics [2]. The Robotics Toolbox is a software package, with many fundamental mathematical functions, that allows a MATLAB user to readily execute robotics simulation and analysis [8]. The application program is coded by using Visual C++ (from Microsoft) and can call MATLAB program via MATLAB Engine [9]. MEX files are also developed to bridge MATLAB functions and 3D-Trunk’s hardware sys-
3.1 Vision Processing

OpenCV is an Intel’s free, open source computer vision library [7, 10]. It provides many basic computer vision algorithms via its lower-level APIs, for C/C++ programmers. Figure 4 shows the organization flow of the software realization; the vision processing steps are exhibited on the left side of this figure.

OpenCV provides the interface function for capturing image frames from a digital video device [7, 10], e.g., an ordinary webcam. To enable our system to recognize the target color from the cluttered background according to human vision, more robustly, we need to convert the originally obtained RGB values to another color space, e.g., YUV or HSV [10, 11]. The HSV (Hue, Saturation, Value) color space is more convenient due to its robustness to illumination variations, since the Hue component represents the color independent of lighting [11, 12]. It is more useful than RGB for robotic applications.

In order to reduce the sensitiveness to background noise, a median filter is employed to smooth the image frames. The median filter causes the image to become blurred. This reduces the number of spurious pixels and allows the processing operations to better focus on the target color object.

As shown in Fig. 4, a thresholding operation is employed to segment our ACO from the background. A group of tolerances (upper and lower thresholds) are assigned for this threshold operation. This way we get the binary images for the target color.

The popular Center-of-Gravity computation method [e.g., 13] can be employed to locate the ACO. Denoting the binary values in the binary image by \( I(x,y) \), then from the obtained binary image, we can calculate the ACO’s center, by the zero-order \( M_{00} = \sum_{x} \sum_{y} I(x,y) \) and the first moments over the x and y axes, \( M_{10} = \sum_{x} \sum_{y} x I(x,y) \) and \( M_{01} = \sum_{x} \sum_{y} y I(x,y) \). Then the center of the ACO is as following,

\[
X_c = \begin{bmatrix} x_c \\ y_c \end{bmatrix} = \begin{bmatrix} M_{10} / M_{00} \\ M_{01} / M_{00} \end{bmatrix}
\]  

(2)

Based on Eqs. (1) and (2), we can obtain the ACO’s mapped coordinate on the motion plane \( P_{Motion} \), used to control 3D-Trunk to reach (see Fig. 2).

Furthermore, before calculating the centers of the ACO from the sequential binary images, we need to reduce noise. This process involves two consecutive steps: erosion and then dilation [10], as shown in Fig. 4. This pair of morphological operations can eliminate most spurious pixels from the binary images and ensures the following center calculation to be more accurate.

The vision processing section mentioned above renders the ACO’s positions; the event callback routine puts some of the results to the point queue, according to the user’s intention and operation.

The vision processing shown in Figs. 3 and 4 equips our 3D-Trunk system with a simple but effective “eye”. This subsystem produces the key point information, which can be used to generate a suitable trajectory. Here, a different policy can be employed, depending on the requirements.

3.2 Inverse Kinematics Positional Control of 3D-Trunk

For a serial robot system, inverse kinematics is an inescapable problem. Let us define the vector of joint variables for this N-DOF HRCR (see Fig. 1) by

\[
q = [\theta_1, \theta_2, ..., \theta_N]^T
\]

Here we have \( N=8 \), for our 3D-Trunk. The additional 1-DOF end effector (paint brush holder) is not counted in here. The gap between the working plane \( P_{Working} \) and the motion plane \( P_{Motion} \) is used for the controlled brush pen, see Figs. 1(b) and 2.

When a serial robot is redundant, its inverse kinematics problem admits infinite solutions. The Jacobian pseudoinverse (inverse) based method is extensively studied. A good review of the prior work had been presented in [14].

Generally, the Jacobian pseudoinverse based methods are non-intuitive and computationally complex. In the neighborhood of singularities, they can generate arbitrary joint position vectors [14, 15]. For an N-DOF 3D-Trunk, the joints cannot be operated simultaneously and coordinated, and the joint’s locking resolution is discrete [2, 3]. Furthermore, the present 3D-Trunk’s design has a limited joint range. By the Jacobian pseudoinverse based method, it is also difficult to ensure that the resulting joint-angles are within an acceptable range (i.e., not violating mechanical limits). To overcome the drawbacks mentioned above, as well as to take-care of the special kinematics properties of this new design, a simpler and more suitable method without the pseudoinverse calculation is required here.

Indeed a computationally cheaper solution exists, based on the transpose of the Jacobian matrix [e.g., 14, 16], i.e.,

\[
\dot{q} = aJ^T e
\]  

(3)

Figure 5 Inverse kinematics positional control scheme for 3D-Trunk.
where \( \mathbf{e} = \mathbf{x}_d - \mathbf{x} \) denotes the error between the desired task trajectory \( \mathbf{x}_d \) and the actual end-effector trajectory \( \mathbf{x} \). Here, \( \mathbf{x} \) can be calculated by the forward kinematics of the actual manipulator. \( \mathbf{J}^T \) is the transpose of the Jacobian, and \( \alpha \) is a positive definite (diagonal) matrix that suitably shapes the error convergence \([17, 18]\). By a simple Lyapunov argument, we can guarantee limited tracking errors and null steady-state errors \([14]\).

The implemented command set interface, running in the distributed and embedded micro-controllers inside the 3D-Trunk, supports absolute joint angle control. By iterating and summing Eq. (3), we can implement the inverse kinematics positional control for this novel HRCR design. The implemented control scheme is shown in Fig. 5.

In Figs. 4 and 5, there are some processes following the inverse kinematics calculation. They are caused by the special driving principle of the 3D-Trunk.

### 3.3 Special Kinematics Properties of 3D-Trunk

3D-Trunk is a novel concept based design \([2]\) and, thus, it owns some special kinematics properties. To approach another vector of joint variables from its present pose, we have to use \( N \) “movement steps” to synthesize the incremental vector of joint variables \( \Delta \mathbf{q} \) derived from inverse positional kinematics. Thus, we involve a “Sequence Generator” for dividing \( \Delta \mathbf{q} \) into a sequence, as following,

\[
\Delta \mathbf{q} = \sum_{j=1}^{N} \Delta \mathbf{q}_j
\]  

(4)

where, any one item on the right hand of Eq. (4) is an \( N \times 1 \) vector. Each component of these vectors follows as:

\[
\Delta q_i = 0 \quad (i = 1, 2, \cdots N \quad \text{and} \quad i \neq j)
\]  

(5)

and

\[
\Delta q_j = k \Delta \theta \text{ or } \min(|\Delta q_j|) = \Delta \theta
\]  

(6)

where, \( \Delta \theta \) is the locking resolution which is predefined by the actual mechanical design \([2]\).

For 3D-Trunk, as the joint’s locking resolution is discrete \([2]\), the here existing inverse kinematics problem admits finite and approximate solutions. We need to discretize all the calculated joint variables. After such a discretization operation, some movement steps could be zero vectors (i.e., \( k = 0 \) in Eq. (6)). For speeding up processing, we can ignore all such null operations. In this paper, this processing is called “Filter” (see Figs. 4 and 5). As the related operations are quite easy to understand and implement, here we will not explain them any further.

For the actual system (see Figs. 1 and 2), the necessary coordinate frame design of 3D-Trunk, as well as the related D-H parameters of our original prototype are presented in \([2]\). The Robotics Toolbox V7.1 \([8]\) and MATLAB were used to implement the inverse kinematics positional calculation, shown in Fig. 5.

### 3.4 Softwares’ Interfaces

In Fig. 3, the application program is coded by using Visual C++, and 3D-Trunk’s motion calculation and control are running in a MATLAB environment.

There is a specific interfacing issue, which exists in the implementation architecture, shown in Fig. 3.

As shown in Fig. 3, the main application program calls our MATLAB programs via the MATLAB Engine \([9]\). There are some MEX files also be programmed to output the final results to a developed USB control board (MSP430 and PDIUSB12 based) to bridge 3D-Trunk’s hardware system and some other devices.

So far, we have introduced the key low-level implementation technical and interfacing issues for this system.

### 4 Experiment

Figure 6 shows some screen snapshots, captured during experiments. There are two groups of snapshots in Fig. 6 (a). The raw RGB frames are shown on the left side; and the obtained binary images are shown on the right. Here, the ACO is an orange bottle cap. We find that the vision processing subsystem can always recognize the APO in a cluttered environment somewhat robustly.

Figure 6(b) shows the key point sequence recognized during an operation course. During this course, this system recorded the center of the ACO in real-time, and the recording action was triggered by the operator.

The right image shown in Fig. 6(b) is a result output. It shows the recorded points (centers of the ACO, during a course) in the queue. All points are marked with a disk, and the start point is marked in a bigger diameter. Note, the straight lines between the disks are not stored in memory; they are generated automatically and just connect neighboring points. In this way, we can visualize their temporal sequencing easily.

Figure 6(c) shows a visualized 3D-Trunk model running in MATLAB, its action is same as that of the real 3D-Trunk platform. This 3D plotting function is again supported by the Robotics Toolbox. This visualized model is also very helpful for debugging. In summary we find that this vision processing subsystem exhibits quite satisfactory capabilities.

A simple Chinese character (which means “mountain”) finished by our system is shown in Fig. 6(d). By the method mentioned in this paper, we don’t have to calculate and program a new trajectory to paint a new character. 3D-Trunk can be taught to complete a new work by the monocular vision system.

### 5 Conclusions

Following this research \([2, 3, 6]\), we will in the near future, equip 3D-Trunk with some learning abilities. The content presented in this paper is an implementation design of a vision-based Human-Machine Interface (HMI), and can provide the input for “teaching” methods for our 3D-Trunk.
The system-level implementation of a vision-based manipulation on a new HRCR platform has been described here. Its basic functionality has been implemented and tested on the prototype. We observed that the redrawing is still not completely satisfactory, due to one drawback of the original prototype: it cannot perform a “continuous path task” well but has no problems in a “point to point” task. Even though, the whole implementation architecture shown in this paper has exhibited some application potentials. The technologies disclosed here have been transferred and tested on another five-bar parallel arm mechanism (with different kinematics control formulae), which is an independent manipulator with continuous path fitting capability in 2D work space. At present, we are also working on some careful calibrations and other improvements on the whole system.

3D-Trunk is a special manipulator with 3D moving capability. The present work concerns 2D work space. Possible future work will, thus, implement a similar 3D HMI, supported by a stereovision system.

Acknowledgement

The work was supported by PACO_PLUS from the European Commission and by the BMBF BCCN-Göttingen W3-project.

Literature


