=1

Open Source Software in the Development and Testing of an Image-Guided Robot System

Release 1.00

Peter Kazanzides

July 10, 2006

Department of Computer Science, Johns Hopkins University, USA, pkaz@cs.jhu.edu

Abstract

This paper describes the use of open source software in the development and testing of an image-guided robot system for small animal research, presented at MICCAI 2006[5]. This system relied on a significant amount of open source software, including 3D Slicer, VTK, our own *cisst* software, the NetLib numerical methods, Python, and wxPython (which uses wxWidgets). In addition, several open source development tools were used, including CVS, CMake, and Swig. The paper will be accompanied by the source code and raw data that were used to obtain the results presented at MICCAI.

Contents

1 Introduction

We developed an image-guided robot system to assist with cancer research at the Memorial Sloan Kettering Cancer Center (MSKCC) in New York City. The initial application for the robot system is to verify the

efficacy of PET scans, with various radioactive tracers, for locating hypoxic cancer cells prior to radiation treatment. This is important because hypoxic cells are resistant to radiation and therefore treatment can be improved by tailoring the radiation dosage directed at them. Therefore, the basic requirement for the robot system is to insert an oxygen measurement probe in a three-dimensional (3D) grid pattern defined with respect to a PET scan of a tumor. Anticipated future applications include biopsy and injection of adenoviral sequences for gene therapy, based on CT or MRI images, so the initial requirements specified a design compatible with PET, CT, and MRI imaging modalities.

The design and validation of the system are presented at the MICCAI 2006 conference[5] and are summarized below. The next section includes information about the incorporation of open source software, such 3D Slicer (www.slicer.org), and describes the robot control software in detail. Section 3.1 presents the software environment that was used to collect the data for the results presented at MICCAI 2006. A key point is that we found it more convenient to use Python scripts, with our Interactive Research Environment (IRE)[6], to collect data with the robot. Section 3.2 describes the software that was used to analyze the raw data. This software is based on our open source *cisst* software package (www.cisst.org/cisst), which uses some NetLib numerical methods (www.netlib.org).

2 Image Guided Robot System for Small Animal Research

2.1 System Design

As shown in Fig. 1, the system consists of a a mobile cart that houses the electronics and provides a table top for the four axis robot and display. The robot is composed of of a two degree-of-freedom X-Y horizontal platform and two vertical slides (Z1, Z2). The rodent bed (Fig. 2) is mounted on the X-Y platform. The Z1 axis positions a cannula at the skin surface and the Z2 axis drives the measurement probe (or needle, in the future) through the cannula. The rodent bed fits inside the bore of the small animal imaging scanners at MSKCC and contains 4 markers (fiducials) to enable the registration between image and robot coordinates. The user provides initial estimates of the marker positions in the image, which are then refined by image processing operations in the application software. The user manually guides the robot's registration probe (mounted in place of the cannula) to the marker positions using a force control mode[7]. The image to robot transformation is computed from the 4 marker positions in image and robot coordinates.

The robot controller consists of a rackmount computer connected via Ethernet to a DMC-2143 controller board and AMP-20540 power amplifier (Galil Motion Control, Rocklin, CA, USA). The controller provides low-level servo control of the four robot motors. Application software on the PC sends position goals via Ethernet to the controller, which then moves each joint to its goal. The power amplifier provides eight analog inputs, three of which are used for the interface to the XY and Z1 force sensors that enable the force control mode described above.

2.2 Software Design

The software consists of two primary modules: the application (visualization) software, based on the open source 3D Slicer package, and the robot control software, which is a C++ "wrapper" class around the Galil API. The robot control software uses portions of our open source *cisst* package.

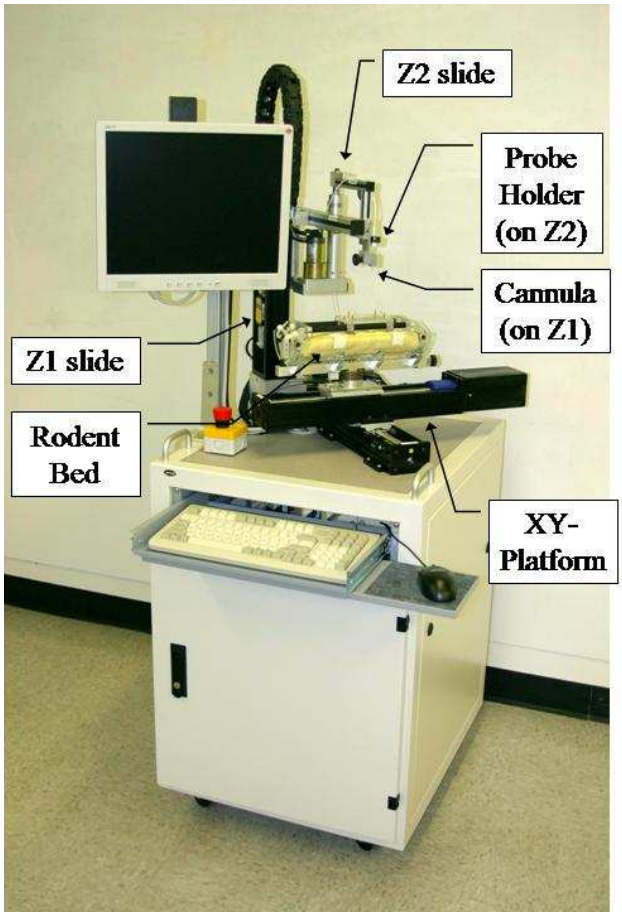

Figure 1: Robot system

Figure 2: Rodent bed with marker bridge

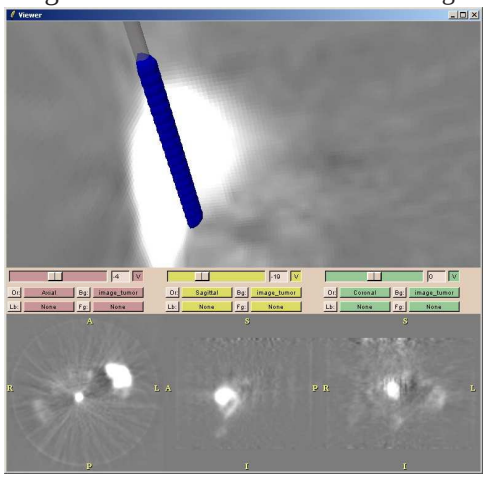

Figure 3: Measurement track in PET

Application Software

The requirements for the application software are to provide visualization of PET (and eventually CT and MRI) image data, compute the registration between image and robot coordinates, and guide the user through the steps of the procedure, as outlined in [5]. The main steps are:

1. Load and display the PET image data.
2. Allow the user to locate the 4 markers in the image.
3. Allow the user to guide the robot's registration probe to the 4 markers.
4. Compute the registration between image and robot coordinates.
5. Allow the user to specify a set of measurement tracks on the PET image (see Fig. 3).
6. Move the robot to the specified points and record the probe measurements and corresponding voxel intensities.

We satisfied all the requirements by using the 3D Slicer software, with the addition of custom modules for the following:

1. A module (vtkRodent), added to the Slicer modules directory, for reading the PET image and controlling the robot. This module includes C++ code that is linked with the robot control library, described in Section 2.2, and a TCL script, Rodent.tcl, that handles the integration with Slicer.

2. A region growing filter that was added to the Slicer base software. This was used to find the centroids of the markers in the PET image, based on seed points identified by the user.

3. A standalone executable, invoked from Rodent.tcl, that performs the image to robot registration. This program expects an input text file containing the positions of the 4 markers in image and robot coordinates and outputs a text file with the 6 transformation parameters (3 translations and 3 rotations).

The vtkRodent module consists of the C++ source files vtkRodent.h and vtkRodent.cxx, and the TCL script, Rodent.tcl. The vtkRodent.h file defines the `vtkRodent` class, which includes a member object of type `mskccRobot`, described in the following section. The vtkRodent.cxx file includes wrappers for `mskccRobot` methods; for example, the `vtkRodent::SetSpeed` method calls the `mskccRobot::SetSpeed` method. Because `mskccRobot` member functions can throw exceptions, the corresponding `vtkRodent` member functions use `try...catch` blocks to catch them; otherwise, the application software would crash if an exception was thrown. The vtkRodent module also includes code for reading the PET image, in a proprietary format, and transforming it into a format readable by Slicer.

The region growing filter probably should have been put in the modules directory (if we were more proficient with Slicer) or replaced by an existing ITK filter (if we were more familiar with ITK). Therefore, it will not be discussed further.

The registration method was implemented as a separate executable so that it could be used to process test data collected outside the Slicer-based application program. As noted in [5], this software currently uses an iterative method, based on a variation of Powell's method, but the test data was analyzed using the well-known closed form solution based on a matrix singular value decomposition (SVD) [1][9] and we intend to use this for the application in the near future. Therefore, Section 3.2 will present the registration software that was actually used for the data analysis.

Robot Control Software

Figure 4 depicts the architecture of a typical robot controller. We used a commercially-available Galil motion controller board that provides the low-level servo control software. We were also able to implement the supervisory (trajectory) control loop on the controller board due to the following simplifying factors:

1. Our robot system is a Cartesian design (X, Y, Z1, and Z2), so motion in Cartesian coordinates is equivalent to motion in joint coordinates (i.e., no kinematic computations are required).

2. We used force sensors that provided analog outputs corresponding to forces applied on the X, Y, and Z1 axes. Therefore, we did not require kinematics or coordinate transformations for the force control mode.

Thus, our robot control library consists of a C++ wrapper class, `mskccRobot`, for the API provided by the Galil motion controller (see files mskccRobot.h and mskccRobot.cpp). This class also contains a small supervisory control loop that is written in the Galil-specific interpreted language and is downloaded to the controller during initialization (see the `force_controller` character array in mskccRobot.cpp).

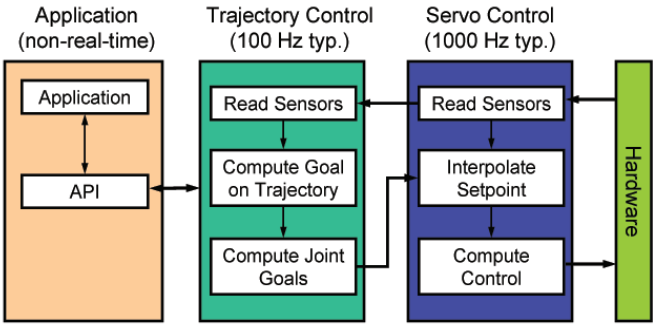

Figure 4: Robot Controller Architecture

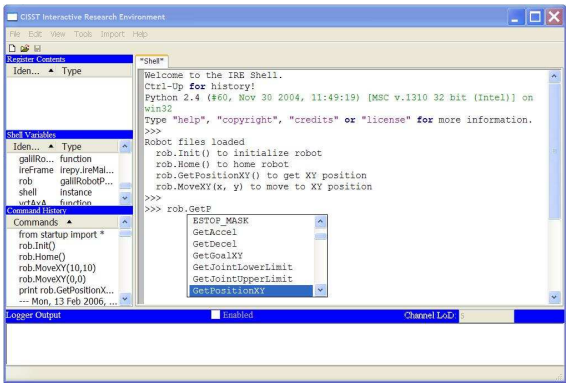

Figure 5: IRE User Interface

The robot control library relies on the *cisstCommon* and *cisstVector* libraries from the open source *cisst* software package. We did not use the *cisstDeviceInterface* library, described in [6],[4], because it was not available when we began the project.

The `mskccRobot` class, and the *cisstCommon* and *cisstVector* classes, are automatically wrapped for use with Python by the open source Swig tool (`www.swig.org`). The existence of a Python interface to the C++ software proved invaluable during system development. Historically, all robot programming languages have been interpreted, rather than compiled, because this enables interactivity between the user, the software, and the robot. Although graphical user interfaces are nice, there is a limit to the number of buttons and controls that can be displayed before overwhelming the user (not to mention the amount of effort necessary to create all these items). By automatically wrapping every public member function, it is possible for the user to invoke any class operation from the interpreter command line. Although any interpreted language could have been used, we chose Python because it is a modern object-oriented programming language that has a syntax similar to C++. Because our software is written in C++, it is logical to have an interactive shell that is familiar to C++ programmers.

We created the Interactive Research Environment (IRE), Fig. 5, to provide an interactive environment that is more user-friendly than the standard Python shell [6]. The IRE relies on the open source wxPython package (`www.wxPython.org`), which includes:

- A Python interface to the open source wxWidgets cross-platform GUI package (`www.wxWidgets.org`), which is written in C++.
- The Py package (formerly PyCrust), which contains a GUI-based Python shell with features such as command completion, calltips, and history.

The *cisst* package facilitates the integration of the IRE into C++ programs (e.g., via an “IRE” button in the graphical user interface). The *cisstCommon* library provides an *object registry* that enables the Python and C++ software to share objects. This is especially useful when embedding the Python interpreter in an application, using the *cisstInteractive* library, because it allows the user to modify C++ objects from the Python interpreter. Although the Python interface was not used with the Slicer-based application software, it proved valuable for system testing and all robot data reported in [5] was collected using the Python test programs described in Section 3.1.

3 Software for Phantom Experiments

The experiments reported in [5] were performed with a phantom (Fig. 6) that contains 20 small hemispherical holes (and larger holes that were not used) at 5 different heights. Four of these holes (1, 3, 10, 11) are designated the *registration markers* because they are arranged in the same pattern as the four markers on the rodent bed. The phantom is constructed of Delrin for compatibility with all image modalities of interest (PET, CT, MRI) and was accurately machined using a CNC.

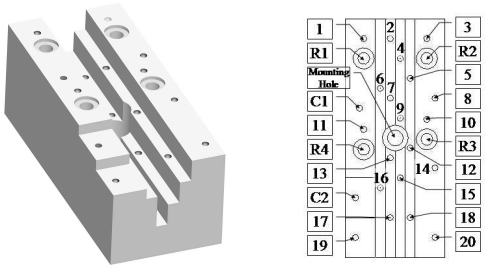

Figure 6: Phantom

For microPET imaging of the phantom, the hemispherical holes were filled with contrast agent (radioactive tracer). After scanning, the Slicer-based application software was used to find the centroid of each marker in the image. For the robot measurements, the robot's registration probe was manually guided to each accessible hole, using the Python test program described below.

3.1 Data Collection

The attached Python test program, collect.py, was used for the data collection. The `CollectAll` function uses the known CNC coordinates to facilitate the data collection process by moving the robot's registration probe to the vicinity of each of the 20 markers (this assumes that the CNC coordinate system is roughly aligned with the robot coordinate system). The `Collect` function is called to find the position of each marker. The procedure is as follows:

1. Put the robot in force control mode so that the user can guide the registration probe to the marker.
2. When the Enter key is pressed, record the robot's position and then slowly move the probe 2.5 mm above the marker.
3. Query whether the user wishes to collect the point again. If so, go back to Step 1.

We found this method to be necessary to handle errors due to compliance. The registration probe would often deflect when manually placed in the hemispherical hole; this deflection was evident when the probe was retracted in Step 2. An alternative solution would have been to automate the procedure by using the force feedback to seat the ball inside the hole (similar to the ball-in-cone method reported in [7]). This was not possible, however, due to limitations of the low-cost force sensors used for this system.

3.2 Data Analysis

The key results in [5] were measurements of the Target Registration Error (TRE) and the Fiducial Localization Error (FLE) for the image and robot markers (for a definition of these terms, see [8]). We used two methods for estimating the FLE:

1. Compute the Fiducial Registration Error (FRE) that results from matching the marker positions in the image or robot coordinate system with the known marker positions (in CNC coordinates). Estimate FLE from FRE using the equation given in [3] (equation (3) in [5]). To improve the robustness of the estimate, all available markers (14 image, 16 robot) are used for the registration.

2. Compute the Fiducial Distance Error (FDE), which is the difference, for each pair of markers, between the measured distance and the known distance (see `Distances.cpp`) [5]. FLE is estimated from FDE using an empirical relationship that was determined by simulation.

The TRE was obtained by computing the transformation between image and robot coordinates (using the 4 registration markers) and then transforming all other markers into a common coordinate system. The TRE is the mean value of the error (distance) between each set of corresponding points.

Both the FLE computation (first method) and the TRE computation require a registration between a number of paired points. We created an analysis program, `Accuracy.cpp`, based on the method described by Arun[1], as modified by Umeyama[9] to handle cases where the determinant is -1. This program used the fixed size vector and matrix classes provided by *cisstVector* and the SVD function, `nmrSVD`, provided by the *cisstNumerical* wrapping of the LAPACK3E method (`www.netlib.org`). As a result, the amount of new code required to implement the registration was minimal.

The analysis program expects one or two command line arguments that specify the file names of the input data. If one file name is specified, the program performs a registration between the CNC coordinates (hard-coded in the program) and the specified input data (image or robot coordinates), using all available markers. This provides the FRE that is used to estimate FLE using the first method described above. If two file names are specified, the program performs a registration, using the four registration markers, between the two data sets (e.g., image and robot coordinates). The program then transforms all marker positions to the same coordinate system and computes the difference between each matched pair of markers. The TRE is obtained by computing the mean difference between the marker pairs that were not used for registration. The raw data and the software files used to process the data are attached to this paper.

4 Conclusions

We constructed an image-guided robot system to assist with cancer research and performed phantom experiments to measure its accuracy. Both the construction and phantom experiments benefited significantly from the use of open source software. The 3D Slicer software package provided the bulk of the application software – it was only necessary for us to add custom modules for reading the PET image data (proprietary format), locating the marker centroids in the image, performing the image to robot registration, and controlling the motion of the robot. Although we do not consider ourselves proficient in the use of 3D Slicer, it was still easy enough for us to accomplish our goals. Our task would have been even easier if we had been more experienced with Slicer and ITK; it is likely that some of our custom modules (e.g., locating marker centroids, performing image to robot registration) are already available in these packages.

The robot control module and the test and analysis software benefited from the open source *cisst* software developed in our lab and available at `www.cisst.org/cisst`. This was the first time that we used the Galil motion controller, so we did not already have a device interface for it. Furthermore, the *cisstDeviceInterface* library was not mature when we began this project, so our solution (the `mskccRobot` class described earlier) is not compatible with it. We plan to create a `ddiGalil` class, derived from `ddiDeviceInterface`, that uses the Command Pattern mechanism described in [6].

3D Slicer and *cisst* also utilize a number of other open source software packages, such as VTK for visualization, TCL and Python for scripting, and wxPython/wxWidgets for cross-platform GUI development. Most of our development tools are also open source, as described in [6]. In fact, the only proprietary software used for this project is Microsoft Windows, Microsoft Visual Studio.NET 2003, and the Galil drivers (binaries available from `www.galilmc.com/support/download.html`).

The existence of these open source software packages facilitated the development and testing of this image-guided robot, so that it could be made available for cancer research as quickly as possible. The robot was installed at MSKCC in January 2005 and is currently in experimental use[2].

Acknowledgments

Emese Balogh performed the initial implementation, at JHU, of many of the 3D Slicer modules, including the software for reading the PET image data, locating the marker centroids in the images, and wrapping the robot control library for use with Slicer. Jenghwa Chang made improvements to these modules at MSKCC. Iulian Iordachita and Jack Li performed the mechanical design and assembly of the robot system. Anton Deguet, Ofri Sadowsky, Ankur Kapoor, and Andy LaMora contributed to the development of the *cisst* package. Bixiu Wen, Pat Zanzonico, Andrei Pugachev, Qing Chen, Jose Campa, and C. Clifton Ling provided valuable assistance and input at MSKCC. Gabor Fichtinger and Russell Taylor provided assistance and advice at JHU. This work is supported by NIH RO1 CA84596 and NSF EEC 9731748.

References

[1] K.S. Arun, T.S. Huang, and S.D. Blostein. Least-squares fitting of two 3-D point sets. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 9(5):698–700, Sep 1987. 2.2, 3.2

[2] J. Chang, B. Wen, P. Kazanzides, P. Zanzonico, R. Finn, and C. Ling. PO2 measurements in animal tumors using an image-guided robotic system. In *AAPM 48th Annual Meeting*, Orlando, FL, Aug 2006. 4

[3] J.M. Fitzpatrick, J.B. West, and C.R. Maurer. Predicting error in rigid-body point-based registration. *IEEE Trans. on Medical Imaging*, 17(5):694–702, Oct 1998. 1

[4] A. Kapoor, A. Deguet, and P. Kazanzides. Software components and frameworks for medical robot control. In *IEEE Intl. Conf. on Robotics and Automation*, pages 3813–3818, Orlando, FL, May 2006. 2.2

[5] P. Kazanzides, J. Chang, I. Iordachita, J. Li, C. Ling, and G. Fichtinger. Design and validation of an image-guided robot for small animal research. In *MICCAI*, Copenhagen, Denmark, Oct 2006. (document), 1, 2.2, 2.2, 2.2, 3, 3.2, 1, 2

[6] P. Kazanzides, A. Deguet, A. Kapoor, O. Sadowsky, A. LaMora, and R. Taylor. Development of open source software for computer-assisted intervention systems. 1, 2.2, 4

[7] P. Kazanzides, J. Zuhars, B. Mittelstadt, and R.H. Taylor. Force sensing and control for a surgical robot. In *IEEE Intl. Conf. on Robotics and Automation*, pages 612–617, Nice, France, May 1992. 2.1, 3.1

[8] C.R. Maurer, J.M. Fitzpatrick, M.Y. Wang, R.L. Galloway, R.J. Maciunas, and G.S. Allen. Registration of head volume images using implantable fiducial markers. *IEEE Trans. on Medical Imaging*, 16(4):447–462, Aug 1997. 3.2

[9] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans. Pattern Anal. and Mach. Intell.*, 13(4):376–380, Apr 1991. 2.2, 3.2