The Surgical Assistant Workstation

Release 1.00

Balazs Vagvolgyi[1], Simon P. DiMaio[2], Anton Deguet[1], Peter Kazanzides[1], Rajesh Kumar[1], Christopher Hasser[2], and Russell H. Taylor[1]

July 25, 2008

[1]Johns Hopkins University, Baltimore, MD
[2]Intuitive Surgical Inc., Sunnyvale, CA

Abstract

The Surgical Assistant Workstation (SAW) is a software development framework that can be used to develop new applications in robot-assisted surgery with augmented visualization. Robot-assisted laparoscopic surgery and micro-surgery—as presented by the *da Vinci* telerobotic system and the Johns Hopkins Steady Hand, respectively—can be improved by providing fully-integrated image guidance and information-enhanced intra-operative assistance to the surgical team and to the surgeon in particular. This paper describes several use case applications of the SAW framework, as well as a novel user interface library being developed to support 3D interactive menu systems and overlays for surgical guidance.

Latest version available at the Insight Journal [http://hdl.handle.net/1926/1466]

Contents

1 Introduction

The Surgical Assistant Workstation (SAW) project provides a modular software framework to support rapid prototyping of telesurgical research systems, enhanced 3D visualization, and user interactions based on 3D manipulations (see Figure 1). This framework includes a library of components that can be used to implement master-slave or collaborative robot control systems with support for complex video pipelines and a novel interactive surgical visualization environment. This library is intended to be easily extensible, such that developers can add support for their own robotic devices and associated hardware platforms. The intent is similar to mobile robot frameworks, such as Player [4], that support various actuators and sensors. In addition, the Image Guided Surgery Toolkit (IGSTK) [3] is targeted at image-guided interventions, but currently focuses primarily on navigation and 2D user interfaces.

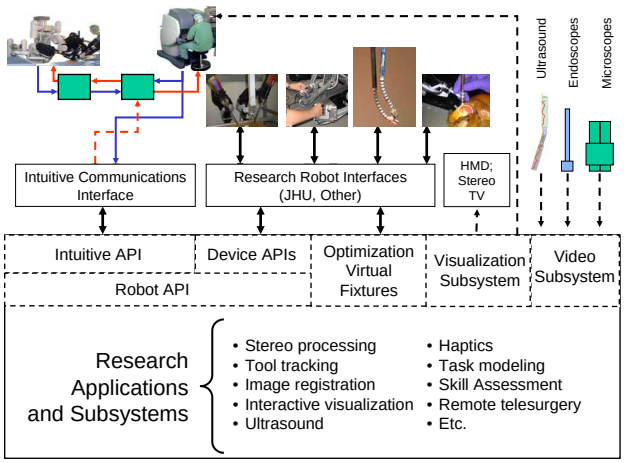

Figure 1: Architecture Overview

The paper is organized as follows: Section 2 describes several use cases for the SAW framework, in order to clarify its application scope. Section 3 outlines two key components of the framework, namely the video processing pipeline that is used to manage video acquisition and processing for visualization and vision-based guidance, as well as a surgical interface manager that can be used to build complex interactive 3D user interfaces and visualization environments.

2 Use Cases

This section describes a selection of specific use cases that provide context and motivation for the SAW framework. One common element is the use of a *Masters-as-Mice mode* that allows the surgeon to interact with the SAW user interface directly from the surgical console, where the term "Master" refers to the user input manipulandum of a telesurgical system. This allows the surgeon to use the Masters as 3D pointers/cursors to activate controls (e.g., buttons displayed in the stereo view) or to directly manipulate graphical objects. When using a *da Vinci* surgical system (Intuitive Surgical Inc., Sunnyvale, CA) [5], the Masters-as-Mice mode is entered by pressing the *master clutch pedal* on the surgeon's console, followed by closing and releasing both master grips. Typically, one master is used as the primary 3D cursor input. The surgeon can activate a control (e.g., a menu item) by placing this 3D cursor over it and closing the master grip; this is analogous to a "mouse click". The surgeon exits Masters-as-Mice mode by releasing the master clutch.

2.1 Image Guidance: *da Vinci* with Laparoscopic Ultrasound Instrument

This scenario assumes that a laparoscopic ultrasound (LapUS) instrument is attached to one of the active Patient Side Manipulators (PSMs) and is inserted through a cannula into the body cavity. If the ultrasound transducer has been calibrated to the *da Vinci* instrument, then it should be possible to dynamically overlay the ultrasound image on the tracked LapUS instrument in the stereo view of the *da Vinci* master console [8].

In this use case, the surgeon enters Masters-as-Mice mode to select one of the following options: "LapUS

Flashlight View" and "LapUS Inset View." In the first case, the ultrasound image plane is overlaid onto the ultrasound instrument, and moves with the LapUS instrument. In the second case, the ultrasound image plane is displayed as an inset in the stereo view (i.e., it is not affixed to the LapUS instrument).

2.2 Image Guidance: *da Vinci* with Medical Image Overlay

In this use case, the surgeon enters Masters-as-Mice mode and selects "View Image Volume" to overlay a medical image volume within the surgical console of a *da Vinci* system. Once an image volume is selected and loaded, the surgeon can move the 3D pointer over the crosshair at the origin of the image volume and "click" (close the grip) to perform any of the following actions:

1. PAN: the primary Master TeleManipulator (MTM) translates the origin of the image volume.
2. ZOOM: the secondary MTM is selected (by closing its grip); the distance between the primary and secondary MTMs controls the zoom level.
3. ROTATE: the secondary MTM is selected and positioned over one corner of the image volume; relative motion between the primary and secondary MTMs controls volume orientation.

The surgeon can reformat the slice plane by moving the 3D pointer over one of the corners of the slice plane and "clicking" (closing the grip) to achieve any of the following actions:

1. TRANSLATE: the slice plane follows the motion of the primary MTM.
2. ROTATE: the secondary MTM is selected and positioned over a second corner of the slice plane; relative motion of the primary and secondary MTMs controls slice plane orientation.

2.3 Haptic Guidance: Virtual Fixtures

In this use case, the surgeon shares control of the robot with the computer process. The goal of these task-dependent computer processes is to provide assistance to the surgeon by limiting the robot's motion within restricted regions and/or by influencing it to move along desired paths. The literature on virtual fixtures (VFs) classifies these behaviors as either forbidden region virtual fixtures (FRVFs) or guidance virtual fixtures (GVFs). FRVFs allow desired motion only in a predefined task space, whereas GVFs provide assistance in keeping the motion on desired paths or surfaces. In this architecture, FRVFs are defined using a fixed number of virtual planes, whereas GVFs can be selected from a predefined set of primitives [9].

The surgeon enters Masters-as-Mice mode to initiate "virtual fixture mode" and proceeds to place constraint planes within the surgical field. Using the 3D pointer, the surgeon may grab and move control points that describe these planes, in order to adjust the position and orientation of forbidden regions. This interactive manipulation of the virtual fixtures is similar to that described in Section 2.2.

2.4 Research Hardware: Snake Robot

This use case describes an application in which a *da Vinci* console is used to control a snake robot system that is being developed at Johns Hopkins University for laryngeal surgery [6]. Such a hybrid system could

potentially leverage the *da Vinci* platform to develop and test specialized surgical slave mechanisms, as proof-of-concept demonstrations.

In this case, both the *da Vinci* console and the Snake robot are connected to the SAW application. Endoscopic video outputs would also be connected to the SAW for display on the surgeon's console.

3 Application Framework Components

The SAW application development framework is modular by design, such that it can be used to implement different physical architectures and applications. It provides the main application context and event loops, as well as communication mechanisms for interconnecting modules and devices with the application logic. A generic API is used to interface with robotic manipulators, while extensive support is provided for implementing video pipelines and an interactive user interface.

This paper focuses on SAW modules that have been developed to support visualization and user interaction at the "surgeon's console". For a broader discussion of the *cisst* libraries that underlie the SAW framework, as well as the research interface that is used to interface with the *da Vinci* surgical system (Intuitive Surgical Inc., Sunnyvale, CA), please refer to the following companion papers in the proceedings of this workshop: [1, 2]. The heirarchy of SAWsoftware components is illustrated in Figure 2.

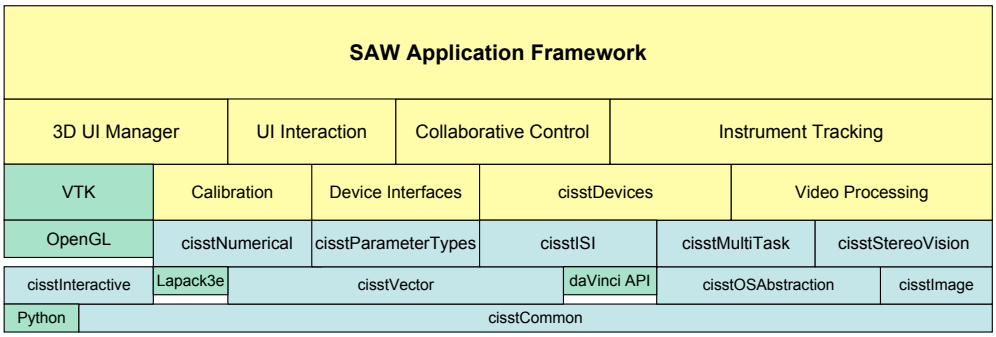

Figure 2: SAW Components

3.1 The Video Processing Pipeline

A library of video pipeline functions provides mechanisms for acquiring and processing streams of images, including stereo endoscopic video and 2D ultrasound. The SAW framework is designed to facilitate the development of augmented reality displays for surgical navigation. This is done either by providing means to enhance the images displayed in the surgical console, or by fusing multi-modality image data into one easily comprehensible visual display. For live image enhancement, the Video Processing Pipeline incorporates a number of image filters that may be used to correct various optical and image distortions or to perform simple image manipulations, such as image resizing, cropping, rotating, spatial filtering, etc. This same pipeline may also include more complex machine vision algorithms, such as instrument and tissue tracking, and image registration. Graphical overlays and user interface widgets are rendered using the Visualization Toolkit (VTK, Kitware Inc., New York); therefore, SAW is capable of leveraging a wide range of existing 3D rendering capabilities. The library provides infrastructure for displaying live stereoscopic video and multiple layers of complex 3D overlays of various imaging modalities, including live 2D ultrasound or preoperative 3D CT/MRI data sets that may be registered to the live video background.

3.2 Interactive Surgical Interface

The surgeon's user interface facilitates interactive manipulation and visualization of 2D and 3D data objects—including medical images and video—directly within the surgical console, as described in several of the use cases outlined in Section 2. A 3D graphical user interface manages user interaction from various input devices (including the master manipulators of the *da Vinci* Surgical System) and renders a menu system and graphical overlays to the stereo display of the surgical console. A 3D User Interface Manager—as opposed to a 2D Window Manager—provides application-level "widgets" and interaction logic.

Many surgical interventional procedures may benefit greatly from the ability to render images on stereoscopic displays similar to the surgical console of the *da Vinci* system. Stereo vision provides 3D perception for the human visual system, which not only improves immersion but enables new kinds of 3D user interactions, thus allowing the surgeon to visualize and interact with three dimensional data—such as preoperative image volumes, intraoperative imaging and navigation systems—that augment the surgical field. The SAW architecture incorporates support for a wide variety of stereoscopic displays for visualization, while remaining compatible with traditional monoscopic rendering. While 3D virtual environments are beneficial for many applications, they present a number of new challenges for system developers, namely in terms of visual ergonomics.

Central to the SAW User Interface Manager is the concept of *Behaviors* and a *Behavior Manager*.

Behaviors

In the SAW architecture, individual features (or application modules) are called *behaviors*. For example, each of the use cases described in Section 2 constitutes a *behavior*. The "*da Vinci* with Medical Image Overlay" use case can be considered to be a behavior that is capable of loading CT datasets, displaying them on an image overlay layer on top of the stereoscopic live camera view, and allowing the surgeon to interact with this dataset through the surgical console.

The SAW software framework provides an easy and transparent interface to access system resources and user inputs and to seamlessly integrate application logic into the Surgical Assistant Workstation. Behaviors may create toolbars using a standard set of widgets provided by the SAW framework. These widgets include buttons, two- or multiple-state check boxes, static text widgets, menu bars, and sliders. The toolbars are managed by the *Behavior Manager*, which notifies the behavior when the user interacts with any of the graphical widgets. In this way the framework can provide a consistent user interface look and feel—with careful consideration of the limited visual field of a surgical display.

Although most use cases can be implemented using a single behavior, there are application scenarios for which multiple behaviors may be needed to perform more complex tasks. For example, in order to implement dynamic virtual fixtures, one might want to use features from both a "Video to CT Registration" behavior and a "Static Virtual Fixtures" behavior. In this case, instead of implementing a single new behavior, it makes sense to build a communication channel between the two existing behaviors in order that the registration filter may adjust the virtual fixtures should the anatomy move within the surgical field. In terms of user experience, this new dynamic virtual fixture feature would then only be an option that can be enabled or disabled in the 'Virtual Fixtures' behavior, which can in turn automatically start the registration behavior when needed.

Behaviors may be in one of the following three states: *idle, background, foreground*. *Idle* behaviors are not allowed to perform any computations, the behavior manager does not dispatch any user events to them, and they cannot draw anything on the display. In the *background* state they are allowed to render to the surgical

console and will receive notification of user and system inputs/events. In the *foreground* state the behavior manager displays the behavior's toolbar and dispatches all user inputs to the behavior.

Behavior Manager

The module responsible for managing behaviors and providing the interface for user interactions is called the *Behavior Manager*. All behaviors added to an application get registered with the Manager, which generates the behavior control menu on the main user interface. This control menu lists all behaviors implemented by the application, and displays their status changes over time. When the application starts up, the manager initializes all registered behaviors automatically and sets them to the idle state. Initially, no visual overlays are displayed on the screen. When the surgeon switches to the SAW user input mode, the manager displays the main behavior menu and lets the user select which behavior to open. Once a behavior is opened, the manager sets it to the *foreground* state and enables behavior-specific user interactions. While the behavior might have its own controls on the screen, the manager keeps the behavior control menu visible so that the user can exit or hide the running behavior(s) at any time, similar to the 'close' and 'minimize' buttons in traditional 2D windowing systems. The Behavior Manager is in fact implemented as a special type of behavior that implements the menu system, and that in turn enables application behaviors and facilitates transitions between forground and background behaviors.

3.3 User Interactions

One of the main contributions of the SAW project is that it provides a standard interface for 3D pointer devices, in a way that is similar to their 2D counterparts. User interactions on traditional 2D user interfaces have been widely studied and understood. Users have become quite familiar with the computer mouse, which allows full freedom to explore the two dimensional desktop space. In three dimensions, however, the methods for efficient user interaction have yet to be developed. The SAW architecture implements a user interface programming environment where 3D user inputs from a variety of input devices can be transparently mapped to a simple set of standard pointer events, such as click, drag, or move gestures. The standard set of events then can easily be accessed by the behaviors regardless of the type of pointer device. The translation from raw inputs to pointer events is done by device-dependent input interpreters.

The role of the pointer device is crucial in many user interfaces. In the virtual surgical environment, the position of the cursor indicates the place of interest or intent and usually pointer actions trigger different application features. The simplest example is clicking a button on the main behavior menu to start a behavior. In the case of the *da Vinci* system, the master grip position is directly mapped to the 3D cursor position in the virtual 3D enviroment. As the surgeon moves the master handle in the console, the cursor follows his/her movements. To "push" a button, the surgeon moves the cursor over the button of the behavior, then quickly closes and opens the grip with his/her fingers to generate a click event. A more complex example is to grab a 3D object in the medical image viewer behavior and move it to another 3D position. In this case, the user needs to move the cursor inside the volume of the object, close the grip to grab it, then move the master arm to reposition the cursor and the object with it. Once satisfied, the user opens the grip to release the object and leave it in its current position.

Some of these functionalities can be found in existing toolkits (e.g., VR-VTK [7]); however, these are either incomplete for our application, or impose an execution flow that is incompatible with that of the SAW framework (e.g., VR-VTK relies on VTK's events).

2D Interactions

Graphical widgets, menus and toolbars that are handled by the Behavior Manager are rendered on the *control layer*. The control layer is an overlay plane—perpendicular to the camera axis—that appears as a 2D desktop layer upon which all 2D widgets are placed. The apparent depth of this layer is a function of the cursor position.

During perceptual experiments we observed that when the control layer is maintained at a fixed apparent depth within the stereo view, users tend to have difficulty when trying to adjust to a different stereo disparity each time they switch their attention between the work area and the control layer, particularly when there is a large difference in depth between these focal regions.

In order to ameliorate this problem, we have developed an algorithm to dynamically change the apparent depth of the control layer as the user manipulates the work area (the control layer is correspondingly re-scaled such that the size and location of 2D widgets remain static within the view, regardless of their depth). For example, when the user maneuvers a 3D cursor, the depth of the control layer tracks the depth of the cursor, such that the user can easily focus on both objects simultaneously, thus significantly reducing visual stress and fatigue.

3D Interactions

Behaviors may create and manipulate 3D objects on the video overlay through the built-in helper functions of the SAW user interface. Once a 3D object is created, the behavior may subscribe to its user interface events. The framework takes care of interfacing with the pointer hardware (surgical console) and translates pointer actions into object manipulation events. The framework provides default handlers for every user interface event (such as grabbing or scaling objects in the 3D space); however, the behavior also has the option to override the default actions with custom logic.

3D Pointers

Although visual ergonomics is an extremely important factor in interventional imaging, we need to pay equal attention to control ergonomics. For example, in the 2D user interface world the computer mouse was specifically designed for easy navigation, while in the operating room environment surgeons will end up using devices not originally designed for manipulating 3D user interfaces. Some of the simplest user interface actions, such as clicking, holding down a button, or dragging objects are intuitive when using a computer mouse, due to the ergonomic placement of mouse buttons on the device, as well as careful decoupling of mouse motion and button actions. However, teleoperated surgical systems, such as the *da Vinci* robot, usually do not have buttons on their master arms because they were not designed to perform clicking actions. When applied to the *da Vinci*, the SAW framework maps the master grip activations to pointer actions, as described earlier in the paper.

During user experiments we observed that simply mapping the grip state (i.e., open or closed) to pointer actions (e.g., clicking) introduces a high error rate in widget interaction accuracy. The root cause of the problem is that users find it difficult to hold the master arms stationary while closing and opening their grip, i.e., there is too much motion coupling between pointer positioning and click actuation. As a result, the cursor position may change significantly by the time the user has completed a click action, thereby causing the pointer events generated by the input interpreter to deviate from the user's original intentions. The SAW interaction logic includes a novel algorithm for inferring the user's intended click actions with high accuracy.

The algorithm maintains a history of motion for both the arm and the grip, and after detecting a grip event (such as click or push down), it searches back in the motion history to find the moment when the user started performing the action.

4 Conclusion

The SAW framework will be made available as an open source project in the near future. We invite interested groups and individuals within the research community to make use of, and to further develop, the SAW concept and its code base. We hope that this work will be used as a platform for research and development in telesurgical systems, and that it will provide a convenient means of sharing and integrating new technologies and methods that will ultimately benefit patients. For further information, please monitor our project website, located at http://www.cisst.org/cisst/saw/.

Acknowledgements

This work is supported in part by National Science Foundation and National Institutes of Health grants EEC-9731748 and R42-RR019159.

References

[1] A. Deguet, R. Kumar, R. Taylor, and P. Kazanzides. The *cisst* libraries for computer assisted intervention systems. Insight Journal, 2008. http://hdl.handle.net/1926/1465. 3

[2] S. DiMaio and C. Hasser. The *da Vinci* research interface. Insight Journal, 2008. http://hdl.handle.net/1926/1464. 3

[3] K. Gary, L. Ibáñez, S. Aylward, D. Gobbi, M. Blake, and K. Cleary. IGSTK: An Open Source Software Toolkit for Image-Guided Surgery. *IEEE Computer*, pages 46–53, 2006. 1

[4] B. Gerkey, R. Vaughan, and A. Howard. The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *Intl. Conf, on Advanced Robotics (ICAR)*, pages 317–323, Jun 2003. 1

[5] G. Guthart and J. Salisbury. The Intuitive^{TM} telesurgery system: Overview and application. *Proc. IEEE International Conference on Robots and Automation*, 1:618–621, 2000. 2

[6] A. Kapoor, N. Simaan, and R. H. Taylor. Telemanipulation of Snake-Like Robots for Minimally Invasive Surgery of the Upper Airway. In *MICCAI Medical Robotics Workshop, Copenhagen*, pages 17–25, 2006. 2.4

[7] A. J. F. Kok and R. van Liere. A multimodal virtual reality interface for 3D interaction with VTK. *Knowledge and Information Systems*, 13(2):197–219, 2007. 3.3

[8] J. Leven, D. Burschka, R. Kumar, G. Zhang, S. Blumenkranz, X. Dai, M. Awad, G. Hager, M. Marohn, M. Choti, C. Hasser, and R. Taylor. Davinci canvas: A telerobotic surgical system with integrated, robot-assisted, laparoscopic ultrasound capability. *MICCAI*, pages 811–818, 2005. 2.1

[9] M. Li, A. Kapoor, and R. Taylor. Telerobot Control by Virtual Fixtures for Surgical Applications. In *Advances in Telerobotics Human Interfaces, Bilateral Control and Applications*, pages 381–401, 2007. 2.3