Software Architecture of the da Vinci Research Kit

Zihan Chen et. Al

A paper review summary by Anurag Madan
Computer Integrated Surgery II, 2019

Project overview

Some anatomies are very fragile, and it is extremely undesirable to contact or even damage those tissues during surgery. For example, during skull drilling process of mastoidectomy, the facial nerve is very close to the surgical field. Any contacts with the facial nerve can cause paralysis. Therefore, the goal of this CIS II project is to develop complex virtual fixtures to protect critical anatomies such that the burden of surgeons can be reduced. In this project, the virtual fixtures will be developed for the Mark I robot from the company Galen Robotics Inc. This robot is a platform robot that can be used for many surgeries, mostly head and neck ones, which makes it a suitable robot for this project.

Paper selection

Initially, the goal of the project is to enable bimanual teleoperation of the EyeRobot with the dVRK, which are both developed at the LCSR, JHU. For this, the paper titled 'Software Architecture of the DaVinci Research Kit' was selected since it provides the general software architecture of the dVRK and CISST, and it's interface with ROS. However, the project was changed after the paper presentation. So, this paper does not have much relevance to the current project goal, but it is still a very informative paper on the dVRK framework, and can be used to understand the fundamentals of robotic software.

Problem and key results

Telerobotic systems have a proven track record in several application domains. However, s. One obstacle had been the lack of a robust common research platform, but this has recently been addressed by the availability of systems such as the da Vinci Research Kit (dVRK). The authors explain the software architecture of the dVRK, and the reasoning behind the design. The design had the following requirements:

- Scalability to multiple master and slave robot arms.
- Easy reconfiguration, such as adding or removing arms or even splitting the system into multiple independent setups.
- Use of a familiar software development environment, such as C++ on a Linux PC, for all levels of the software architecture.
- Real-time performance for high-frequency, low-level robot control.
- Ability to integrate with other high-level robot components and development environments, such as Matlab and Python, via middleware.

The requirements were met by the following key features:

- Use of a high-bandwidth field bus that supports daisychain connection, multicast communication, and an efficient (low overhead) software interface, which satisfies the requirements for scalability and reconfigurability.
- A real-time, component-based framework that enables high bandwidth, low latency control.
- Bridge or proxy components that provide interfaces between the real-time component-based framework and other systems.

Scalable and Reconfigurable hardware

The goal for this design was to provide deterministic performance with low latency to – minimize total number of communication transactions and support daisy chain connections, which allows scalability and reconfigurability by introducing a different control PC.

The daisy chain sequence was implemented using FireWire bus. The system takes advantage of the FireWire broadcast and peer-to-peer communication capabilities. The system sends a query, and waits for 5N microseconds, where N is the number of boards. Each board sends feedback after 5i microseconds, where i is the board node number. The PC reads and broadcasts as a single packet to one board, and each board extracts its own commands based on the node number.

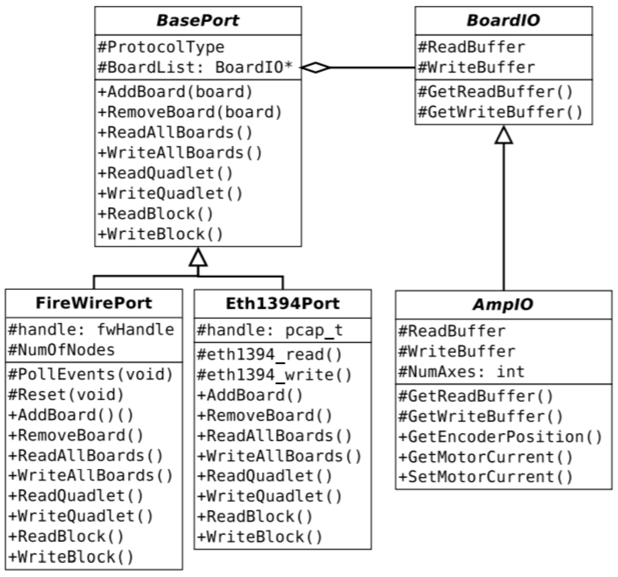
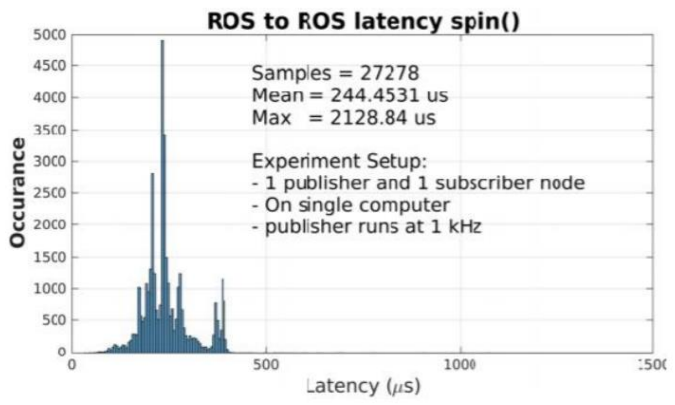
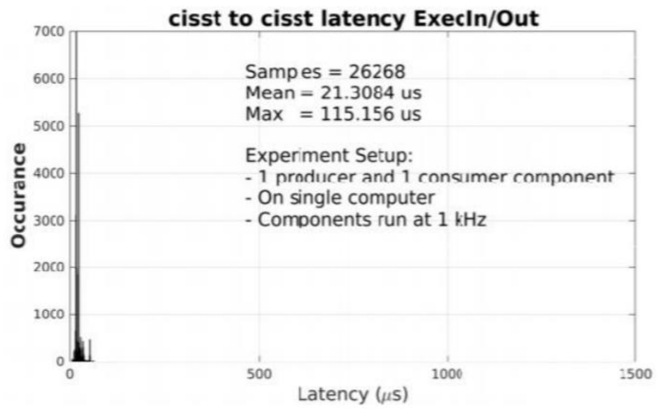


Figure 1: UML class diagram of interface software (subset of class members shown)

Real-Time Framework for Robot Control

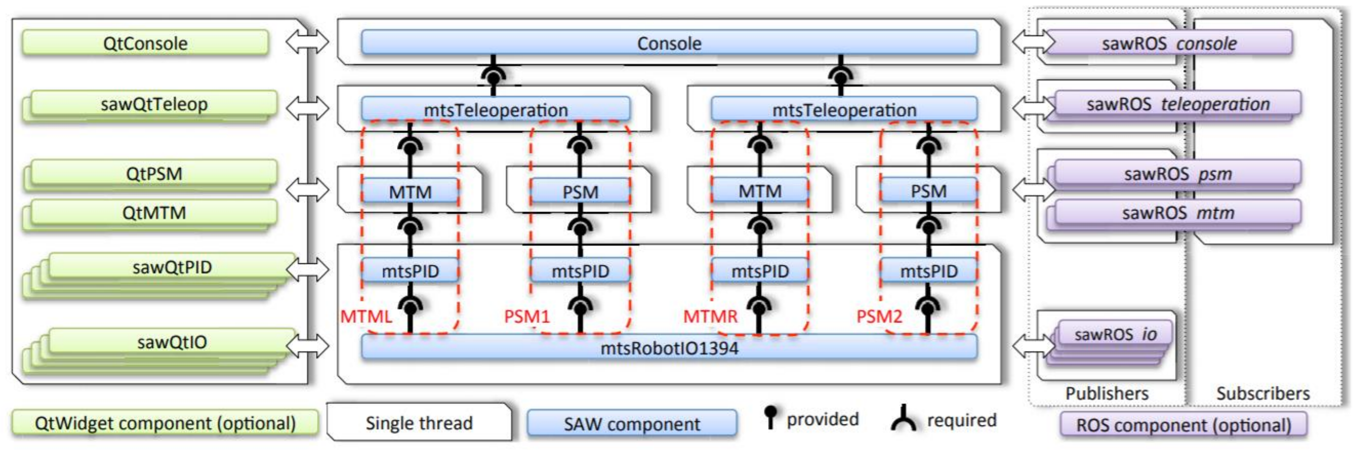
The main goals for this design were to provide a component-based framework, with well-defined interfaces between components, and Efficient communication between components to support control rates of 1 kHz or more. The goals were achieved using a ROS publisher subscriber system for efficient communication. Since the publisher subscriber system has significant communication latency for high frequency controllers, the JHU CISST multithread paradigm was added to reduce latency.





Latency using ROS vs Latency using CISST

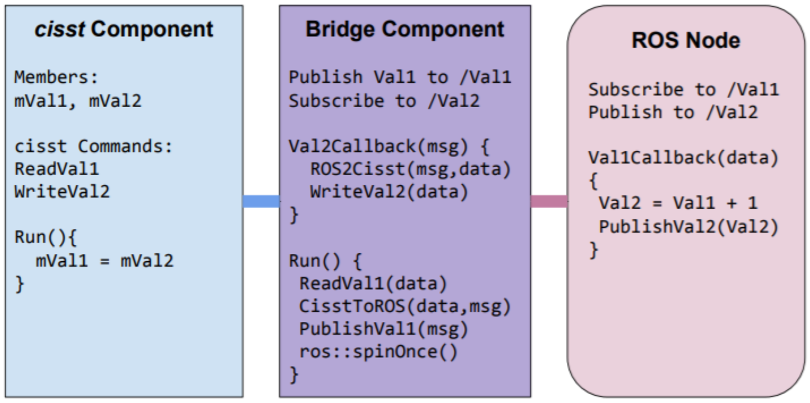
The shared memory, multi-threaded design is better suited for the high frequency, low-latency control requirements for the dVRK. The design extends from hardware interface to low level and mid level controller.



Robot tele-operation control architecture with two MTMs and two PSMs, arranged by functional layers and showing thread boundaries

CISST to ROS Integration

ROS was chosen due to high acceptance and usage in the robotics community, large number of packages for control and visualization and standardized middleware. A bridge based design to communicate between CISST and ROS was implemented. The bridge contains a CISST publisher and subscriber, conversion functions, and bridge component. CISST was converted to Catkin packages, making it easier to access from ROS. The Robot models were also converted to URDF or Xacro files for simulation purposes.



```
Members:
mVal1, mVal2

cisst Commands:
ReadVal1
WriteVal2

Run(){
  mVal1 = mVal2
}
```

```
Publish Val1 to /Val1
Subscribe to /Val2

Val2Callback(msg) {
  ROS2Cisst(msg,data)
  WriteVal2(data)
}

Run() {
 ReadVal1(data)
 CisstToROS(data,msg)
 PublishVal1(msg)
 ros::spinOnce()
}
```

```
Subscribe to /Val1
Publish to /Val2

Val1Callback(data)
{
 Val2 = Val1 + 1
 PublishVal2(Val2)
}
```

Simple example of CISST to ROS bridge

Conclusion

In conclusion, the authors Presented a scalable, reconfigurable, real-time and ROS-compatible software architecture for DVRK with a distributed hardware interface, real-time component based framework and multithreading paradigms, and a High level integration with the ROS ecosystem.

Assessment

Pros

The CISST to ROS bridge is a very useful feature given that a vast majority of the robotics community uses ROS, and the bridge makes it very easy to transition between the two. Another useful feature was the multithreading paradigm. ROS systems can have latency in communication which can cause issues in real-time systems, especially medical applications, which is what a lot of research using the dVRK is focused on. Having parallel threading to reduce latency is a very useful feature to have.

Overall, the paper is well structured, with each feature of the architecture explained in detail, including the reason it was needed and how it was implemented.

Cons

The primary con in this paper is there is not much experimental validation that is provided. The only piece of data shown is the ROS vs CISST latency graph. However, no experiment of validation has been provided for other architecture features that were implemented.

Relevance to current project

As mentioned, the project was changed after the paper presentations, and therefore, this paper review does not have a strict relevance to the current project. However, there are still important aspects that can be adapted into the project. The most important is the software for the Galen robot is also built on CISST libraries, and therefore, the implementation of the CISST to ROS bridge is very useful even for this project. Moreover, as a future scope, the dVRK may be implemented to teleoperatively control the Galen robot, and this is a good study for the groundwork of that scope.

References

- Chen, Z., Deguet, A., Taylor, R. H., & Kazanzides, P., (2017). Software Architecture of the da Vinci Research Kit. Proceedings – 1st IEEE International Conference on Robotic Computing, IRC 2017, 2017 http://doi.org/10.1109/IRC.2017.69