Collaborative Robotics Toolkit (CRTK): Open Software Framework for Surgical Robotics Research

Yun-Hsuan Su, Adnan Munawar, Anton Deguet, Andrew Lewis, Kyle Lindgren, Yangming Li, Russell H. Taylor, Gregory S. Fischer, Blake Hannaford and Peter Kazanzides

***Abstract*—Robot-assisted minimally invasive surgery has made a substantial impact in operating rooms over the past few decades with their high dexterity, small tool size, and impact on adoption of minimally invasive techniques. In recent years, intelligence and different levels of surgical robot autonomy have emerged thanks to the medical robotics endeavors at numerous academic institutions and leading surgical robot companies. To accelerate interaction within the research community and prevent repeated development, we propose the Collaborative Robotics Toolkit (CRTK), a common API for the RAVEN-II and da Vinci Research Kit (dVRK) - two open surgical robot platforms installed at more than 40 institutions worldwide. CRTK has broadened to include other robots and devices, including simulated robotic systems and industrial robots. This common API is a community software infrastructure for research and education in cutting edge human-robot collaborative areas such as semi-autonomous teleoperation and medical robotics. This paper presents the concepts, design details and the integration of CRTK with physical robot systems and simulation platforms.**

***Keywords* — medical robotics, collaborative robotics, teleoperation, open software framework.**

I. Introduction

A. Background

Surgery is a compelling application domain for both telerobotic systems and cooperatively-controlled robots. In telerobotic systems, the surgeon uses a master console to control one or more robots that operate on the patient. These systems can: (1) provide high dexterity through small incisions for minimally-invasive surgery, (2) operate while being exposed to ionizing radiation, such as during computed tomography (CT) or x-ray imaging, (3) fit in confined spaces such as the bore of a magnetic resonance imaging (MRI) scanner, and (4) allow expert surgeons to operate on patients in remote locations. Cooperatively-controlled robots allow the surgeon and robot to share control of the surgical instrument, often using a wrist-mounted force sensor to measure the surgeon's intent. Potential benefits include: (1) reduction of hand tremor, (2) enforcement of safety barriers, (3) mechanical guidance to improve precision of motion, and (4) hybrid control strategies that partition the task space into autonomous (sensor-based) and manual directions of control.

Although numerous research and commercial surgical robot systems have been introduced to provide one or more of the above benefits, to date, the da Vinci Surgical System (Intuitive Surgical, Inc., Sunnyvale, CA) [1], has achieved the most success with over 5,000 robot systems installed at hospitals around the world and with more than 6 million surgeries performed on patients. The da Vinci, however, only provides direct teleoperation, where a human surgeon controls each action of the patient-side robots, even though research in semi-autonomous teleoperation, including supervisory control, shared control, and other co-robotic methods, has been active for decades. Furthermore, the da Vinci does not currently support teleoperation across large distances, though that capability was demonstrated in 2001 with a competing system [2].

In the research domain, however, there is a relatively newer trend of incorporating semi-autonomous agents trained via machine or reinforcement learning (ML/RL) into the surgical workflow. Some notable research in this area includes autonomous algorithms for performing soft-tissue suturing [3], an automated approach for sinus surgery using computer navigation techniques [4], characterization and automation of soft-tissue suturing using a curved needle guide [5] and automation of cutting/creasing sub-tasks while employing learning by observation [6]. Additionally, [7] presents a holistic approach to simplifying the task of manipulator positioning prior to surgeon interaction, and [8] demonstrates a telemanipulated surgical simulation designed for heart surgery. A trainable infrastructure is presented in [9] with controllable dominance and aggression factors for automating repetitive surgical tasks. Lastly, a shared infrastructure for training learning agents by motion decomposition of sub-tasks is developed in [10].

Of course, these research systems cannot be applied directly to an actual surgical procedure but instead rely on mock

Y-H. Su is with the Dept. of Computer Science, Mount Holyoke College, South Hadley, MA USA, msu@mtholyoke.edu.

A. Lewis is with the Dept. of Mechanical Engineering, K. Lindgren, and B. Hannaford are with the Dept. of Electrical and Computer Engineering, University of Washington, Seattle, WA USA, {alewi, kyle509, blake}@uw.edu.

A. Munawar and G.S. Fischer are with the Robotics Engineering Program, Worcester Polytechnic Institute, Worcester, MA USA, {amunawar, gfischer}@wpi.edu.

A. Deguet, R.H. Taylor and P. Kazanzides are with the Dept. of Computer Science, Johns Hopkins University, Baltimore, MD USA, {anton.deguet, rht, pkaz}@jhu.edu.

Y. Li is with the Dept. of Electrical, Computer, and Telecommunication Engineering Technology at Rochester Institute of Technology, Rochester, NY USA, Yangming.Li@rit.edu.

This work was supported by NSF National Robotics Initiative Awards IIS-1637789, IIS-1637759, and IIS-1637444.

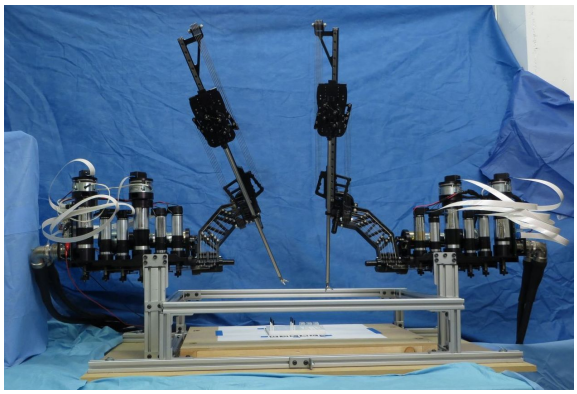

Fig. 1. The Raven-II surgical robot

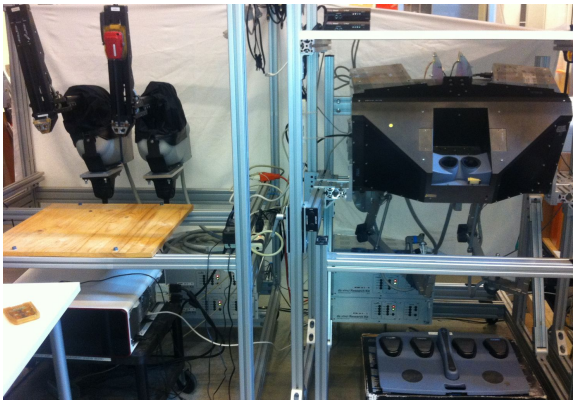

Fig. 2. The da Vinci Research Kit (dVRK)

setups with custom robots or commercially-available industrial robots. Researchers are often faced with the difficult choice between: (1) building a realistic experimental robotic system (e.g., custom robot) that matches the intended application in complexity and task-richness, but is expensive and one-of-a-kind, or (2) selecting a "bench-top" platform (e.g., industrial robot) with less complexity and task-richness, which is easier to develop but reduces the research impact. Several years ago, it became apparent that one obstacle to surgical robotics research was the lack of a robust and realistic common research platform. This led to the development and community adoption of two open research platforms: the Raven-II and the da Vinci Research Kit (dVRK), described in the following sections. Today, these two surgical robot platforms have a combined installed base of about 50 systems in more than 40 research centers (several labs are using both systems) and several of the publications cited above were developed using one of these systems [5], [6], [9], [10]. In addition, simulations of the da Vinci robot have been developed, including commercial products that are used for surgeon training. In the research domain, simulations of the da Vinci include [11] and [12], with the former based on Gazebo and the latter on V-REP. Typically, these simulate the patient-side manipulator (PSM) and utilize a human input device, such as the da Vinci Master Tool Manipulators (MTMs) or haptic devices. In the following, we also summarize the recently developed Asynchronous Multi-Body Framework (AMBF) [13], which supports simulation of Raven II, dVRK and other robots.

1) Raven II: The Raven-II [14], [15] (Fig. 1) was created by the University of Washington and the University of California Santa Cruz in 2012, as an update to the prior Raven surgical robot, and was initially disseminated to 7 institutions. It was designed to deliver the required forces and range of motion from a compact package with state-of-the-art motion control. Experiments have been conducted to demonstrate the portability and durability of the Raven-II system, including a simulated telerobotic surgery in a tent in a remote site, north of Simi Valley, California; a wireless teleoperation with a gasoline-fueled generator to provide power; and transporting and re-assembling the robot at the Aquarius undersea research station at a depth of 19 meters off the coast of Key Largo (as part of NASA's NEEMO program). In late 2013, production of Raven II systems was spun out to a startup, Applied Dexterity Inc., which has installed several additional systems.

2) da Vinci Research Kit (dVRK): Also in 2012, Johns Hopkins University (JHU) and Worcester Polytechnic Institute (WPI) released open source mechatronics and software to enable researchers to create dVRK platforms (Fig. 2) [16], [17] from retired first-generation da Vinci Systems. Specifically, researchers could connect the cables from the da Vinci Master Tool Manipulators (MTMs), Patient Side Manipulators (PSMs) and Endoscopic Camera Manipulator (ECM) to an open source controller consisting of bridge linear amplifiers to drive the motors and field programmable gate arrays (FPGAs) to process the associated sensor feedback and control signals. The FPGAs exchange all data with a control PC via IEEE-1394 (FireWire), achieving closed-loop control rates in excess of 1 kHz.

3) Asynchronous Multi-Body Framework (AMBF) Simulator: To address the needs for a highly flexible simulation framework in terms of robot definitions, support for a broad array of disparate input devices, and interactions with the environment, we recently proposed a simulation framework based on a front-end description format called Asynchronous Multi-Body Framework (AMBF Format) and an associated robust real-time dynamic simulator [13]. The AMBF Format allows for: enhanced human readability and editability, distributed definition of the simulation elements, independent constraint handling, controllability of the way forces are applied to the bodies, communicability of all aspects of every body independent from each other, and dynamic loading with the ability to add bodies and change constraints at run-time. The AMBF simulator was designed around this AMBF format and provides for dynamic simulation of the bodies, flexible visualization options, asynchronous support for a diverse array of input devices used simultaneously without degrading performance, and soft body support. An approach to modeling the dynamic model parameters for the dVRK is presented in [18], and an example of implementing closed-loop kinematic chain mechanisms using this framework, a challenge in many simulation environments, is presented in [11]. More detail about how AMBF has been applied to the dVRK and Raven II is provided in Section III-C.

B. Motivation

While the Raven II and dVRK each provide a shared research platform, it became clear that achieving something closer to the "one design" advocated by [19] would be even better. At first glance, it would appear impossible to create a common hardware and software platform because Raven II and dVRK are based on different hardware designs. In reality, however, many Raven II robots are used to drive the four degree-of-freedom da Vinci instruments, so the part of the robot that interacts with the environment is often the same for both systems. This provided the initial motivation to create a common software interface for Raven II and dVRK, as well as a common surgical tool class, as shown in Fig. 3. It soon became apparent, however, that a common software interface could apply to other robots used by researchers in surgical robotics, so the goal broadened to define a common "language" for component-based robotics software. One specific motivation was to enable research, such as the works cited above, to be easily replicated on other robot platforms.

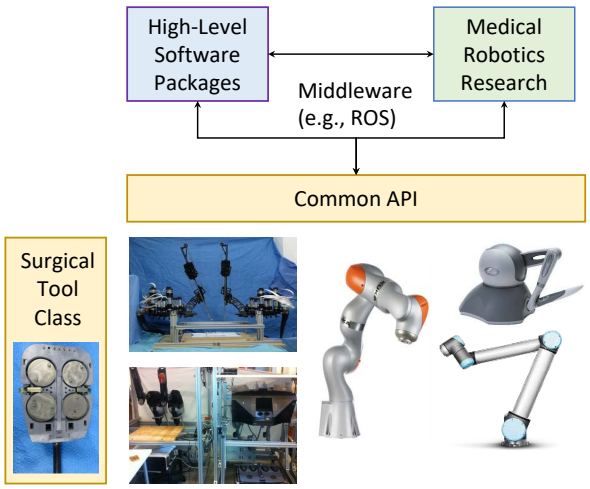

Fig. 3. Concept for Common API to Raven II, dVRK and other systems, and Surgical Tool Class to facilitate sharing of surgical instruments, especially those actuated by the four disks introduced by the da Vinci robot.

We consider two aspects of this common language: (1) the communication infrastructure that passes messages between the components, and (2) the content of the messages. Our work specifically focuses on the second aspect, with abstract (implementation agnostic) definitions of the message content, while making the assumption that one or more existing middleware packages can satisfy the first requirement.

Robot Operating System (ROS) [20] is widely used in robotics and is an obvious candidate for the communication infrastructure. However, while ROS provides a standard middleware (e.g., topics and services) and common message types, there is no well-documented consensus standard on the use of these messages and interfaces. The low barrier to entry provided by ROS topics, and ROS communication structure in general, enables nascent users to quickly get up to speed with ROS based applications. This ease-of-use and the lack of enforcement of a messaging payload standard has provided adequate adoption for ROS to be considered as the "community standard" middleware. However, the lack of a payload standard results in the creation of redundant "wrapper" or "adapter" nodes for connecting independently developed ROS applications. This was recognized early on and efforts were made to develop some unofficial, yet commonly agreed upon, payloads for specific applications. These messaging payload specifications have been adopted widely by the community, especially when used by popular open source packages within ROS. However, as useful these standards are, they do not cater to the complexity associated with the hierarchical control structure of modern robots.

The early releases of both Raven-II and dVRK software provided simple, ad-hoc ROS interfaces that utilized the commonly used message payloads where they made sense and new payloads where required. However, as the software matured and the messaging interface became complex, many of the payloads were revised. Due to the size of the communities, this became less and less feasible as many groups were developing extended applications that relied on the core messaging interface exposed by the ROS packages.

This motivated the design of an interface specification that caters to the complexity of teleoperated, cooperatively-controlled and general purpose robots. We call this proposition the Collaborative Robotics Toolkit (CRTK), emphasizing its origin in robots that work collaboratively with surgeons. The Canonical Robot Command Language (CRCL) [21] is a similar effort that focuses on industrial robots and automatic guided vehicles (AGVs), but is primarily concerned with low-level interfaces to industrial controllers. In contrast, CRTK caters to the hierarchical control structure of robots that includes the high level and low level states/commands.

II. Project Outline and Software Architecture

A. Project Outline

The Collaborative Robotics Toolkit (CRTK) is a community-based software infrastructure developed for research and education in cutting edge human-robot collaborative areas such as semi-autonomous teleoperation and medical robotics. The project development process is composed of two processes: Community Engagement and Technical Implementation. Community engagement includes in-person and online forums for discussion across medical robotics communities around the world. Techincal implementation incorporates those comments and inputs from the community and defines the CRTK structure and hierarchy according to user feedback.

1) Community Engagement: Engagement of the worldwide community of medical robotics researchers and software developers is a major goal of the project. We hosted workshops at IROS 2017 (*Shared Platforms for Medical Robotics Research*), ICRA 2018 (*Supervised Autonomy in Surgical Robotics*), and ISMR 2019 (*Open Platforms for Medical Robotics Research*), as well as a tutorial at IROS 2018 (*Collaborative Robotics Toolkit (CRTK) and Open Platforms for Medical Robotics*

Research). These workshops and tutorials resulted in joint editing of collaborative documents defining a set of use cases, naming conventions, and functionalities. Raven-II and dVRK robots were physically present at some of the events and some of the examples described in Section III were shown during hands-on demonstration sessions.

2) Technical Implementation: Through these community workshops and events, the authors collected community ideas and defined use cases (Section II-B), which were then used to define and modify the CRTK infrastructure. The authors, including both Raven-II and dVRK developers, conducted weekly teleconferences to ensure consistent implementation of the API across both robotic platforms as well as the ROS message payloads including frames, units, API, and namespace usage. Interfacing scripts and example tests were implemented in both Python and C++. During these weekly meetings, implementation status and strategies to tackle device specific technical challenges were discussed.

B. Use Cases

Our collaborative design process identified several medical robotics research use cases on which to base the API development. The authors categorized the use cases into five themes:

1) *Teleoperation*: Support teleoperation across different communication channels, with diverse master and slave devices, and allowing the incorporation of force information through bilateral teleoperation or force reflection.
2) *Autonomous Motion*: Interfaces that enable researchers to incorporate autonomous robot motion planners in both Cartesian and joint space.
3) *Custom Kinematics/Control*: Enable researchers to implement advanced controllers, such as constrained optimization, that simultaneously solve kinematics and control for applications such as optimization of kinematic redundancy or enforcement of virtual fixtures.
4) *Cooperative or Compliant Control*: Provide capabilities to implement custom cooperative or compliant control, such as by attaching a force sensor to the robot wrist and using measured forces to drive robot motion.
5) *Custom Instruments*: Enable researchers to easily integrate custom instruments, providing capabilities such as increased dexterity or additional sensing, with the Raven-II or dVRK.

C. Software Architecture

The goal of CRTK is to provide standard conventions for the command and feedback messages that flow within a robotic system. We consider each message to contain an identifying *name* and associated *payload*. In ROS, the *name* corresponds to the topic or service name and the *payload* is specified in a message description file (e.g., `msg` file). ROS provides tools to parse the message files and generate the software to convert messages to/from data types in the target programming language (e.g., C++ or Python). However, CRTK is not specific to ROS and other middleware, such as OpenIGTLink [22] could be adopted.

The initial work on CRTK focused on the *Robot Motion Interface* and the *Robot State Interface*, which are presented in this section.

1) Robot Motion Interface: The ability to move is arguably the defining characteristic of a robot and an obvious target for any standardization effort. Traditionally, industrial robots were programmed using high-level motion primitives, such as moving in a straight line to a desired pose. While high-level motions are relevant to medical robotics, there are also numerous examples of medical robots that are teleoperated or cooperatively-controlled (as noted in the use cases in Section II-B), which requires a low-level motion interface. For example, a teleoperated robot may require a stream of position or velocity commands from the master to the slave manipulator. Similarly, some implementations of cooperative control use a force sensor mounted on the robot wrist and convert sensed forces to a stream of desired velocity commands (i.e., admittance control). It is also necessary to consider the rate of command streaming, which may affect assumptions on the slave robot regarding motion smoothness and interpolation.

We thus define three levels of motion commands in CRTK, as shown in Fig. 4. Briefly, the `servo` level is intended for high-rate, low-latency, low-level control of the robot. This includes many teleoperation and cooperative control use cases. Generally, the robot should respond as quickly as possible to the `servo` command, preferably after performing some safety checks. The `interpolate` level is similar, except that the rate of setpoints may be slow or unreliable, so the robot should provide a simple interpolation to achieve smooth motion. Finally, the `move` level is for infrequent, high-level motion commands, such as moving to a specified pose. In this case, the robot should include trajectory planning capabilities. All CRTK motion commands must follow the naming convention defined in Table I, which indicates whether the motion is in joint or Cartesian space and what motion parameter is being controlled (e.g., position, velocity or force).

Figure 4 also shows the motion-related information that can be queried from the robot. In addition to the `measured` (sensor) feedback, it is also possible to query the current `setpoint` as well as the ultimate `goal` of the current motion. Note that for the case of a low-level `servo` motion, the goal will be equal to the `setpoint`.

It is important to note that robots are not required to support all types of motion commands but, if a command is implemented, it must follow the naming convention in Table I. In addition, it must also use the prescribed payload (message type). We initially focused on the `servo` interface, so the payloads for those commands are documented on the project website [23]. We are currently working on the definition of the payloads for the `interpolate` and `move` commands.

An interesting observation is that a standardized `servo` interface could be sufficient because the higher levels could be implemented by generic software modules that interact with that level. In fact, it could be sufficient to have only the joint space `servo` commands. This is similar to the approach taken by ROS, which typically interacts with robots at the

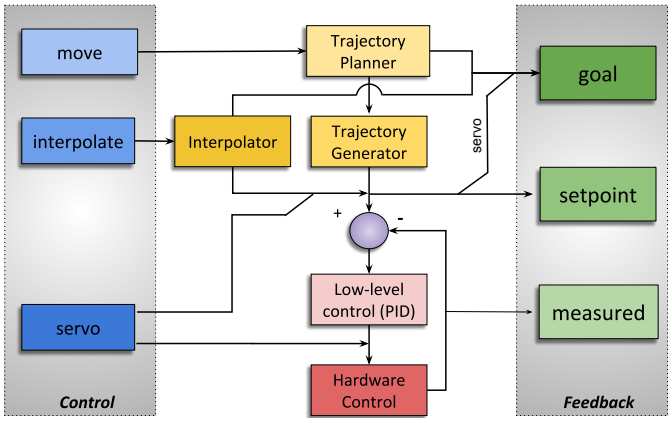

Fig. 4. CRTK motion commands: high-level `move` commands, mid-level `interpolate` commands, and low-level `servo` commands, which move robots in joint or Cartesian space based on various desired quantities, as defined in Table I. The arrows in the diagram represent data flow. Three types of inquiry are supported: `measured`, `setpoint`, and `goal`.

TABLE I
THE NAMING CONVENTION OF THE FEEDBACK AND CONTROL MESSAGES IN THE CRTK COMMON API.

Type	Syntax	Details
Control Level	`servo`	direct real-time stream (pre-emptive)
	`interpolate`	interpolated stream (pre-emptive)
	`move`	plan trajectory to goal (pre-emptive), monitor with `is_busy`
Feedback	`measured`	sensor feedback
	`measuredN`	redundant sensor feedback (N=2, 3...)
	`setpoint`	current setpoint to low-level controller
	`goal`	ultimate goal of current motion
Space	`j`	joint
	`c`	Cartesian
Type	`p`	absolute position or pose
	`r`	relative position or pose
	`v`	velocity or twist
	`f`	generalized force (effort or wrench)
	`s`	joint state (position, velocity and effort)

joint level. The disadvantage of this approach, however, is that it ignores any existing high-level implementations. For example, all industrial robots provide an equivalent `move` command in both Cartesian and joint space. The CRTK approach enables researchers to "wrap" these vendor-supplied capabilities (which may have been optimized for the particular robot) so that they conform to the CRTK naming convention and prescribed message type. A software-based solution can still be used in cases where the high-level functionality may not exist (e.g., custom robots) or when the wrapped vendor-supplied solution is deemed inadequate.

2) Robot State Interface: Although most, if not all, robot systems define operating states, it is impractical to attempt to define a standard state diagram for all robot systems. We therefore focus on defining a high-level "meta-state" diagram that summarizes the operating states as well as the commands to transition between these meta-states, as shown in Fig. 5. These meta-states may correspond to multiple internal states of a particular robot. We also define two operating modes, which can apply to one or more of the meta-states. For example, the `is_homed` operating mode indicates whether the robot

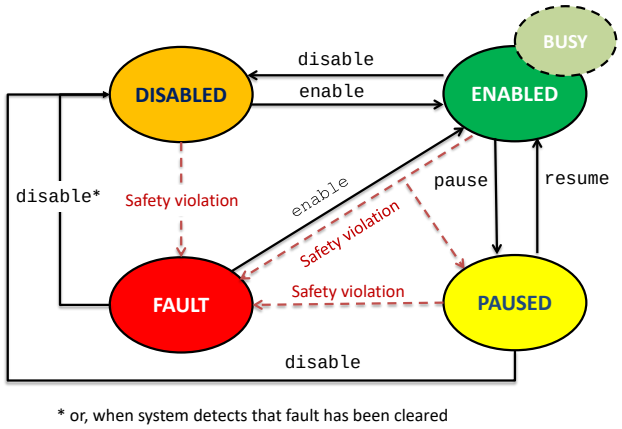

Fig. 5. CRTK robot meta-states and modes. The four meta-state queries are: `is_disabled`, `is_enabled`, `is_paused`, `is_fault`. The two operating mode queries are: `is_busy` and `is_homed` (not shown).

TABLE II
THE LIST OF C++ CLIENT APIS

Type	Package	Description
Utilities	`footkey`	A utility script to alternate between the two CRTK robot states - 'ENABLED' and 'PAUSED'. This is served as a keyboard alternative to a foot pedal.
	`holdpos`	A utility script to hold the robot pose and stay there.
Example	`servo_cube`	An example script to randomly trace the edges of a virtual cube using relative Cartesian CRTK command `servo_cr`.

has been homed and applies to all of the meta-states. In contrast, the `is_busy` operating mode indicates that the robot is currently executing a motion command, which applies only to the `is_enabled` meta-state.

D. Client APIs

In 2018-19, the authors implemented the lowest-level (`servo`) CRTK interface on Raven-II, dVRK, the AMBF simulator and other robots and devices in our labs. In order to lower the learning curve for new users, example interfacing scripts, also called the *Client APIs*, are provided for users to modify or test on their robots.

1) C++ ROS Client API: To demonstrate use of the CRTK interface, the authors developed the `crtk-cpp` repository [24] which consists of (a) a library, (b) examples, (c) utilities and (d) functionality tests. The library contains basic CRTK API robot state and motion helper functions that are used in the rest of the package for better readability and compactness of the code. The examples and utilities are the two types of C++ client APIs. Client Test Scripts will be described in more detail in section II-E.

Utilities are software packages designed to be useful in general robot control scenarios. The authors envision users in the community to directly download and use the utilities as part of their research applications that allow robots to demonstrate CRTK functionalities, but are likely to be modified for future user needs (Table II).

2) Python ROS Client API: The main goal of the Python client API is to provide a CRTK API that allows users to communicate with a ROS CRTK compliant robot. One could directly use the *rospy* package in Python but the learning curve can be steep. The Python client API hides the ROS publishers/ subscribers, converts the payloads to more convenient data types (i.e. *PyKDL* frames and *Numpy* vectors and matrices) and relies on Python thread events to implement blocking commands (for state changes and `move` commands).

Since CRTK devices might implement different subsets of the CRTK specifications and the application might only need some of the CRTK features, the Python client module provides methods to instantiate only parts of the CRTK standard. For example, if one only needs to monitor the operating state and the Cartesian position of a device, one would call `add_operating_state()` and `add_measured_cp()`:

```
import crtk
import PyKDL

# instance of CRTK client
class custom_client:
  # configuration
  def configure(self, namespace):
    # add CRTK features needed
    self.utils = crtk.utils(self, namespace)
    self.utils.add_operating_state()
    self.utils.add_measured_cp()
```

Later on, the user can create an instance of the customized Python client and access the CRTK feature `measured_cp()`:

```
client = custom_client()
client.configure('/dvrk/PSM1')
p = client.measured_cp()
```

The Python client API also provides methods to wait for CRTK state events. For example, the client can wait while the device is busy executing a `move` command using `wait_while_busy`.

The latest version and examples can be found at [25].

E. Client Test Scripts

In parallel to the CRTK API design, we created a set of standardized client test scripts (and written descriptions of expected robot behavior) to demonstrate that a robot system correctly supports the API (Table III). Each test is performed on a per arm basis. The robot namespace is sent as an input when running the test; robot-specific information including the joint types, joint number and home poses are then loaded as ROS parameters. The authors envision all CRTK-compliant robots to be able to execute the same test scripts and use these tests to validate CRTK API adherence.

III. Examples

A. Teleoperation

During the hands-on session of the tutorial at IROS 2018 in Madrid, CRTK `servo_cr` and robot state transitions were

TABLE III
The List of Client Test Scripts

Test	Package	Test Purpose
measured	`measured`	CRTK `measured_js` and `measured_cp` commands.
state	`state`	CRTK robot states and state commands.
servo	`servo_cp`	CRTK absolute Cartesian position command.
	`servo_cr`	CRTK relative Cartesian position command.
	`servo_cv`	CRTK Cartesian velocity servo command.
	`servo_jp`	CRTK absolute joint position servo command.
	`servo_jr`	CRTK relative joint position servo command.
	`servo_jv`	CRTK joint velocity servo command.

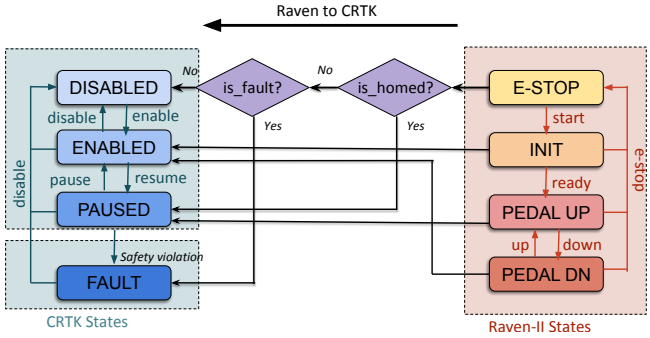

Fig. 6. The correspondence map between Raven-II states and CRTK states.

successfully implemented to allow teleoperation of a Raven-II at the University of Washington in Seattle. The Raven-II robot checks for new CRTK commands through ROS topics at 1000 Hz. As illustrated in Fig. 6, if a state transition command is received, Raven-II automatically maps the desired CRTK states to the internal Raven-II states and proceeds with the state change. When a CRTK motion command is received, in this case `servo_cr`, Raven-II responds as follows:

1) Check if the `servo_cr` incremental Cartesian command falls within a predetermined step size safety threshold. Proceed if true and ignore command otherwise.
2) Reformulate `servo_cr` into `raven_cr` by a change in units and a spatial transformation of the reference coordinates from the CRTK frame to the Raven-II base frame.
3) Add the `raven_cr` command to the desired Raven-II Cartesian pose `raven_cp_d`.
4) Cap `raven_cp_d` if the desired Raven-II pose deviates from the current Raven-II pose by more than a predetermined safety threshold.
5) Execute the motion command.

The CRTK based teleoperation was later demonstrated, using `servo_cp` and the Python client interface, at the workshop at ISMR 2019 in Atlanta, where a Phantom Omni was used to teleoperate a dVRK system at Johns Hopkins University in Baltimore, Maryland. The same code was also demonstrated with a Novint Falcon as the slave arm.

```
# in setup, created two clients: master and slave
scale = 0.25
# record where we started, only positions
start_master = PyKDL.Frame(master.measured_cp())
start_slave = PyKDL.Frame(slave.setpoint_cp())
```

```
# create target goal for slave, use current
  orientation
goal_slave = PyKDL.Frame()

# loop
  # get master measured position
  current_master = \
      PyKDL.Frame(master.measured_cp())
  # compute goal for slave
  goal_slave.p = start_slave.p + scale \
      * (current_master.p - start_master.p)
  goal_slave.M = current_master.M \
      * start_master.M.Inverse() * start_slave.M
  # tell slave to move, then sleep
  slave.servo_cp(goal_slave)
  rospy.sleep(1.0 / rate)
```

B. Image-Guided Surgery

A goal of the CRTK API is that its commands can be readily translated to different surgical robot systems, each with their own unique software architectures. In addition to the teleoperated and cooperatively controlled surgical robots, another class of robots is those that are image-guided. Often these robots are used for placing instruments that may include biopsy and therapy delivery needles, ablation instruments, and electrodes through percutaneous or stereotactic means. In this scenario, typically medical imaging, such as MRI, ultrasound, or CT is used intraoperatively or registered to an intraoperative tracking system to guide the procedure.

To demonstrate the capability of the proposed framework for this class of robot, CRTK command structures were implemented on the WPI NeuroRobot system [26]. This is a seven degree of freedom MRI-compatible stereotactic surgical robot used for interstitial needle-based therapeutic ultrasound for brain tumor ablation, and its use case is representative of the type of robot wherein one or more targets, and optionally an associated trajectory to reach them, are defined in medical imaging and the robot is intended to follow that trajectory to align and insert the instrument, often under real-time imaging which may be integrated into updating the trajectory on the fly. The NeuroRobot system is controlled by a modular MRI-compatible robot controller used to control a number of surgical robots with applications including prostate cancer [27] and neurosurgery [28]. This control system is a self-contained centralized controller that resides inside the MRI scanner room and couples to a robot that resides on the scanner bed inside the bore of the scanner with the patient. Onboard, the system runs a real-time Linux operating system on a National Instruments sbRIO 9561 module, and this system communicates with external devices such as surgical navigation software (e.g., 3D Slicer [29]) over a fiberoptic Ethernet network connection.

This image-guided surgery robot controller uses the Open Network Interface for Image-Guided Therapy (OpenIGTLink) communication interface which provides a standardized mechanism for communication among computers and devices in operating rooms for a wide variety of image-guided therapy (IGT) applications [22], and this this implementation of CRTK is directly translatable to a wide array of devices also using this communication interface. The NeuroRobot controller runs the C++ OpenIGTLink library for two-way communication with external systems. The packet naming convention of the OpenIGTLink interface was modified to use the CRTK notation. This update allows the NeuroRobot to receive a desired Cartesian set-point via the `servo_cp` command and desired joint set-points via the `servo_jp` command. The NeuroRobot can also transmit internal robot parameters using the `measured_cp`, `measured_jp`, `measured_jv`, `desired_cp`, and `desired_jp` commands, as defined by the CRTK specification. This transmit and receive implementation is also compatible with the ROS-OpenIGTLink bridge, allowing for easy ROS-based control.

C. AMBF simulator example

The AMBF Simulator [13] incorporates the physical dVRK MTMs using a plugin based interface for haptic interaction in a dynamic simulation. The plugin is called `dVRK ARM` and can be found at "https://github.com/WPI-AIM/ambf/tree/master/ambf_ros_modules/dvrk_arm". This plugin utilizes the ROS messaging interface exposed by the dVRK software and provides class methods that are modeled after the CRTK specification (Figure 7). Specifically, it implements `servo_jp`, `servo_jf`, `servo_cp`, `servo_cf`, `measured_jp`, `measured_jv`, `measured_cp`, `measured_cf`, `move_jp` and `move_cp`. Since the requirements of the plugin to control and sense a simulated environment included different types of feedback data and control modes, the use of the hierarchical controller structure specification proposed by CRTK simplified both the design and implementation of the plugin.

Another CRTK example was demonstrated by implementing a controller for a simulated Raven-II in AMBF. The code was written by Yun-Hsuan Su and can be found at [30]. This code contains Raven-II kinematics calculation and support for various control modes including homing, sinusoidal motions and a virtual 3-dimensional cube tracing, which are also available in the physical Raven-II system. These modes can be selected simply by pressing pre-specified keyboard shortcuts.

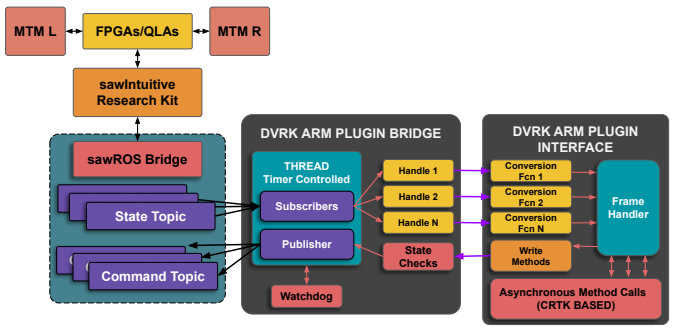

Fig. 7. The design of the `dVRK ARM` plugin based interface that is used by the AMBF Simulator. The block labeled "sawROS Bridge" provides the ROS Based interface. The sub-block labeled "Asynchronous Method Calls (CRTK BASED)" wraps the CRTK specification for state and command data for the method based interface.

IV. Conclusion

This work targets the design and implementation of the Collaborative Robotics Toolkit (CRTK), a common robot command and feedback interface suitable for complicated teleoperation and cooperative control tasks at various control levels. The Raven-II and dVRK software and the AMBF simulator have been updated to support CRTK. A set of example client API and test codes can be downloaded at [31]. We also hosted workshops and tutorial sessions at various international robotics conferences in the past years to introduce CRTK to the community, collect user feedback, and promote community adoption. Meanwhile, the design details, user guide and supplemental documentations can all be found at [23]. In future work, we hope to continue to expand the user community and improve usability of the CRTK infrastructure.

References

[1] G. Guthart and J. Salisbury, "The Intuitive^{TM} telesurgery system: Overview and application," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2000, pp. 618–621.

[2] J. Marescaux, J. Leroy, M. Gagner, F. Rubino, D. Mutter, M. Vix, S. E. Butner, and M. K. Smith, "Transatlantic robot-assisted telesurgery," *Nature*, vol. 413, no. 6854, p. 379, 2001.

[3] A. Shademan, R. Decker, J. Opfermann, S. Leonard, A. Krieger, and P. C. W. Kim, "Supervised autonomous robotic soft tissue surgery," *Science Translational Medicine*, vol. 8, May 2016.

[4] K. Bumm, J. Wurm, J. Rachinger, T. Dannenmann, C. Bohr, R. Fahlbusch, H. Iro, and C. Nimsky, "An automated robotic approach with redundant navigation for minimal invasive extended transsphenoidal skull base surgery," *Minimally Invasive Neurosurgery*, vol. 48, pp. 159–64, July 2005.

[5] S. Sen, A. Garg, D. V. Gealy, S. McKinley, Y. Jen, and K. Goldberg, "Automating multi-throw multilateral surgical suturing with a mechanical needle guide and sequential convex optimization," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 4178–4185.

[6] A. Murali, S. Sen, B. Kehoe, A. Garg, S. McFarland, S. Patil, W. D. Boyd, S. Lim, P. Abbeel, and K. Goldberg, "Learning by observation for surgical subtasks: Multilateral cutting of 3d viscoelastic and 2d orthotropic tissue phantoms," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 1202–1209.

[7] A. Krupa, J. Gangloff, M. de Mathelin, C. Doignon, G. Morel, L. Soler, J. Leroy, and J. Marescaux, "Autonomous retrieval and positioning of surgical instruments in robotized laparoscopic surgery using visual servoing and laser pointers," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, vol. 4. IEEE, 2002, pp. 3769–3774.

[8] R. Bauernschmitt, E. U. Schirmbeck, A. Knoll, H. Mayer, I. Nagy, N. Wessel, S. Wildhirt, and R. Lange, "Towards robotic heart surgery: Introduction of autonomous procedures into an experimental surgical telemanipulator system," *The Intl. Journal of Medical Robotics and Computer Assisted Surgery*, vol. 1, no. 3, pp. 74–79, 2005.

[9] K. Shamaei, Y. Che, A. Murali, S. Sen, S. Patil, K. Goldberg, and A. M. Okamura, "A paced shared-control teleoperated architecture for supervised automation of multilateral surgical tasks," in *IEEE Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 1434–1439.

[10] T. D. Nagy and T. Haidegger, "An open-source framework for surgical subtask automation," in *ICRA Workshop on Supervised Autonomy in Surgical Robotics*, 2018.

[11] R. A. Gondokaryono, A. Agrawal, A. Munawar, C. J. Nycz, and G. S. Fischer, "An approach to modeling closed-loop kinematic chain mechanisms, applied to simulations of the da Vinci Surgical System," *Acta Polytechnica Hungarica*, vol. 16, no. 8, pp. 2019–2048, 2019.

[12] G. A. Fontanelli, M. Selvaggio, M. Ferro, F. Ficuciello, M. Vendiuelli, and B. Siciliano, "A V-REP simulator for the da Vinci Research Kit robotic platform," *Intl. Conf. on Biomedical Robotics and Biomechatronics*, pp. 1056–1061, 2018.

[13] A. Munawar, Y. Wang, R. Gondokaryono, and G. Fischer, "A real-time dynamic simulator and an associated front-end representation format for simulating complex robots and environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019.

[14] B. Hannaford, J. Rosen, D. W. Friedman, H. King, P. Roan, L. Cheng, D. Glozman, J. Ma, S. N. Kosari, and L. White, "Raven-II: an open platform for surgical robotics research," *IEEE Transactions on Biomedical Engineering*, vol. 60, no. 4, pp. 954–959, 2012.

[15] Y. Li, B. Hannaford, and J. Rosen, "The Raven open surgical robotic platforms: A review and prospect," *Acta Polytechnica Hungarica*, vol. 16, no. 8, 2019.

[16] P. Kazanzides, Z. Chen, A. Deguet, G. S. Fischer, R. H. Taylor, and S. P. DiMaio, "An open-source research kit for the da Vinci® surgical system," in *IEEE Intl. Conf. on Robotics and Auto. (ICRA)*, Hong Kong, China, Jun 2014, pp. 6434–6439.

[17] Z. Chen, A. Deguet, R. H. Taylor, and P. Kazanzides, "Software architecture of the da Vinci Research Kit," in *IEEE Intl. Conf. on Robotic Computing*, Taichung, Taiwan, April 2017.

[18] Y. Wang, R. Gondokaryono, A. Munawar, and G. S. Fischer, "A convex optimization-based dynamic model identification package for the da Vinci Research Kit," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3657–3664, Oct 2019.

[19] E. Messina, "Your mileage may vary," *Science Robotics*, vol. 4, no. 35, p. eaay6004, 2019.

[20] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[21] F. Proctor, S. Balakirsky, Z. Kootbally, T. Kramer, C. Schlenoff, and W. Shackleford, "The Canonical Robot Command Language (CRCL)," *Industrial Robot: An International Journal*, vol. 43, no. 5, pp. 495–502, 2016.

[22] J. Tokuda, G. S. Fischer, X. Papademetris, Z. Yaniv, L. Ibanez, P. Cheng, H. Liu, J. Blevins, J. Arata, A. J. Golby, T. Kapur, S. Pieper, E. C. Burdette, G. Fichtinger, C. M. Tempany, and N. Hata, "OpenIGTLink: an open network protocol for image-guided therapy environment," *Intl. J. of Medical Robotics and Computer Assisted Surgery*, vol. 5, no. 4, pp. 423–434, 2009.

[23] *CRTK Documentation GitHub URL*, 2019, https://github.com/collaborative-robotics/documentation/wiki.

[24] *CRTK-cpp GitHub URL*, 2019, https://github.com/collaborative-robotics/crtk-cpp.

[25] *CRTK Python Client GitHub URL*, 2019, https://github.com/collaborative-robotics/crtk-python-client/.

[26] N. A. Patel, G. Li, W. Shang, M. Wartenberg, T. Heffter, E. C. Burdette, I. Iordachita, J. Tokuda, N. Hata, C. M. Tempany, and G. S. Fischer, "System integration and preliminary clinical evaluation of a robotic system for MRI-guided transperineal prostate biopsy," *Journal of Medical Robotics Research*, vol. 04, no. 02, p. 1950001, 2019. [Online]. Available: https://doi.org/10.1142/S2424905X19500016

[27] M. Wartenberg, J. Schornak, K. Gandomi, P. Carvalho, C. Nycz, N. Patel, I. Iordachita, C. Tempany, N. Hata, J. Tokuda, and G. S. Fischer, "Closed-loop active compensation for needle deflection and target shift during cooperatively controlled robotic needle insertion," *Annals of Biomedical Engineering*, vol. 46, no. 10, pp. 1582–1594, Oct 2018. [Online]. Available: https://doi.org/10.1007/s10439-018-2070-2

[28] J. MacDonell, N. Patel, G. Fischer, E. C. Burdette, J. Qian, V. Chumbalkar, G. Ghoshal, T. Heffter, E. Williams, M. Gounis, R. King, J. Thibodeau, G. Bogdanov, O. W. Brooks, E. Langan, R. Hwang, and J. G. Pilitsis, "Robotic Assisted MRI-Guided Interventional Interstitial MR-Guided Focused Ultrasound Ablation in a Swine Model," *Neurosurgery*, vol. 84, no. 5, pp. 1138–1148, 06 2018. [Online]. Available: https://doi.org/10.1093/neuros/nyy266

[29] R. Kikinis, S. D. Pieper, and K. G. Vosburgh, *3D Slicer: A Platform for Subject-Specific Image Analysis, Visualization, and Clinical Support*. New York, NY: Springer New York, 2014, pp. 277–289. [Online]. Available: https://doi.org/10.1007/978-1-4614-7657-3_19

[30] *AMBF Raven Controller GitHub URL*, 2019, https://github.com/WPI-AIM/ambf/tree/master/ambf_controller/.

[31] *CRTK GitHub URL*, 2019, https://github.com/collaborative-robotics/.