Special Issue: Robotics Collaborative Technology Alliance (RCTA) Program

Survey Article

Robotics CTA: 10 Years of Integration, Assessment, and Lessons Learned

Long Quang[2], Marshal Childers[1], Jonathan Fink[1], Mark Gonzalez[2], Arnon Hurwitz[1], Matthew Kaplan[1], Craig Lennon[1], Dilip Patel[2], Jaymit Patel[2], Jason Pusey[1], Trevor Rocks[1], Ed Weller[2] and Chad Kessens[1]
[1]DEVCOM Army Research Laboratory, 2800 Powder Mill Rd, Adelphi, MD 20783
[2]General Dynamics Land Systems, 1231 Tech Ct, Westminster, MD 21157

Abstract: Over the past decade, robotics technologies and the tools used to develop them have undergone significant advancement and transformation. In this paper, we observe and assess these changes from the perspective of a 10-year research program sponsored by the DEVCOM Army Research Laboratory, named the Robotics Collaborative Technology Alliance. Beyond advancing the state of the art by conducting research at some of the top academic institutions across the United States, the alliance also worked with top government and industry partners to integrate the research into meaningful experiments and demonstrations with military relevance. This paper assesses and provides insight into the effectiveness of the collaboration tools used by the team, management methods, data collection efforts, and live and virtual experiments. Ultimately, we seek to inform future efforts requiring disparate and distant teams of the potential advantages and challenges of using such tools by providing our lessons learned for how most effectively to work as a team of teams for advancing robotics.

Keywords: military applications, programmatics, integration

Introduction

Robotics technologies have advanced significantly over the past decade, especially for autonomous robot systems operating in complex environments. In an effort to advance the state of robotics with military relevance, the DEVCOM Army Research Laboratory brought together leading academic, government, and industry partners through the Robotics Collaborative Technology Alliance (RCTA). Targeted research investments through the program resulted in breakthroughs in robotic perception, intelligent control, human-machine interfaces, mobility, manipulation, and highly capable human-scale platforms. These research results were integrated together onto common platforms and applied in militarily relevant scenarios, providing a mechanism for research validation and field experimentation, while accelerating traction towards the Army's manned-unmanned teaming goals.

Received: 2 December 2020; revised: 19 June 2022; accepted: 14 July 2022; published: 1 September 2022.

Correspondence: Chad Kessens, DEVCOM Army Research Laboratory, 2800 Powder Mill Rd, Adelphi, MD 20783, Email: chad.c.kessens.civ@army.mil

DOI: https://doi.org/10.55417/fr.2022058

While robotics inherently requires multidisciplinary teaming and collaboration, research is often conducted in a stove-piped fashion. A group with a particular expertise uses baseline capabilities either from a commercial system or the open source community, and extends those baseline capabilities deeply within their particular area of focus. This is largely due to typical funding structures with limited budgets for large-scale teaming. One model that has broken through to facilitate more tightly integrated teams is the issuing of "challenge" problems. Perhaps the most famous of these have been issued by DARPA, including the original DARPA Grand Challenge [McBride et al., 2008], the DARPA Urban Challenge [Buehler et al., 2009], the DARPA ARM Challenge [Hackett et al., 2013], the DARPA Robotics Challenge [Johnson et al., 2015, Atkeson et al., 2015, Haynes et al., 2017], and the DARPA Subterranean (Sub-T) Challenge [Rouček et al., 2019]. The incredible successes of these challenges most recently spurned the new DARPA RACER Challenge.

Beyond DARPA, other communities such as Robocup [Asada et al., 2019] have also invested heavily in the challenge model over the past 2 decades. In particular, Robocup Rescue [Sheh et al., 2016] has long been an important impetus for improving and bringing side-by-side metric quantification of capabilities for the search and rescue community. These competitions have included physical hardware [Takahashi, 2009] as well as simulations [Akin et al., 2013]. The European Land Robot Trail (ELROB) has become one of the most established competition events in Europe [Bruggemann et al., 2016], focused on both military and civilian problems including medical evacuation and explosive ordnance disposal. More recently, industry has also begun to invest in challenges such as the Amazon picking challenge [Eppner et al., 2016, Correll et al., 2016].

The numerous lessons learned from these activities have been enormously helpful to the community in learning how to build teams and integrate tools on robotic platforms for solving real-world problems. However, the teams that compete in these competitions are typically comprised of a small number of institutions, and may produce redundant products and results. Examples of large-scale teams from a large number of institutions performing basic research and integrating it toward common applications are relatively rare, but they have enormous advantages and impact. The EURON (European Robotics Research Network) (www.euron.org) endeavor, the Naval Intelligent Autonomy program, and the RCTA are three examples.

The EURON program is an ongoing network of collaborating research entities. It has an overall objective of ensuring that adequate resources and mechanisms are available "to enable Europe to be a leader in robotics." To this end, it has established a Benchmark system to guide the contracted research. Benchmarks, here, are set up to focus on particular subdomains (e.g., grasping), and specify the "quality" (i.e., efficacy of serving a purpose) of results. The EURON Benchmarking effort is also directed toward obtaining replicable results from the experimental programs [Bonsignorio and del Pobil, 2015].

Naval Intelligent Autonomy was a 6-year program "to develop, mature, and evaluate autonomy and human interface technologies through a series of phased demonstrations." It was organized for the U.S. Navy by the Office of Naval Research to focus on research into autonomy for unmanned naval vehicles [Steinberg, 2006]. The program allowed for experimentation with different technologies provided by major defense contractors, small businesses, a research laboratory, and several universities. These technologies were integrated with government-owned assets in government laboratories and test ranges such as NAVAIR. Early on, the program developed a consensus functional architecture. The architecture was built around a publish/subscribe component on the control station and on each vehicle. These components handled most of the data management; in addition, components were designed to be as modular as possible. Aspects of this modularity were as follows: (1) Common data interfaces were developed based on the widely used XML Schema. (2) Open Source software was used for the geographic display. (3) Standards were used to the greatest extent possible. Lessons Learned in this program are presented in [Moulds et al., 2008] and [Steinberg, 2008].

This paper adds to our understanding of multi-institutional teaming in robotics, examining the RCTA's recent integration and experimentation efforts at a high level. In particular, we focus here on the Integrated Research task, which was extremely challenging and time-consuming for a large and evolving consortium such as the RCTA. The motivation for this paper is to document lessons

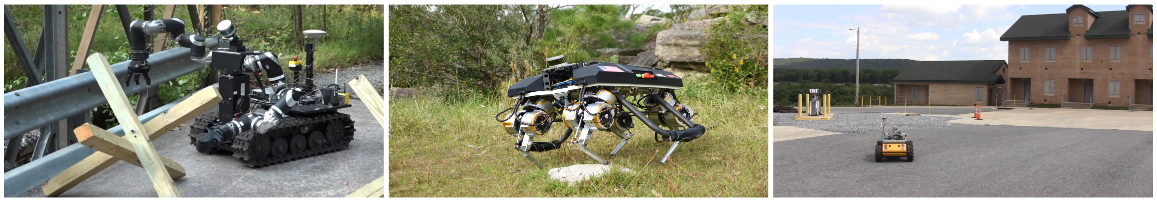

Figure 1. Left: RoMan clearing Czech hedgehog debris. Center: LLAMA taking a step over a boulder. Right: Husky navigating to a gas pump.

learned as well as helpful tools and techniques, with the goal of accelerating the development of integration planning for future similar programs.

Organization

The RCTA program plan supported 4 fundamental research areas: Perception, Intelligence, Human-Robot Interaction (HRI), and Dexterous Manipulation, and Unique Mobility (DMUM). Each of these areas was funded to perform basic and applied research to address Army-recognized capability gaps. A fifth area of applied research was later added to bring the other 4 research areas together and facilitate integrated experiments and demonstrations relevant to Army field applications. This fifth area, Integrated Research, is the main focus of this paper.

The Integrated Research (IR) team was responsible for designing experiments, building full stack platforms to facilitate experiments and demonstrations, developing and maintaining laboratory and field infrastructure, collecting data, and working closely with researchers to ensure the technologies provided were ready for experiments. The IR team included members from each of the government, academic, and industry partners. Integration tasks were performed at a variety of locations, including member facilities as well as notable Army-relevant field locations.

The first 5 years of the RCTA program focused on advancing the state of the art in the 4 individual research areas. As a result, the early integrated experiments were brittle, and progress was limited. The second half of the program benefited from realigned thrusts based on functional capabilities, which required intense collaboration of the 4 research areas to demonstrate Army-relevant field capabilities. In addition, as this paper summarizes, the program successfully developed human-scale robotic platforms that made it possible to demonstrate and communicate the viability of robots as teammates operating in unstructured environments. Some examples of these platforms are shown in Figure 1.

Lessons Learned

While the RCTA program had some unique challenges, the lessons learned presented here reflect common contemporary robotics development challenges and will be widely applicable. The lessons learned are categorized into Software Infrastructure, Hardware Infrastructure, Collaborative Integration, Experiment Infrastructure, Data Collection Infrastructure, and Demonstration Infrastructure. Each lesson listed below indicates the general category that it falls under in parentheses following the title. The categorization of the lessons learned is provided here for ease of manageable future planning and budgeting effort; however, there is obvious overlap between the categories.

Lesson 1—Adopt a common software repository early in the program

Software collaboration was especially challenging throughout the RCTA program. Early in the program, global coding standards were loosely defined, and documentation formats were not standardized. Although some performers did manage their own repositories for their capability packages, that was not the case for the majority of collaborators. For example, source code and documentation files were sometimes distributed via email or external media. This led to poor version control and dependency mismanagement, where few people understood how code was pieced together and actually operated on the robots. A common software repository, Bitbucket for RCTA, quickly remedied these problems by forcing collaborators to adhere to a common workflow, and allowed for proper review of contributed code. In effect, fewer bugs were introduced during integration, which reduced delays and problems during experimentation.

Lesson 2—Support Docker-based development and deployment

The RCTA used Docker to containerize the development workspace for users. Given the complicated web of dependencies required by the RCTA software stack, the installation process could often cause problems with other projects or systems installed on the computer being used. By using a common Docker file to create an image and work in a container, the RCTA environment and dependencies were isolated from the rest of the machine. This also simplified troubleshooting and development because all users could compile and run the system on virtually the same container. Errors stemming from personalized systems or customized tools for other projects were avoided using the Docker containerization. Docker was not used on machines such as the onboard Autonomy Modules, as there were no conflicting projects or software where the RCTA Stack was the primary system function.

Lesson 3—Adopt and maintain a comprehensive software design diagram

A comprehensive design view or system diagram of the program supported staff transitions throughout the life of the program. This identified common terms, functional diagrams, standard functions, software build processes, and test methods. We recommend the following considerations:

- Vertical integration: understand and standardize connections to higher and lower levels from human to actuator or sensor. User-to-software, user-to-hardware, and software-to-hardware interfaces are often designed without regard to types outside their own. For example, be prepared to bridge different standards such as Rest API for user interfaces, Robot Operating System (ROS) for autonomous functions, and EtherCAT for hardware.
- Horizontal integration: generalize connections between levels to allow reuse and substitution of components or research, facilitating flexibility. Standardized ROS messages for packages performing similar functions allows for easily swapping out functions. This requires ensuring that existing messages can carry all of the data being generated and maintaining compatibility when messages are modified. This allowed us to compare software for things like object detection and localization, arm motion planners, and navigation planners. By collecting data from perception algorithms to a single world model, any number of behaviors could be executed on an object in the world model. For example, an object detector could add the pose of an object to the world model, and the robot could then navigate to that pose or attempt to manipulate the object at that pose.
- Work bottom-up for platform stability: Hardware failures typically happen more frequently as high-level tasks become more complicated. Thus, to improve stability, work bottom-up as it is easier to lock down low-level components and then explore high-level options rather than vice versa. For example, many hardware failures such as overheating actuators were alleviated by setting velocity and torque limits at the firmware level.
- To have a successful testbed platform, it is important to balance stability versus flexibility. Consider use-case implications by not getting stuck assembling building blocks, but instead make sure the overall system is capable of all tasks if the program has multiple uses of the same platform.

Lesson 4—Provide and enforce coding frameworks early

Software frameworks present many benefits in collaboration, such as providing a collection of libraries, configuration standards, code reusability, and quick debugging. The RCTA developed a software framework referred to as “RFrame” which provided useful functions and lifecycle for execution on real-time systems. Unfortunately, RFrame was not widely adopted due to its complexity and late introduction. In hindsight, we needed to provide and enforce coding frameworks (loose standards) to all team members at the beginning of the program. Since the ROS was already popular among the open-source community, it was a logical communications framework choice. RCTA software development utilized ROS 1.0 configuration techniques and development following ROS Enhancement Proposals (REPs) (i.e., launch files, catkin workspace, catkin-tools, dynamic reconfigure, etc.). Non-ROS components were easily wrapped with ROS interfaces. Ideally, different levels of frameworks would be recommended:

- At the base level, agree on a communication framework (e.g., ROS) and start an interface design description (IDD) for each pipeline in the stack. The pipeline messages will evolve over time to suit the needs of future work as opposed to gluing together ad hoc pieces with band-aids for communication. The latter adds latency and produces unreliable output.
- At the next layer, agree on lifecycle frameworks (e.g., RFrame) and require it for critical pieces. It is important for critical components of the system to work together as a whole rather than as multiple nodes to start up. As nodes and modules start, pause, or terminate, introducing a life cycle that is common across the modules prevents unexpected logical errors such as race conditions, undesired consumption of valuable system resources, and/or data loss. This intrinsically produces optimized code that is reliable and reusable. As a result, robot operation may also see an improvement in robustness.
- Nodes or modules that are noncritical should be modular and not system-specific. For example, abstract away high-level, hardware-specific parameter, configurations.

Lesson 5—Simultaneously support different Linux distributions

As the RCTA software stack used the ROS running in an Ubuntu Linux environment, the stack was tied to the Ubuntu update cycle. Ubuntu releases new Long Term Support Operating System versions every 2 years, and ROS versions are released on a similar schedule to support the latest version of Ubuntu. The RCTA's integrated software stack began with Ubuntu 14.04 Trusty Tahr and ROS Jade Turtle. Eventually, to keep up with the latest open-source software and community development, the RCTA moved to primarily support Ubuntu 16.04 Xenial Xerus and ROS Kinetic Kame, while maintaining support for Ubuntu 14.04. This allowed us to continue to use software developed for 14.04, conduct a slow transition between OS versions to avoid upgrade-related errors, and support hardware on the RoMan platform (see Figure 1, left), which was only compatible with 14.04. This presented a number of problems concerning cross-compatible software maintenance and testing. All critical testing was doubled to ensure that updates would not break components on either system. The extra testing represented an increased workload, but it was essential to maintain functionality on both operating systems. If an external dependency ended support for 14.04, the integration team needed to find a solution to maintain functionality. Certain advantages of the latest ROS and Ubuntu systems could not be fully utilized due to the required backwards compatibility. However, this approach reduced technical debt required through a forced upgrade of the system. Hardware components of the RoMan platform (which did not support 16.04) could be used without a major redesign, and the process allowed for a gradual refactor of code between OS versions, as opposed to a single major overhaul happening at once. With platforms distributed to collaborators across the country, the downtime required to fully upgrade the operating system of all systems would have been significant. While this approach was not without flaws, overall it reduced technical debt and platform downtime while maintaining a system able to utilize the latest developments from the ROS community.

Lesson 6—Use simulation as a project standard early in the program

Leveraging simulation, participants from remote locations can reproduce end-to-end systems prior to field events. The RCTA initially developed the Robotic Interactive Visualization and Experimentation Technology (RIVET) for robot simulation [Fields et al., 2016] and human-in-the-loop (HITL) experimentation [Schaefer and Brewer, 2016]. RIVET was designed to support rapid, consecutive, functional, and regression testing, as well as to consider multiple different testing scenarios. However, RIVET was not easily accessible and was eventually superseded by open-source simulators for robotics development, such as Gazebo and Unity. Early in the program, we focused heavily on research integration on the actual hardware. This was a burden on integrators as it presented challenges when dealing with many different hardware platforms. Consequently, it delayed important discussions among integrators about component interfaces and pipelines. In addition, many robot configuration decisions (e.g., sensor placement) could have been finalized ahead of hardware development by experimenting in simulation. Running the software stack inside a simulated environment also would have helped shake out potential bugs before heading out to the field. Simulations can verify the performance of the algorithms, which sometimes does not translate in a real-world testing. For example, there is usually noise in our inputs, which sometimes degrades performance and causes unexpected errors. Often times, our simulations are running on different computing platforms and require excess overhead processing. A synergy of testing against recorded real-world data and simulation may help bridge this gap. For researchers, simulation or emulation may provide isolation from the rest of the system.

Lesson 7—Assess reliability before capability/Work bottom-up for platform stability
A majority of the RCTA research efforts demanded a mobile robotic platform to conduct experiments and execute demonstrations. Hardware reliability was essential to ensure consistent performance. We found that as a platform's complexity and experimental scope/demand increased, the reliability of the platform's performance decreased. For example, the basic 4-wheeled platforms used throughout the program (e.g., Clearpath Husky) suffered relatively few hardware issues, as their use was primarily towards low-risk perception and intelligence-based research activities. As we introduced a tracked, dual-limbed platform called RoMan to perform perception, intelligence, and manipulation and mobility activities, we encountered many instances of hardware failure or malfunction, requiring much down-time for maintenance and upgrades. The intended use of the limbed platform was to directly interact with the environment by grasping, lifting, and moving objects to progress manipulation research. This frequent interaction with the environment coupled with the two 7-DOF (degree of freedom) limbs created a high-risk combination leading to critical failures and required maintenance. The test-first approach should rely heavily on a backup supply of hardware components until software or procedural controls become mature enough to limit self-inflicting damages. The most complex platform developed by the RCTA program, a 4-legged state-of-the-art dynamic platform called LLAMA (for Legged Locomotion and Movement Adaptation) [Nicholson et al., 2020], heavily supported legged mobility research. Its direct interaction with the environment to navigate, coupled with pushing the boundaries of legged robotic research, required regular maintenance and redesign to improve reliability and performance.

We strongly recommend using as simple a platform as necessary to execute given research tasks. By keeping the maintenance demand low and reliability high, researchers can devote more time to experiments and data collection. When research is not focused on hardware development, purchasing commercial platforms that meet the necessary criteria to conduct experiments will reduce the engineering and maintenance burden. If existing platforms do not meet your criteria or more complex systems are required, ruggedization of the platform and all Commercial Off The Shelf (COTS) hardware should be a top priority to ensure sufficient reliability.

Lesson 8—Navigating hardware updates requires extensive resources
Through experience, 3 main factors were found to drive the need for robotic platform upgrades: 1) technology obsolescence, 2) redirecting research focus, and 3) unforeseen issues. Obsolescence is a common trait for robotic platforms destined for research, as the performance of the equipped payload (e.g., sensors, cameras, computers, batteries, etc.) is often scrutinized against the "latest and greatest." This scrutiny typically restarts conversations to replace components as new products come out. Typical justifications for replacing components include significant improvement in component performance, the availability of desired features for a newer version of an existing component, or the manufacturer of an existing component no longer providing service or support due to obsolescence or closure. To adapt to an ever-evolving technological landscape without introducing significant reductions in platform "up" time, a compromise must be made between the researchers'

desires to work with the cutting edge of technology and the integrators' desire to ensure a robotic platform that can successfully execute the required tasks, regardless of what suite of components is used.

Changes in research focus, whether driven internally by researchers' interests and discoveries, or by sponsors, investors, or customers, are often a desired change influenced by needs, but consequently to the detriment of platform designers. The frustration comes from the levels of interdependency generated when designing a robotic system. It is common, especially in legged robotic systems, for performance criteria to have cyclical dependencies that cannot be decoupled and are sensitive to deviations from an optimized solution. Thus, changes in focus that affect a heavily constrained platform often lead to changes in architecture at the lowest levels when they no longer fit the initial engineering constraints, and delay delivery of an upgraded platform for researchers.

Unforeseen issues often arise in experimental settings, either as unplanned environmental factors, unplanned behavioral factors, or engineering oversights. A system experiencing difficulty operating within an environment may require hardening to protect the critical systems of the platform. As new capabilities are explored, observed limits in behavior (e.g., terrain navigation, manipulation work space, sensor field of view, data bandwidth, etc.) can drive necessary changes to pursue behavioral development. Engineering oversight is common in an environment where a compromise is made between exhausting all possible factors that affect a system's performance and producing a functional platform within a timely and cost-effective manner.

Introducing a structured system that systematically simplifies the decision-making process and minimizes cognitive bias can generate consistent and reliable decisions that favor the customer's needs and justifies the required work for the engineers and technicians. Additionally, this process generates documentation that can later be referenced for all future decision-making regarding the robotic platform hardware. One such system is a Pugh analysis, which is intended for comparing new design concepts (and, in some cases, component selection or comparison) to a baseline (or current) design. A consensus of criteria based on the needs of the customer or user is considered against each design concept to identify which best satisfies the overall needs and justify the necessary changes.

As a means to streamline the process of introducing new components to a robotic platform and troubleshoot platform performance, all of the major components of a robotic platform should be accessible on a "flat testbed." As platforms become more compact and complex, accessing all of the components becomes troublesome, especially when isolating issues. A flat testbed allows users to reproduce conditions and identify performance issues more efficiently. Additionally, a flat testbed can allow users to explore state-of-the-art devices and products without disrupting operations on an existing platform.

Lesson 9—Compartmentalize subsystems to simplify routine maintenance and upgrades

Finding an opportunity to bring a platform entirely down for upgrades can be difficult if long-term or trial testing is in progress. Conducting in-use maintenance and upgrades was ideal whenever possible to maximize "uptime" on the platform for testing, and provided an excellent recovery strategy when turnaround time between experiments or demonstrations was low and issues arose.

To minimize platform downtime, compartmentalize parts of the system and minimize connections between systems, both mechanically and electrically, as much as possible. For example, the power system may be partitioned in a way that allows parts of a system to always remain online while onboard batteries are charging, discharging, or being replaced. Compartmentalization and modularity also simplify routine hardware maintenance and facilitate future expansion, which is especially helpful for a long-term research program. The upfront design efforts we chose to make for future-proofing our systems were well worth the time and resources.

Lesson 10—Maintain a common computing architecture

Certain algorithms required special computing using Graphics Processor Units (GPUs) while maintaining the power budget on the robotic platforms. This introduced the motivation for using an ARM architecture for computing efficiency. Specifically, the Nvidia Xavier GPU-accelerated computing platform offered extremely low power, an abundance of shared memory, and Nvidia's CUDA compatible hardware for GPU parallel programming.

Despite the favorable specs for ARM based GPU-accelerated computing, there were many setbacks with this approach. Many of these could have been avoided by dedicating a single team member to maintain the platform with the codebase. In addition, we observed:

- Without a cross-compilation setup, building source code is very slow. The necessary overhead development required may be beneficial if automated builds are also made possible.
- Separate deployment processes for general computing and GPU libraries or executables could be orchestrated using containerization tools such as Docker.
- For scalability to systems with a large number of robots and computers, ensure appropriate infrastructure is in place to facilitate Linux driver updates and SDK maintenance.

Lesson 11—Organize capability focused sprints frequently

To develop capabilities within the RCTA program, integrators and researchers collaborated to address the needs of leadership through demonstration. Two observed downsides with this approach were discovered in hindsight: 1) potential to relinquish expertise, and 2) inconsistent enforcement of coding standards.

A common trait among universities involved in the collaborative research is that many participants are often students or postdoctoral fellows. For the RCTA, many of these researchers became experts in their field of study as well as significant contributors to various capabilities within the RCTA. However, when students or researchers left the program prior to its conclusion, their expertise and insight into existing resources went with them. This posed integration problems and influenced the scope of various capabilities. Long-term staff and career researchers, who remained with an institution throughout the course of the program and became an important source of continuity and expertise, later became on-site integrators. They integrated the algorithms developed by researchers into the software stack for use on real-world robots. The integrators then maintained the software for further research and testing.

Researchers focused on exploring and understanding problems will rightfully use the tools with which they are most familiar. However, custom tools and environments posed problems with scaling collaborative efforts across institutions to demonstrate capabilities using common robotic platforms. Integrators were placed in a tough position near the end of the program, needing to decide if accommodating existing custom software from a researcher would be a time-favorable solution, or if it would be better to reverse-engineer the software to better operate on the system. Accommodating existing software might be better for brief, exploratory testing, but it would not be ideal for a long-term solution. An optimized solution may not be realized, and compatibility limitations among other modules may impact overall performance. Reverse-engineering software has proven to provide integrators with better insight into its functionality, but it comes at the cost of time.

To best leverage the skills of integrators and researchers as well as mitigate the potential loss of knowledge and resources, the following role distinctions should be considered for research collaborations:

1. Researcher teaches Integrator – A Researcher should be encouraged to educate integrators on his or her research and its potential functionality within a robotic system. The intention is to allow researchers to communicate their work and become a reference for research-related issues without the burden of engineering their work into a functional application.

2. Dedicated Integrator – An Integrator should be responsible for collaborating with a specific researcher to build expertise. This allows the integrator to better understand how to integrate the work being done from the beginning, given prior knowledge of the system architecture and desired functionality. It also creates some redundancy in expertise if a researcher leaves the program prior to its completion.

Lesson 12—Frequent collaboration events are critical

The RCTA held weekly status meetings to discuss the objectives and challenges of each performer. These meetings were done via conference call and usually consisted of at least 10 participants. In-person meetings also initiated many large-scale workshops and field activities, including experimentation and data collection at military sites such as Fort Indiantown Gap and Camp Lejeune. Annually, the RCTA would host an in-person program review consisting of meetings between principal investigators and program management. These activities were especially useful for capability integration, as they forced collaboration leading up to the events. Each capability thrust would similarly conduct biweekly status meetings to discuss near-term goals and smaller integration efforts. These meetings were virtual and usually consisted of 5–10 participants. The objectives derived from capability thrust meetings were then executed in smaller groups (usually fewer than 5 participants), either virtually or on-site with a robot. In-person meetings, which typically had robot hardware available, focused heavily on integration, and the low percentages of small group meetings are evidence of the lack of integration time and testing on robots. The simple solution to this would be to conduct more small-scale collaboration events where integration on the robots is more frequent. As an example, consider holding small meetings twice a week (77% of total), medium-sized meetings biweekly (20%), and large meetings quarterly, with 1 annual program review (3%).

Lesson 13—Engage collaboration through periodic surveys

To receive feedback from the entire RCTA program, which consisted of academic principal investigators (PIs), students, postdocs, government researchers, and industry partners, the integration team conducted a survey after the program concluded. We asked questions about the RCTA events they had attended, robotic platforms and hardware provided by the program, tools used on the Robotics Collaboration Environment (RCE), what went well, and where improvements were needed. The survey included ordinal and open-ended questions. The majority of the survey respondents, which comprised about 47% of the total number of consortium members, approved of the use of ROS as a software framework and found the Atlassian Suite tools helpful for collaborative research and development of software. Over 72% of survey respondents also thought that the hardware provided by the RCTA in terms of sensors and platforms was generally helpful for their research. Similarly, 70% indicated that data collected on RCTA platforms helped with development when it was not possible to test online.

Our survey feedback shows that the integration team was able to create a helpful environment to facilitate research and development by providing a collaborative online environment to develop and document research. Providing platforms for hardware testing furthered development and created more robust research, with platforms and real-world datasets to use for experiments and testing. Over 93% of survey respondents also felt that field experiments and demonstrations provided the most valuable experiences. These events allowed for close collaboration, as well as rapid testing and feedback in a real-world environment.

The most negative feedback pertained to virtual experiments and the use of the RCTA Gazebo simulation for autonomy development. Less than 37% of survey respondents participated in virtual experiments. The simulation was not heavily maintained within the RCTA and could not provide a real-world comparable environment for testing. If the program had provided a better simulation environment [Fields et al., 2016], it could have facilitated better offline testing and experimentation. Virtual experiment sessions suffered from the lack of a strong simulation environment as well. Had virtual experiments been more successful, they would have allowed for less logistically challenging experiment collaboration events.

The results from the survey helped the IR team understand the benefits and drawbacks of the decisions made earlier in the program. If these surveys had been exercised more regularly (e.g., at annual program reviews), the team would have been able to negotiate the feedback much quicker.

Lesson 14—Continually ensure progress can be easily transitioned to new personnel

In 2014, as robotics technology was showing early signs of maturity and industrial application viability, tech companies (e.g., Google, Uber, Apple, etc.) started hiring away robotics talent. This had a major impact on the RCTA, as we lost quite a few researchers to this "brain drain." We addressed this by adding universities to our consortium and increasing the bench strengths within existing membership. It is not always possible to plan for a challenge like this. However, in one notable instance, a key developer was so singularly entangled in a module that his departure left the remaining team and future members unable to recover, resulting in significant backtracking. Additional code documentation, reporting, and methods of knowledge transfer were needed to ensure progress stayed on track. Over the course of a long program, personnel attrition is inevitable. Consider periodic "shock-testing" to examine the potential impact of specific personnel loss and determine proactive steps that could facilitate knowledge transfer. In this way, existing and new future teammates can pick up where others might leave off, especially unexpectedly.

Lesson 15—Clearly ascribe functional requirements early to reduce incompatibilities
The coordination of broad investment in many research tasks across technology areas and performers proved challenging when it came time to integrate those technologies for a specific task. This was partly due to a late programmatic change in focus toward a more integrated task demonstration. Observations of the Micro-Autonomous Systems Technology (MAST) Collaborative Technology Alliance (CTA) Capstone helped us understand the need to demonstrate how multiple lines of research could synergize to perform specific, Army priority tasks. However, at that point our basic science investments had been broad to cover a wide range of capabilities, and they were not necessarily designed specifically to support one another.

The development of targeted functional requirements for the various technology areas earlier in the program could have led to capabilities that more directly supported one another. As one example, the search and retrieve task would have likely proved more robust and effective if the robot could have obtained a more precise localization and orientation of the object, especially when the robot was in close proximity to the object. Clear performance bounds designed for the task could have been set had the task been determined sooner. See [Lennon et al., 2017] for additional details.

Lesson 16—Applying formal experimental design saves time by gathering data more efficiently
When using a limited number of "research grade" experimental platforms, it is often impractical to run large numbers of experiments. This can be due both to time and personnel constraints as well as hardware "downtime" resulting from less than ideal robustness. In one particular case, though our experimental design was fairly simple conceptually, the inputs were categorical mixed with noncategorical factors, so a D-optimal design was utilized to get an experimental plan that accommodated all requirements within a reasonable number of runs. This required the availability of experiment design software in the hands of a participant familiar with the use of such a tool. If a standard $3 \times 3 \times 3$ design had been chosen for the 3 inputs factors each at 3 levels, this would have required 27 runs without any desirable run replication. In our case, using a D-optimal design gave a 21-run design with 3 replicated runs. See [Tsai, 2020] for additional details.

Lesson 17—Focus on a small number of variables, and clearly define their bounds early in the process
One important criticism of autonomy is its brittleness, and justifiably so. To address that issue, we originally sought to run experiments across multiple lighting conditions, drive surfaces, obstacle geometries, objects to be grasped, and other variables. The sheer number of dimensions we aimed to account for led to mission creep and spread resources too thin. Pilot studies in advance of formal experiments provided significant value by eliminating experimental conditions for which our solution was underdeveloped. While this may seem obvious, ambition can be a tempting trap, so this lesson is worth a reminder. See [Kessens et al., 2020] for additional details.

Lesson 18—Use nonbinary performance metrics (e.g., execution time, etc.) as opposed to pass/fail
Particularly for longer runs requiring execution of multiple autonomous tasks over several minutes, our initial attempts to quantify experimental results with binary success/fail metrics provided crude

insight into how the experiment went. Switching to continuous metrics, such as hand position error or lid opening angle, improved insight and yielded better results for capturing the performance of tasks. It also improved our ability to debug modules by indicating not only what failed, but how it failed. See [Kessens et al., 2020] for additional details.

Lesson 19—Build a strong data infrastructure that is efficient and scalable
As sensor technology and processing techniques advanced, the amount of data collected by the robot drastically increased. With the entire stack running the capstone setup at the end of the program, the platform would produce roughly 5 GB of data per minute. This generated massive bagfiles, which hampered data storage and sharing. Even reasonably sized bags would easily exceed sharing limitations of email or chat applications. Still, any missed topics could render a data set significantly less useful. Without all sensor data and the robot transform tree, the bagfile data could be confusing to work with, or be marginally useful for certain algorithms.

To reduce data set sizes, we limited the number of recorded messages to only the essentials. Most important was the sensor data. For camera images, data were compressed prior to storage. For other message types where a lower rate topic could be recorded, we reduced the total number of messages captured in the data recording. With these measures, the data captured were significantly reduced, to roughly 1 GB per minute. To facilitate data sharing, the RCTA created a repository with support for Git Large File Storage (LFS) in early 2018. This turned out to be a less than perfect option due to inefficiencies moving large files in Git. In hindsight, we should have used a private file storage server such as Amazon Web Services' Simple Storage Service (AWS S3) from the beginning of the program to maintain all of the data collected through the years. As it was, a majority of developers and collaborators maintained their own data storage on personal computers or their institution's servers for many of the early years of the program. An easy option for data sharing from the start would have facilitated valuable collaboration and created a larger pool of test data for collaborators.

Conclusions

The RCTA program was a great opportunity to field research on physical robots in complex environments. It was also a valuable resource for researchers, engineers, and students to learn and grow the robotics community. While other papers detailed numerous advances in basic research made by the RCTA related to navigation, intelligence, perception, mobility, and manipulation, this paper shared our many lessons learned on the integration and program levels. This included challenges and proposed solutions for software, hardware, collaboration, and experimentation. We have listed these lessons by category below and recommend that they be considered as a checklist for planning, budgeting, and executing robotics projects. Our hope is that sharing these lessons will save future programs valuable time and resources as they plan and execute the work that will bring about the next wave of exciting advancements in field robotics.

Software Infrastructure
Lesson 1—Adopt a common software repository early in the program
Lesson 2—Support Docker-based development and deployment
Lesson 3—Adopt and maintain a comprehensive software design diagram
Lesson 4—Provide and enforce coding frameworks early
Lesson 5—Simultaneously support different Linux distributions
Lesson 6—Use simulation as a project standard early in the program

Hardware Infrastructure
Lesson 7—Assess reliability before capability/Work bottom-up for platform stability
Lesson 8—Navigating hardware updates requires extensive resources
Lesson 9—Compartmentalize subsystems to simplify routine maintenanceand upgrades
Lesson 10—Maintain a common computing architecture

Collaborative Integration
Lesson 11—Organize capability focused sprints frequently
Lesson 12—Frequent collaboration events are critical
Lesson 13—Engage collaboration through periodic surveys
Lesson 14—Continually ensure progress can be easily transitioned to new personnel

Experiment Infrastructure and Assessment
Lesson 15—Clearly ascribe functional requirements early to reduce incompatibilities
Lesson 16—Applying formal experimental design saves time by gathering datamore efficiently
Lesson 17—Focus on a small number of variables, and clearly define their boundsearly in the process
Lesson 18—Use nonbinary performance metrics (e.g., execution time, etc.) asopposed to pass/fail
Lesson 19—Build a strong data infrastructure that is efficient and scalable

Acknowledgments

This research was sponsored by the U.S. DEVCOM Army Research Laboratory (ARL) under Cooperative Agreement W911NF-10-2-0016. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of ARL or the U.S. Government.

ORCID

Long Quang https://orcid.org/0000-0001-8780-7592
Marshal Childers https://orcid.org/0000-0002-8723-6977
Jonathan Fink https://orcid.org/0000-0002-1834-2442
Mark Gonzalez https://orcid.org/0000-0002-7531-5774
Arnon Hurwitz https://orcid.org/0000-0002-6392-6228
Matthew Kaplan https://orcid.org/0000-0002-9625-0682
Craig Lennon https://orcid.org/0000-0002-2229-2919
Dilip Patel https://orcid.org/0000-0002-2719-3936
Jaymit Patel https://orcid.org/0000-0003-0783-9397
Jason Pusey https://orcid.org/0000-0001-9620-6449
Trevor Rocks https://orcid.org/0000-0003-1723-2791
Ed Weller https://orcid.org/0000-0001-9175-7401
Chad Kessens https://orcid.org/0000-0001-6366-0880

References

Akin, H. L., Ito, N., Jacoff, A., Kleiner, A., Pellenz, J., & Visser, A. (2013). Robocup rescue robot and simulation leagues. *AI Magazine*, 34(1):78–78.

Asada, M., Stone, P., Veloso, M., Lee, D., & Nardi, D. (2019). Robocup: A treasure trove of rich diversity for research issues and interdisciplinary connections [tc spotlight]. *IEEE Robotics & Automation Magazine*, 26(3):99–102.

Atkeson, C. G., Babu, B. P. W., Banerjee, N., Berenson, D., Bove, C. P., Cui, X., DeDonato, M., Du, R., Feng, S., Franklin, P., et al. (2015). No falls, no resets: Reliable humanoid behavior in the DARPA robotics challenge. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pp. 623–630. IEEE.

Bonsignorio, F., & del Pobil, A. P. (2015). Toward replicable and measurable robotics research. In *IEEE Robotics & Automation Magazine*. IEEE.

Bruggemann, B., Wildermuth, D., & Schneider, F. E. (2016). Search and retrieval of human casualties in outdoor environments with unmanned ground systems-system overview and lessons learned from elrob 2014. In *Field and Service Robotics*, pp. 533–546. Springer.

Buehler, M., Iagnemma, K., & Singh, S. (2009). *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*, volume 56. Springer.

Correll, N., Bekris, K. E., Berenson, D., Brock, O., Causo, A., Hauser, K., Okada, K., Rodriguez, A., Romano, J. M., & Wurman, P. R. (2016). Analysis and observations from the first amazon picking challenge. *IEEE Transactions on Automation Science and Engineering*, 15(1):172–188.

Eppner, C., Höfer, S., Jonschkowski, R., Martín-Martín, R., Sieverling, A., Wall, V., & Brock, O. (2016). Lessons from the amazon picking challenge: Four aspects of building robotic systems. In *Robotics: Science and Systems*, doi:10.15607/RSS.2016.XII.036.

Fields, M., Brewer, R., Edge, H. L., Pusey, J. L., Weller, E., Patel, D. G., & DiBerardino, C. A. (2016). Simulation tools for robotics research and assessment. In *Unmanned Systems Technology XVIII*, volume 9837, pp. 156–171. SPIE.

Hackett, D., Pippine, J., Watson, A., Sullivan, C., & Pratt, G. (2013). An overview of the DARPA autonomous robotic manipulation (ARM) program. *Journal of the Robotics Society of Japan*, 31(4):326–329.

Haynes, G. C., Stager, D., Stentz, A., Vande Weghe, J. M., Zajac, B., Herman, H., Kelly, A., Meyhofer, E., Anderson, D., Bennington, D., et al. (2017). Developing a robust disaster response robot: Chimp and the robotics challenge. *Journal of Field Robotics*, 34(2):281–304.

Johnson, M., Shrewsbury, B., Bertrand, S., Wu, T., Duran, D., Floyd, M., Abeles, P., Stephen, D., Mertins, N., Lesman, A., et al. (2015). Team IHMC's lessons learned from the DARPA robotics challenge trials. *Journal of Field Robotics*, 32(2):192–208.

Kessens, C. C., Fink, J., Hurwitz, A., Kaplan, M., Osteen, P. R., Rocks, T., Rogers, J., Stump, E., Quang, L., DiBlasi, M., Gonzalez, M., Patel, D., Patel, J., Patel, S., Weiker, M., Bowkett, J., Detry, R., Karumanchi, S., Burdick, J., Matthies, L., Oza, Y., Agarwal, A., Dornbush, A., Likhachev, M., Schmeckpeper, K., Daniilidis, K., Kamat, A., Choudhury, S., Mandalika, A., & Srinivasa, S. (2020). Toward fieldable human-scale mobile manipulation using RoMan. In Pham, T., Solomon, L., & Rainey, K., editors, *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications II*, volume 11413, pp. 418–437. International Society for Optics and Photonics, SPIE.

Lennon, C., Childers, M., Camden, R., Sapronov, L., Mihael, M., Dornbush, A., Weller, E., Fields, M. A., & Lebiere, C. (2017). Assessment of RCTA research. Unmanned Systems Technology XIX, 1019502, SPIE.

McBride, J. R., Ivan, J. C., Rhode, D. S., Rupp, J. D., Rupp, M. Y., Higgins, J. D., Turner, D. D., & Eustice, R. M. (2008). A perspective on emerging automotive safety applications, derived from lessons learned through participation in the DARPA grand challenges. *Journal of Field Robotics*, 25(10):808–840.

Moulds, T., Steinberg, M., & Kaner, S. (2008). Lessons Learned When Leaping from Simulation to Live Operations with Multiple Unmanned Systems. In *Proceedings of AUVSI Unmanned Systems North America*.

Nicholson, J., Jasper, J., Kourchians, A., McCutcheon, G., Austin, M., Gonzalez, M., Pusey, J., Karumanchi, S., Hubicki, C., & Clark, J. (2020). Llama: Design and control of an omnidirectional human mission scale quadrupedal robot. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3951–3958. IEEE.

Rouček, T., Pecka, M., Čížek, P., Petříček, T., Bayer, J., Šalanskỳ, V., Heřt, D., Petrlík, M., Báča, T., Spurnỳ, V., et al. (2019). Darpa subterranean challenge: Multi-robotic exploration of underground environments. In *International Conference on Modelling and Simulation for Autonomous Systems*, pp. 274–290. Springer.

Schaefer, K. E., & Brewer, R. (2016). A guide for developing human-robot interaction experiments in the robotic interactive visualization and experimentation technology (rivet) simulation. Technical report, U.S. Army Research Laboratory Aberdeen Proving Ground United States.

Sheh, R., Schwertfeger, S., & Visser, A. (2016). 16 years of robocup rescue. *KI-Künstliche Intelligenz*, 30(3-4):267–277.

Steinberg, M. (2006). Intelligent Autonomy for Unmanned Naval Vehicles. Proceedings of the SPIE Unmanned Systems Technology Conference, SPIE.

Steinberg, M. (2008). Lessons Learned from Flight, In-Water, and Operator Evaluations of Intelligent Autonomy Technologies. In *Subcommittee E - Flight, Propulsion, and Autonomous Vehicle Control Systems*. ACGS Committee Meeting 101.

Takahashi, T. (2009). Robocup rescue: Challenges and lessons learned. *Multi-Agent Systems: Simulation and Applications*, pp. 423–450.

Tsai, J. O. (2020). NaviGAN: A Generative Approach for Socially Compliant Navigation. International Conference on Robotics and Automation, IEEE.

How to cite this article: Quang, L., Childers, M., Fink, J., Gonzalez, M., Hurwitz, A., Kaplan, M., Lennon, C., Patel, D., Patel, J., Pusey, J., Rocks, T., Weller, E., & Kessens, C. (2022). Robotics CTA: 10 years of integration, assessment, and lessons learned. *Field Robotics, 2*, 1874–1887.