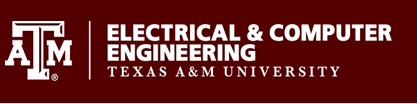
Sony Research Award Program Project Proposal

September, 2020



**PI:** Stavros Kalafatis

**Email:** skalafatis-tamu@tamu.edu

**Phone:** 979-458-8429

**Faculty Innovation Award Theme**

Shared Autonomy for Infrastructure Inspection Robots

**Title**

Developing a Modular Robotic Environment with Shared Autonomy and Resources

**Abstract**

The Multi-robot system has been at the forefront of industrial operation. The study of multi-robot systems has grown significantly in size and importance over the past few years. So far, much more attention has been paid to coordination for the same types of robots. Existing modular robot systems (MRS) mainly focussed on creating formation or structures between robots that are identical in architecture, sensor requirements, computation capabilities, and power requirements. This system might give us an idea about the potential of modular robotics and the future scope but it still needs a lot of refinement for practical usage when the task complexity is too high to be performed by a single type of robot. An MRS introduces robustness that can benefit from data fusion and information sharing among the robots, and fault-tolerance that can benefit from information redundancy. For example, multiple robots can localize themselves more efficiently if they exchange information about their position whenever they sense each other. We can further modify the MRS to make it more modular giving each of the robots further capabilities in terms of computational performance, energy requirement, and mobility within a smart manufacturing or industrial environment. This system will have a universal communication connector that will allow various modules to connect. The resulting system will also be able to communicate to a central cloud server which can be used for shared autonomy among the modular multi-robot units and also for sharing sensor information.

**Background**

Cloud robotics provides an efficient solution for migrating intensive computation from the robot side to the cloud computing infrastructure. The invention of the World Wide Web in the late 1980’s opened up the possibility of connecting a robot to an external machine over the Internet. Networked Robotics has been the classical term for robots connected over a network for functional collaboration. Naturally this domain has seen a lot many paraphrasing with overlapping meanings. (See Fig . 1) In 1994, K. Goldberg was amongst the first to successfully connect a robot to the web, teleoperating the robot through an Internet browser [1]. In 2009, the ‘RoboEarth’ project was announced with the intent to develop a World Wide Web equivalent for robots [2]. The idea was to build a large database that allowed robots to share learned information with one another. Similar to that Cloud Robotics[3] is a platform that moves robot functions implemented on a robot side to the cloud environment. The problem with it was that it was a completely different robotic technology of its own and the programmers need to develop a new skill to make robot services. In [4] the researchers develop their own language ‘CHARON’ for the modular specification of interacting hybrid systems based on the notions of agents and modes. Among the recent advancements in robotics, Smart Cloud[5] tries to mitigate this problem by using a JavaScript-based framework for application development. In 2017, a new protocol called RosLink[6] was proposed to overcome the limitations of ROS over a wide area network. All the research so far has been solely based on a platform having some dependencies on language. Our proposal expands on the current state of the art by developing a system that is platform independent. We will use a ROS master as a cloud server where we can deploy code in any language and can communicate with any robot connected to the server network.

**Overview**

The modular robotic architecture can be traced back to multi-robot systems which are divided into various subparts based on resource sharing, power management, and computational capabilities. Let’s first discuss the basic traditional architecture where each module has embedded intelligent capabilities with an independent onboard controller. The lowest control function detail such as the drivers for the RC servo is closed to the user to improve the programming safety and flexibility. The motion commands can be sent to a certain module individually or broadcast to all modules through the I2C bus according to the task requirements. There is no central control agent in distributed architectures, such that all the robots are equal with respect to control and are completely autonomous in the decision-making process. Since each of the modules is an independent complete unit, we cannot introduce a high variation of tasks assigned to each of the modules. Although such modules are programmable, it is difficult to program the modules separately as the architecture increases. For many applications, creating a monolithic entity that can address all aspects of a problem can be very expensive and complex; instead, creating multiple, more specialized entities that can share the workload offers the possibility of reducing the complexity of the individual entities would have been much helpful.

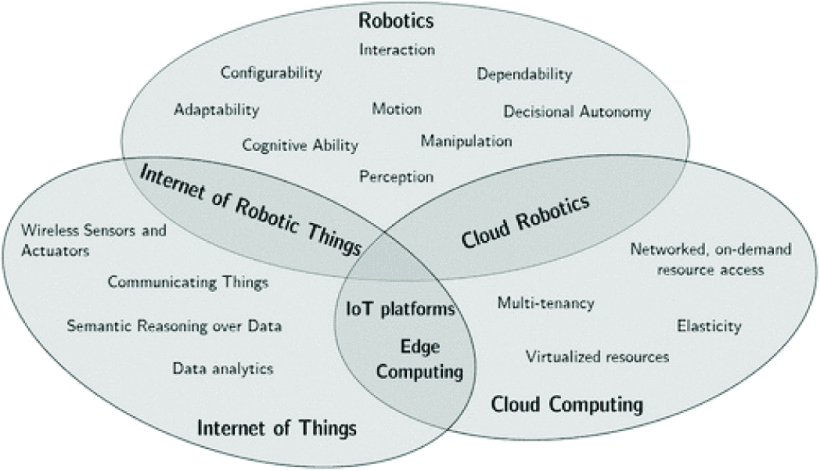


Figure 1: Most common terms around the Internet of Robotic Things concept [7].

Besides, there is a limitation on the amount of computation and power requirements each module can handle. Distributed intelligent systems may require more communication to coordinate all the entities in the system as they must act without complete knowledge of the other entities’ intents. Hence, systems of multiple entities will typically experience increased uncertainty about the state of the system as a whole.

When operating in unstructured or dynamic environments with many different sources of uncertainty, it is very difficult if not impossible to design architecture that will be scalable and reconfigurable. An efficient modular robotic system should have efficient mobility for most of the major parts of the system. It should have an efficient inter-module communication system that has enough bandwidth for information and power-sharing between the modules. Other desirables include fault-tolerant systems and genderless robust connectors with specialized sensors (like cameras or ultrasounds) and linkers. Cost is also a factor. Using several simple robots can be simpler (to program), cheaper (to build) than using a single powerful robot (that is complex and expensive) to accomplish a task.

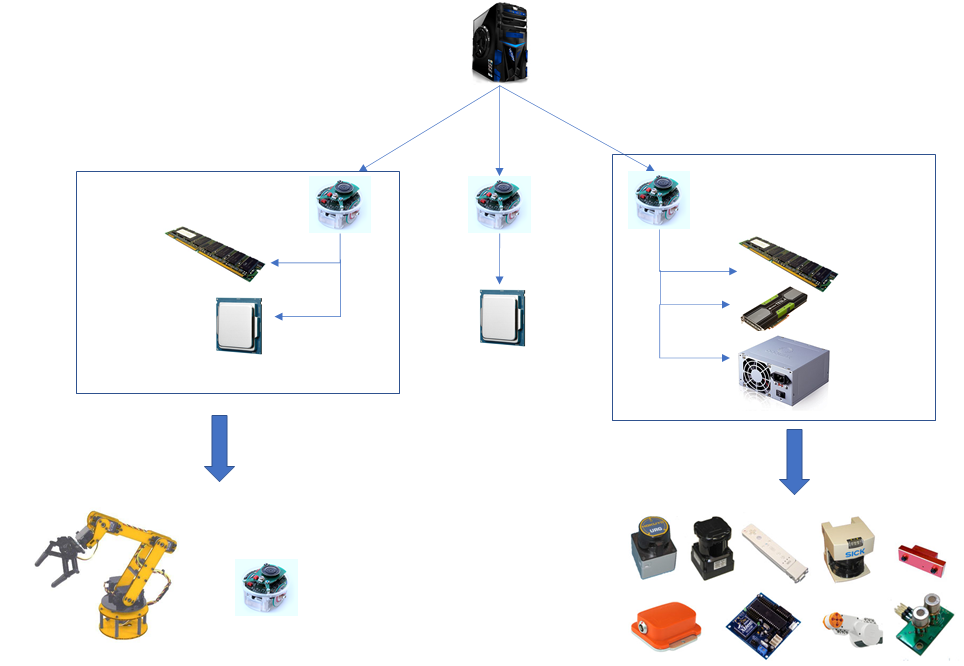


Figure 2: Tentative architecture

A PC can also be directly connected to the bus through a set of wireless data transmission devices. In this way, the PC can be considered as a virtual module in the robotic system and plays the role of the master or a graphic user interface (GUI). Besides, there is a limitation on the amount of computation and power requirements each module can handle.

We are planning to use ROS as a basic backbone for inter-robot communication and resource sharing. ROS is an open-source, meta-operating system for robots. It runs on Linux and Unix-based platforms, so it provides all the operating system services, including hardware abstraction, low level device control, implementation of commonly-used functionality, message-passing between processes, and package management. The main feature of ROS is to support code reuse in robotics research and development. It is a distributed framework of processes that enables executables to be individually designed and loosely coupled at runtime. These processes can be grouped into Packages and Stacks, which can be easily shared and distributed. ROS also supports a federated system of code Repositories that enables collaboration to be distributed as well. To make code written for ROS shareable and integrable with others robots software frameworks, ROS supports any modern programming language, and libraries have clean functional interfaces.

The focus of this project is to implement a system of multi robots or modular subsystems that can be connected to one another based on requirements. See Fig. 2 for the tentative architecture of the interconnected modular devices. Our job will be to test the performance on this architecture with respect to the existing ones in terms of onboard CPU usage, latency and security. We will enable dynamic offloading of computation between cloud and robot based on application memory requirement and criticality. We will compare the latency as well as the volume of data that can be handled simultaneously (throughput). The general idea will be to create microservices for Robots which are independent on their own where each module can act as a microservice. We can scale each microservice separately. We will also explore the concept of reflex action for a module which is like the fallback behavior of a module in case there is a break in connection between the robot within the same network or the cloud server.

**Methods**

In order to create a system of robotic modules for shared autonomy and resources, we divided our project into 3 major subparts: Cloud and software layer, Robot and Module layer, Communication layer. Tailored experiments will be performed to verify the functionality of each of each layer.

Test 1 (Reliability at Startup) [8]: We will set up five local machines to instantiate 10 ROS cores/machine. Multiple ROS cores can work on the same machine by changing the local port and running a dedicated command. We plan to set up a total of 50 ROS cores. A rosbridge server will be instantiated for each ROS core. One simple std\_msgs::String will be sent from a remote machine to each ROS core through the described architecture to test the opening of communication, while a subscriber is already running waiting to receive the message. The communication success rate will be computed via verification of the message sent. Test 1 will be repeated 1000 times, each time using 50 ROS cores at the same time.

Test 2 (Cloud Network): We are planning to use Amazon Web Services (AWS) to host the cloud server. ROS will be installed on the Cloud Server and it will run roscore. Our goal is to support both ROS and non-ROS based robots. We establish a socket connection to the Cloud server from both ROS and non-ROS based robots. We will then develop an interface where the user can see the connected sub-modules in the Cloud network and be able to monitor the data flow across modules by controlling the ROS topics at the server level.

Test 3 (High Rates Communication)[8]: To demonstrate the reliability of communications that involve messages at high frequency, a set of approximately 100,000 std\_msgs::String messages will be sent at 100 Hz (i.e., one message every 0.01 s for 1000 s) from five machines (users) to five different machines (robots), through the described architecture, using the Cloud AWS server as in Test 2. A subscriber will already be up and running, waiting to receive the messages. The metric used to evaluate the reliability of communication will be *data loss*. This is computed as the count of correct messages received over total messages sent. We will be evaluating the communication bandwidth by using iperf [9]. Test 3 will be repeated 10 times, each time using five user–robot pairs. The communication bandwidth will be tested by gradually increasing the data packet size. Finally, we will repeat the same experiment using images followed by video streams. The data resolution will be gradually increased until a threshold is met without data loss.

Test 4 (Security): To test the security of the system, we will use Double Error Detection and Single Error Correction [10] for data integrity check. Every data can be expressed in terms of Hamming codes. In this coding method, the source encodes the message by inserting redundant bits within the message. These redundant bits are extra bits that are generated and inserted at specific positions in the message itself to enable error detection and correction. When the destination receives this message, it performs recalculations to detect errors and find the bit position that has error. We will put appropriate decoders and encoders at every module so that data integrity can be maintained. For further security we will use IPv6 as the internet protocol. We will further try to implement Blockchain[11] technology in the future to secure the modules connected with the same network. Blockchain is a distributed ledger technology (DLT) based on a peer-to-peer (P2P) topology, that allows data to be stored globally on thousands of servers – while letting anyone on the network see everyone else's entries in near real-time. That makes it difficult for one user to gain control of, or game, the network.

Test 5 (Dynamic computation offloading): Based on the number of modules we are using, the central GUI will be able to detect the amount of load each connected module can handle. Connection pooling is a well-known data access pattern, whose main purpose is to reduce the overhead involved in performing database connections and read/write database operations. The computation/storage module which has the least load will get the first connection to any new sensor module that requires some data storage.

**Goals/milestones**

During the course of the project, three (3) quarterly reports and 1 final research analysis and summary support will be provided to Sony. Each of these reports will take place in succession, with one report being delivered upon completion of each of the 4 main milestones outlined below, and the final summary being delivered at the completion of the last milestone. Each milestone has an associated set of tasks that will be successfully implemented before the milestone is seen as being met. As needed, further collaboration between Sony and Texas A&M can be subsequently evaluated. Figure 3 outlines the four described stages. The planned milestones are as follows:

1. Initial framework built
   1. Setup ROS on a cloud server (test key ROS operational features)
   2. Test RosBridge protocol for inter communication between ROS based robots connected within the same network(verify latency, security, bandwidth of connections)
   3. Setup wide area communication between ROS Server and robots through socket connection (verify latency, security, bandwidth of connections)

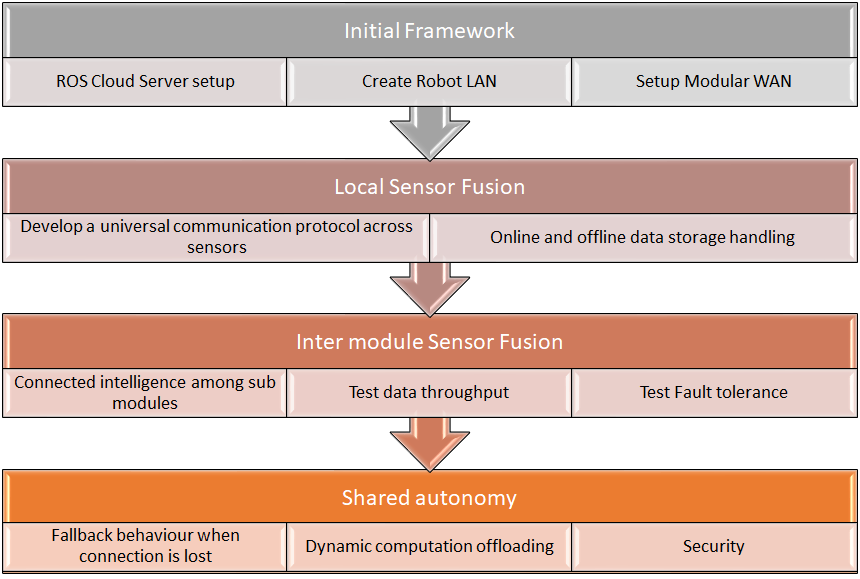


Figure 3: Workflow diagram

1. Cloud layer interaction
   1. Test the IPv6 socket communication between the cloud server and the modules for data sharing (determine packets transmitted vs dropped; evaluate iPV6 compliance)
   2. Storage handling and sensor fusion (determine average storage needs)
   3. Test uploaded computation on server (evaluate speed of operation)
2. Robot Layer interaction
   1. Drone and/or Rover communication with a Remote Joystick (ensure low latency control)
   2. Test data sharing bandwidth within robots (evaluate latency and bandwidth)
   3. Analysis of offloaded test generation/responses
3. Integrate Modular architecture
   1. Test CPU utilization, latency and security of the network
   2. Build and test/verify the architecture is functional via independent tests
   3. Test dynamic offloading of computation between server and robot (evaluate efficiency of task assignment and benefit vs non-offload)

**References**

1. K. Goldberg, M. Mascha, S. Gentner, N. Rothenberg, C. Sutter, and J. Wiegley, “Desktop teleoperation via the world wide web,” in Proceedings of 1995 IEEE International Conference on Robotics and Automation, vol. 1, May 1995, pp. 654–659 vol.1

2. M. Waibel, M. Beetz, J. Civera, R. D’Andrea, J. Elfring, D. Galvez- ´ Lopez, K. H ´ aussermann, R. Janssen, J. M. M. Montiel, A. Perzylo, ¨ B. Schießle, M. Tenorth, O. Zweigle, and R. V. D. Molengraft, “Roboearth,” IEEE Robotics Automation Magazine, vol. 18, no. 2, pp. 69–82, June 2011.

3. J. Kuffner, "What's Next: Cloud-Enabled Humanoids?" in 10th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2010) Workshop, 2010.

4. Fierro, Rafael, et al. "A framework and architecture for multi-robot coordination." The International Journal of Robotics Research 21.10-11 (2002): 977-995.

5. Penmetcha, Manoj, Shyam Sundar Kannan, and Byung-Cheol Min. "Smart Cloud: Scalable Cloud Robotic Architecture for Web-powered Multi-Robot Applications." arXiv preprint arXiv:1912.02927 (2019).

6. Mohanarajah, G., Hunziker, D., D’Andrea, R., Waibel, M. ”Rapyuta: A cloud robotics platform” IEEE Transactions on Automation Science and Engineering, 2015.

7. P. Simoens, M. Dragone, and A. Saffiotti, “The internet of robotic things: A review of concept, added value and applications,” International Journal of Advanced Robotic Systems, vol. 15, no. 1, p. 1729881418759424, 2018.

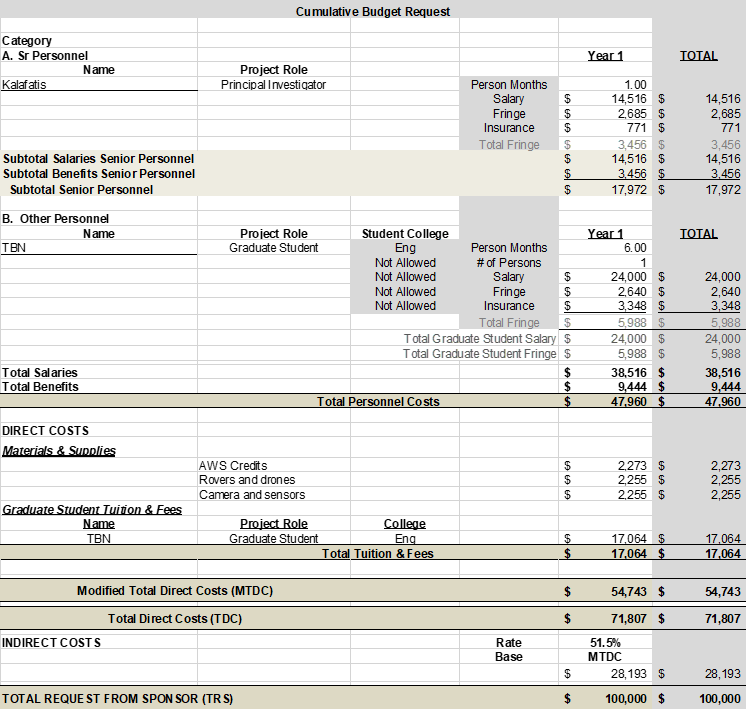
8. Sorrentino, Alessandra, Filippo Cavallo, and Laura Fiorini. "A Plug and Play Transparent Communication Layer for Cloud Robotics Architectures." Robotics 9.1 (2020): 17.

9. IPERF Documentation. Available online: https://iperf.fr/ (accessed on 10 May 2018).

10. Lala, Parag K. "A single error correcting and double error detecting coding scheme for computer memory systems." Proceedings 18th IEEE Symposium on Defect and Fault Tolerance in VLSI Systems. IEEE, 2003.

11. Shrier, David, Weige Wu, and Alex Pentland. "Blockchain & infrastructure (identity, data security)." Massachusetts Institute of Technology-Connection Science 1.3 (2016): 1-19.

**Budget**

****