Modular System of Networked Embedded Components for a First-Year

Mr. Michael Henry Schulz, The Ohio State University

Michael H. Schulz is a teaching assistant with the Fundamentals of Engineering Honors program at The Ohio State University. He is currently the lead developer of the robot course software development team, of which he has been a member for three years. As a Computer Science and Engineering (CSE) student, he will graduate in May, 2017 with his B.S.C.S.E and a minor in Music, Media, and Enterprise.

Mr. Evan J. Danish, The Ohio State University

Evan J. Danish is a Computer Science and Engineering major at the Ohio State University and an Undergraduate Teaching Assistant for the Fundamentals of Engineering for Honors program. He will graduate with a B.S. in May, 2017.

Tyler Wolf Leonhardt, Microsoft Corporation

I am an alumni of The Ohio State University where I achieved my bachelor's degree in computer science and engineering. I was a teaching assistant for the Fundamentals of Engineering for Honors courses for 3 years while attending university. I am currently working as a Software Developer for Microsoft on the Forza video game franchise.

Mr. John William Jackson, Ohio State University

I am a recent graduate of The Ohio State University who majored in Computer Science and Engineering. I worked as a TA for the FEH program for three years during my time at Ohio State and helped to develop the software system that is used for the robotics course each year.

After graduating, I began work as a Software Development Engineer for Amazon. I live and work in Seattle, Washington.

Mr. David Joseph Frank, Ohio State University

David J. Frank is a 4th year Computer Engineering honors student at The Ohio State University and a Graduate Teaching Assistant for the Fundamentals of Engineering for Honors program. He will graduate with his B.S.E.C.E in May 2017, and his M.S.E.C.E in May 2018.

Dr. Richard J. Freuler, The Ohio State University

Richard J. (Rick) Freuler is a Professor of Practice and the Director for the Fundamentals of Engineering for Honors (FEH) Program in Ohio State's Department of Engineering Education in the College of Engineering. He teaches the two-semester FEH engineering course sequence and is active in engineering education research. He is also affiliated with the Mechanical and Aerospace Engineering Department and conducts scale model investigations of gas turbine installations for jet engine test cells and for marine and industrial applications of gas turbines at the Aerospace Research Center at Ohio State. Dr. Freuler earned his Bachelor of Aeronautical and Astronautical Engineering (1974), his B.S. in Computer and Information Science (1974), his M.S. in Aeronautical Engineering (1974), and his Ph.D. in Aeronautical and Astronautical Engineering (1991) all from The Ohio State University.

Modular System of Networked Embedded Components for a First-Year Engineering Cornerstone Design Project

Abstract

In Engineering cornerstone design projects, creating automated evaluation methods for those projects that attempt to mirror the complexity and variability of the real world is a challenging task. Furthermore, achieving such variability while maintaining accuracy often comes with a cost. To address this, an adaptable system of networked devices was developed with a combination of PCs, in-house boards, and hobbyist boards, such as Raspberry Pis and Arduinos.

The function of the system is to evaluate the performance of autonomous robots developed as part of a first-year engineering design project. The components that comprise the system are embedded into each of the four 18-square foot course regions on which an individual robot completes a series of tasks. The tasks must all be completed in two minutes and can consist of anything from switch/button interaction to moving and depositing materials. The course can also include moving parts and illuminated objects that can either be aesthetic or used to communicate information to the robots.

Introduction

The first-year engineering student experience in the Fundamentals of Engineering for Honors (FEH) program of the Department of Engineering Education at The Ohio State University includes a culminating cornerstone design project. The project carried out by teams of four students involves designing, building, testing, and demonstrating small autonomous robotic vehicles which must perform specified tasks while operating on a specially built robot course. To control the mechatronic elements of the course, a Raspberry Pi 2 Model B is used to drive an Arduino (via a serial connection), an I/O Board (via I2C), and a PWM Board (via I2C). This set of boards is synchronized via the Raspberry Pi to a central PC, allowing the four regions of the course to run in parallel and provide the opportunity for competition between four robots.

From the student perspective, the system is simplified and abstracted through a touch screen interface that runs on the central or main course control PC. This interface allows the students to restart runs, stop runs, change task requirements, and randomize tasks on the course. With this interface, students can directly control the specific environment conditions, allowing them to test and prepare their robots for when task states are randomized during competition. This interface, along with the PC's communications with other systems, allows live, automatic scoring and ranking of competing robots during competitions.

The physical robotics course is developed each year to provide a variety of tasks as part of a larger theme. The CAD model for 2016, with the theme "Rocket Launch," can be seen in Figure 1 on the next page. The scenario was: "A group of workers at the Fundamentals of Engineering for Honors Space Administration (FEHSA), a research and exploration collaborative, have designed and built a rocket for space exploration. Most launch preparation tasks can be performed by human personnel; however, a designated safety zone surrounding the rocket itself (known as the spaceport) contains hazardous and sensitive materials. FEHSA believes it would be safest and most efficient if an autonomous robot were used to complete necessary preparation tasks inside the spaceport." The autonomous robots had to perform several tasks including initializing communications systems by toggling several switches, delivering supplies to the launch pad, activating the system for loading the rocket with fuel, and signaling the start of the launch sequence.

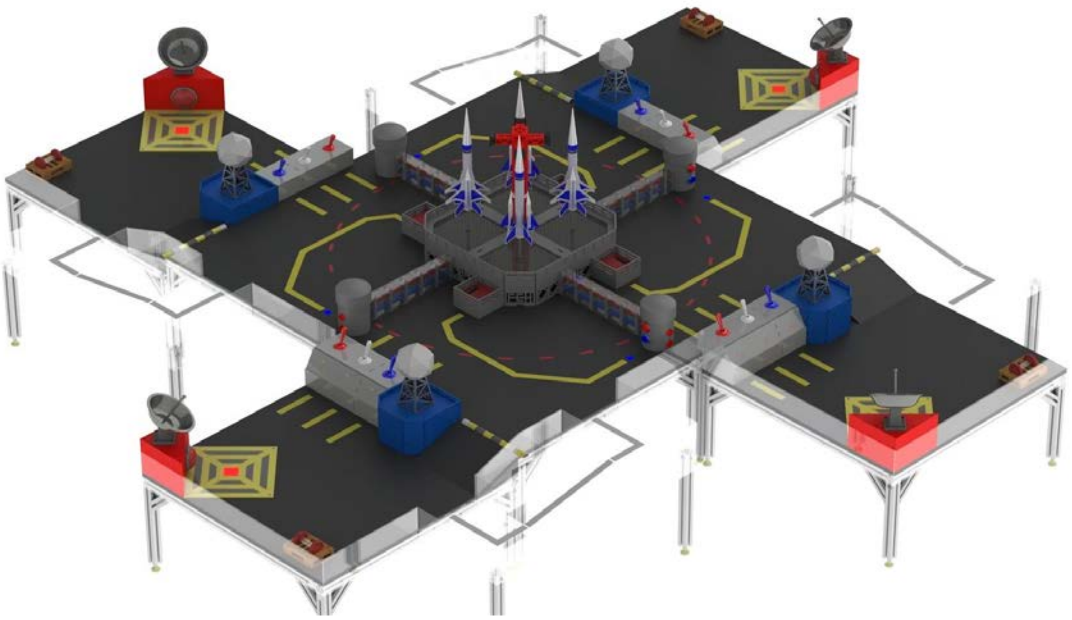

Figure 1: Rocket Launch themed robotics course from 2016

The annual construction of a new robotics course, which would provide unique tasks for students, created a large demand on hardware and software development. That demand then paradoxically hindered the original goal of achieving complex and interactive tasks. To minimize this demand and allow complex course design, a system was developed that can facilitate a variety of sensors and actuators, while maintaining cost effectiveness.

The robustness of the robotics course allows for more involved tasks and thus more complex student-designed robots. The uniqueness of the course year to year provides an environment for students to develop unique robot designs that must employ a variety of components to complete every task. Additionally, the PC user interface provides a way for students to control the course. With this control, students can more thoroughly test their robots, providing expanded opportunities for design and development and enhanced student learning. The interface also allows all four regions of the course to be synchronized, which allows competitive runs between robots with the same randomized requirements.

Similar Systems

A variety of engineering programs include robot design projects as part of their curriculum. Examples include the use of LEGO Mindstorms to complete tasks such as solving a Rubik's cube[1] or the development of robots to follow lines[2]. Because the complexity of these tasks is relatively low, many of these programs implement manual evaluation methods to measure student performance. While some programs utilize electronic scoring systems, a cursory review of the literature did not reveal any systems on the scale of the one described in the following sections.

Automated scoring systems have been designed for other domains. For instance, researchers proposed a system consisting of an Arduino and personal computer to record hit times and force levels during matches of Kendo, a Japanese martial art[3]. This system only measured one sensor and was not designed to be a real-time indicator of progress. Real time monitoring and control systems using networked embedded devices have been implemented in other domains. Others had created a prototype network consisting of Raspberry Pis, Arduinos, and a personal computer to measure vibrations and detect ice buildup on offshore platforms[4]. This system provided a real-time display of the sensor data and allowed for multiple sensor node devices to be connected to the network. Additionally, researchers described a sensor network consisting of multiple Arduinos and a Raspberry Pi as a base station configured in a star topology to monitor environmental conditions within buildings. This system also included a real time graphical web interface to view sensor data[5].

While the networks described above both allowed for the connection of additional homogeneous nodes, neither provided a framework for connecting heterogeneous devices. To address this issue, researchers proposed a framework for Internet of Things devices with the key characteristic of providing a method of addressing variability in connected device types[6]. The framework followed a layered architecture approach, consisting of layers for sensors and actuators, a controller for these devices such as an Arduino or similar microcontroller, a hub layer consisting of a general computer such as a Raspberry Pi, and finally a “cloud” layer that allowed remote user interaction. While this layered approach is like the final method proposed in this paper, it lacked a key component, namely the portion of the software that provided context to the sensor data and ultimately an evaluation system. In addition, this framework did not allow for the configuration or network topology required for the course software framework.

Earlier Systems

The system has evolved over the past ten years. Originally, the course software was entirely rewritten every year, in C/C++, and ran on a BeagleBone Black. This resulted in a limited capability to design new tasks for the students’ robots to complete, as there was no core framework on which to build these features. Eventually, a new, reusable framework was created to address this issue. That system was designed to minimize the amount of development work from year to year. Software components corresponding to hardware devices, such as buttons, levers, and LEDs, would remain unchanged year to year, allowing the developers to focus on new features. Additionally, the system was implemented in C# rather than C/C++ to take advantage of the features available in Microsoft’s .NET Framework. With this modified system, the design of a new course only required changes to several XML files, a XAML file, and a single C# class. The XML files configured the network, the XAML file defined the user interface, and the C# class contained the high-level logic. Examples of high-level logic in the C# class include which LED should be which color, or which button corresponded to the robot completing a particular task. The software ran using Mono on BeagleBone Blacks, which were all connected to a central course computer and to each other via a network switch.

The software was designed with a tree as the underlying data structure. The tree was of arbitrary height and size. Each node in the tree was a class representing a piece of hardware. The root node represented the main computer, internal nodes represented circuit boards and wires, and leaf nodes represented Input/Output (I/O) devices. Each edge in the tree represented a connection between two pieces of hardware; that is, a parent node represented some hardware component that was directly connected to the hardware component represented by the child node. When a circuit board received an interrupt from an input device, such as a button, the software on the circuit board would update its state and send a message up the tree to the main computer, informing it of the change. The main computer would then update its own state to match that of the microcontroller. This state could then be written to other software systems like the Robot Positioning System or competition bracket. Likewise, if a user interacted with the main computer via the user interface, the software on the computer would update its state, then send a message down the tree to the corresponding microcontroller, informing it of the change. The microcontroller would then update its own state to match that of the course computer. This state could then be written to an output device like an LED.

Each node had an address line, such as a string or an integer, to determine the destination or origin of a message. By traversing from the root of the tree to a node, pushing address lines onto the back of a linked list along the way, an address list for that node could be generated. Address lists were unique because nodes could not have the same address line if they were siblings. When the root sent a message to a leaf, it attached the unique address list of the leaf to the message. The root then popped the first line off the address list and sent the message to the child that matched the new front of the address list. This process was repeated on internal nodes, until no more address lines remained, and the message reached the leaf. When a leaf sent a message to the root, it attached an address list to the message and pushed its address line onto the front of the address list. The message was then sent to its parent. When the message arrived at its intermediate destination, the intermediate node repeated the process. This would happen until the message reached the root. When the message arrived at the root it could read the address list to determine the origin of the message. Messages were sent over the network via the UDP protocol.

Current System

The current system was designed using a client-server model with the idea that any client could attempt to connect to the course server, which runs on the course computer. The server accepts clients with a client-requested name and handles each one as if it were its own unique region of the course. Synchronization is done via an event system, where a physical event on the course, such as a button press, results in a software event that is sent over the network, so that the corresponding component on the server is updated. Rather than an addressed based system like the original framework, the current framework uses a subscription based system such that any component of the software that wants to receive an update about a particular component can subscribe to those updates. The state of the entire course is not serialized or sent over the network; only individual events are serialized and sent. Rather than using UDP, the current system works with a TCP connection, which guarantees in-order packet delivery and thus is more reliable.

The client runs on a Raspberry Pi 2 Model B. Each Raspberry Pi has an XML configuration file with network information and other runtime configuration values, and a DLL containing the code required to connect to the server. Once the client connects, the server sends the remaining DLLs across the network, which are dynamically loaded by the client to begin the application. This reduces the amount of time it takes to push new code onto the live system.

Each client Raspberry Pi is connected to an Arduino via a serial port, and to an I/O board, a Pulse Width Modulation (PWM) board, and an Analog Board, all via the Raspberry Pi's I2C ports. The communication is managed with classes that represent hardware. These classes are written as a part of the software, with references to an open source project called Raspberry#. Communication over I2C allows for a reliable and fast update cycle. On the client, all the components are updated at each cycle. Each update is performed by polling the boards over the I2C communication. The I2C protocol operates at approximately 100kHz, so these updates happen extremely quickly. The Arduino is used for aesthetic LED displays, and can either run independently of the Raspberry Pi and the course software, or accept data from the course software and display a specific pattern. A diagram of the system architecture may be seen below in Figure 2.

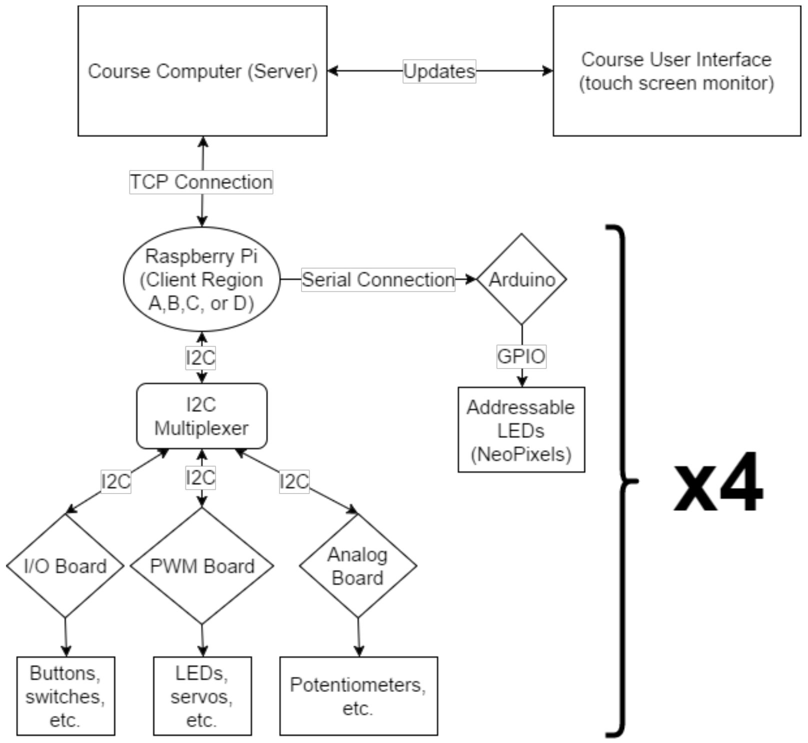

Figure 2: Overview of System Architecture

Like the previous system, the underlying data structure is a tree. Each component is responsible for keeping itself synchronized with its counterpart on the server or client at each discrete update cycle. Between update cycles (cycles are typically defined to be as short as possible, so that the course updates as often as possible), events are queued in a buffer. Events represent anything that happens on the course, such as a button press, or anything that happens on the user interface, such as a student indicating that an LED should be turned on. At the next cycle, this queue is emptied by sending those updates across the network for the server or client to consume, and a new queue is filled with updates from the new cycle. Update cycles occur at a rate of approximately 80 to 100 cycles per second (variable due to hardware clock speeds and network latency), and a single cycle encompasses everything that needs to happen for the server and client to remain synchronized.

The server, which runs on the central course computer, is responsible for determining higher level properties, such as whether a robot has adequately completed a given task. Given the physical state of each client, the server translates that data into a format that is meaningful for the scenario and for the robot competition. The server also maintains a separate timer for each region, so that the students can see how long it takes their robot to complete a run. This timer has millisecond resolution, meaning ties in score can easily be broken using the time. Information regarding task completion, robot performance, and current course state can be seen on the user interface designed for the course. Students interact with the user interface by choosing requirements for specific tasks, randomizing requirements, starting or stopping a run, or viewing their score after a run is over.

Finally, the course computer also communicates with the Robot Positioning System (RPS). The RPS is responsible for tracking each student's robot and providing a coordinate system for those robots on the course[7]. The RPS also repeats the data sent to it from the course software to another networked computer that manages the bracket for the final public competition. The bracket displays the state of each course for the audience, calculates the score for a robot, and generates a PDF of that score to be used for grading.

Comparison of Systems, Effects on Students

The update pattern in the original framework caused delays in the time it took for events on the course to be reflected on the course computer's display. This was caused by the overhead involved with creating and sending the messages, and with generating the address for a message receiver. Network packets sent via UDP, as in the original framework, are not guaranteed to arrive at their destination nor to arrive in any particular order, and issues involving dropped packets impacted performance as well.

These issues impacted the students because the tasks that their robots completed would not register as soon as they needed to for the students to adequately progress in the challenge. Additionally, the steep learning curve of the system made it difficult to learn for new developers, which hindered the original intent for a reusable framework that was simple to add onto. Finally, a runtime error on any region (which could be caused by something being unplugged) would result in the entire four region course crashing. While it is difficult to evaluate the performance from previous years, Table 1 below shows high level, anecdotal data comparing the original and current systems and their performance.

The original framework required the developer to individually and manually put updated code on both course's computers and all eight microcontrollers each time a course software update was developed. The current framework only requires the developer to put the code on the course computer, from which the update is automatically copied to any microcontroller that connects, at runtime. Students benefit from this functionality during testing times because changes to the software do not require the course to be out of commission for longer than about a minute. With the previous framework, any type of update would require the course to be down for at least twenty minutes.

Table 1: Comparison of Systems

	Original Framework	Current Framework
Course State Update Cycle	~10Hz, Inconsistent	~100Hz, Consistent
Network Protocol	UDP	TCP
Course Restarts Per Week	12-15	1-2
Course State Update Reliablility	Low	High
Difficulty of Development	High	Low

The modularity of the course software allows for maximum reusability and simple testing. Each component, like a button or LED, is represented by a corresponding class in software. Those classes, along with all the code related to networking, update time steps, and synchronization, do not change year to year. It is simple to build new components and easy to use old components.

Like the modularity of the code itself, the structure of the course software results in the functional modularity between the independent regions. A runtime issue, such as an unplugged cord, would crash a single region but not an entire course, thus only affecting the students on the affected region, not the surrounding regions. This has a huge impact during the student's testing. In the past, if the course went down, all four regions would be down until someone fixed it; the current system allows for minimum impact on students should something go wrong.

The reliability of this system also benefits the students in several ways. First, when a student group's robot accomplishes a task, the students are guaranteed to see the results practically immediately, both on the course and on the user interface. In other words, they can rely on the integrity of the course so that they can focus on their own design goals rather than accounting for potential errors caused by the course itself. During testing and competition, this software provides students with accurate, immediate feedback on the user interface and on the course regarding how well their robot performed. Additionally, this software displays the actual score received by a robot in real time, even during tests. These two aspects together provide students with the means to perform more formal tests that are backed by real-time scoring data. Because the score is updated the instant a task is completed, students are not required to test their entire robot during every testing session, rather, they can design modular tests that evaluate a single aspect of their design.

Finally, the real-time scoring includes the timing data from the course computer. Students can record, with millisecond resolution, the time it takes their robot to complete the test they are performing. With this information, students can make informed design decisions about where they should improve their robot. The timing data, coupled with the scoring data, also gives insight into the consistency of a team's robot. A team that can complete the course with a perfect score in approximately the same amount of time each time is better off than a team that can earn a perfect score, but with a large variance in time. Having the real-time scoring and timing data exposes these insights, which further allows for refinement of the students' design.

Conclusion

Two iterations of software designed to control a freshman honors engineering design program's robotics course were analyzed and compared. The original system was designed with specific goals in mind, but was unable to perform adequately, and its effects on the students were negative and prohibitive. The current system, which utilized low cost hobbyist microcontrollers such as the Raspberry Pi 2 Model B and Arduinos, allowed for a variety of supported sensors, as well as the ability to automatically and instantly score students on the performance of their robots. Beyond providing a means to operate the robotics course, the system also aided students in testing their robots and provided consistency throughout the project. The user interface allowed students to easily change and control the settings of the robotics course to enable better testing. While mainly supportive in its nature, the system provides great benefit to design programs in general. Through its modularity and support of Arduino, digital and analog I/O, and PWM, the system is easily adaptable to any program in which sensors need to be monitored or actuators need to be controlled.

References

1. Clemson University: "Freshman Engineering Robotics Project." College of Engineering, Computing and Applied Sciences | Freshman Engineering Robotics Project. N.p., 2017. Web. 5 Feb. 2017. http://www.clemson.edu/cecas/departments/ece/academics/undergrad/mindstormslab.html
2. Villanova University: "Freshman Engineering Program." Freshman Engineering Program. N.p., 2017. Web. 5 Feb. 2017. https://www1.villanova.edu/villanova/engineering/undergrad/freshmanprogram.html
3. E. Hogan, "Wireless Electronic Scoring of Kendo Competition Matches Using an Embedded System," *EWU Master's Thesis Collection*, Paper 89, 2013. http://dc.ewu.edu/theses/89/
4. P. Krishnamoorthy, C.S. Chin, Z. Gao, and W. Lin: "A multi-hop microprocessor based prototype system for remote vibration and image monitoring of underwater offshore platform," in *2015 IEEE 7th International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM),* July 2015. http://ieeexplore.ieee.org/document/7274585/
5. S. Ferdoush, and X. Li: "Wireless sensor network system design using Raspberry Pi and Arduino for environmental monitoring applications," *Procedia Computer Science 34*, 103-110, 2014. https://www.researchgate.net/publication/264827590_Wireless_Sensor_Network_System_Design_Using_Raspberry_Pi_and_Arduino_for_Environmental_Monitoring_Applications
6. F. Anon, V. Navarathinarasah, M. Hoang, and C. H. Lung: "Building a framework for internet of things and cloud computing," in *2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom)*, September, 2014. http://ieeexplore.ieee.org/document/7059653/
7. Frank, D.J., K.J. Witt, C.P. Hartle, J.J. Enders, V.M. Beiring, and R.J. Freuler: "A Low-Cost Robot Positioning System for a First-Year Engineering Cornerstone Design Project", *Proceedings of the 2016 American Society for Engineering Education Annual Conference,* New Orleans, Louisiana, June 2016. https://peer.asee.org/26355. DOI 10.18260/p.26355. Also published in the Computers in Education Division of ASEE *Computers in Education Journal*, Vol. XXVI, No. 3, pp. 41-50, July-September, 2016.