EE449 – MS4 – TEAM 4

Factory Floor Testbed

Hardware / Software Details

Stefan Kristjansson, Andrew Lawrence, Richard Wood

5/21/2010

Table of Contents

Introduction

In Milestone 3 (MS3) we discussed our focus shift from low level robotic arm control to high level resource management and structure assembly algorithms. With Milestone 4 (MS4) we look to continue this discussion by describing the details of the hardware and software implementation. Most notably, we present the Computation and Control Language (CCL) being used for the multi-tile control and interaction, and describe how it is being implemented in our project and why CCL is advantageous for use in distributed systems. To realize CCL, a C library interface to Python must be developed and its design is discussed. In addition, we present an update to our project and present our plans for demo, and discuss current technical obstacles and team management.

Project Update

Project goals have not changed since the previous project update. High level resource management and structure assembly algorithms are still the main priority. To support this effort, a C language library interface between CCL and the low level Python is being developed to allow communication from CCL to the CKBots. For MS3 we presented a simulation written in C to demonstrate the controller design, including tile states and roles, and assembly and passing techniques. This simulation has since been written in CCL and was presented in our MS4 presentation. After discussion with our customer, however, modifications to the algorithm structure and to the physical hardware have been determined to be necessary and will be pursued. A framework of the revised CCL implementation and the three programs established to govern the behavior of a tile are presented in the Software Implementation section of this report.

Hardware Implementation

With respect to hardware, little design decision was left to our team. In general, the CKBot modules were fully developed by the MODLAB at the University of Pennsylvania, as were the layout and design of the Factory Floor Testbed (FFT) and use of the Phidget I/O board for sensor data. It is important to consider, however, that even though the hardware was provided and our development for the project is devoted almost entirely to software, there have been many challenges presented by the hardware.

One of these challenges is the inability to place a truss without the previous placement of a corresponding node. This problem arises because the end-effecter of the robot has magnets, which attach to the truss and hold it in place. The truss cradle, however, does not. Consequently, the only way to remove a truss from the end effecter is to have a magnetic node in place at one or both ends of the truss placement location that can pull the truss away from the end effecter.

This constraint severely limits the assembly possibilities in our high level algorithms since we have to ensure this situation never occurs. A solution to this problem is to enable truss removal without the help of a node. This can be realized with the attachment of properly placed magnets on the truss cradles, and is an addition which will be completed in the coming week.

Software Implementation

High Level

The high level software and control of the FFT is implemented using CCL. The implementation of which is discussed in the following section per the algorithms detailed in the MS3 Report.

CCL is a guarded command language and is designed for the control of distributed systems. Some of the advantages of using CCL include its ability to run multiple programs in parallel. That is, if one program were to be written that describes all the algorithms and behaviors of a single tile, this program can be extended and implemented for each tile in the FFT. Another advantage of CCL is its keen use of guarded commands. The guarded commands are composed of simple Boolean expressions that when evaluated, protect the resources used in the Boolean expression. This prevents programs from performing two conflicting tasks on the same resource, such as two tiles commanded to move a node to two different locations.

For the FFT, the behavior of each individual tile is composed of the following set of programs:

1. Check Tile
2. Determine Job
3. Resource Control

The “Check Tile” program deals with determining the completion of a tile, that is, it compares the resources (nodes and trusses) currently placed at a tile with the global reference of the structure. A series of flags for unlocking the guards on various commands in “Determine Job” and “Resource Control” are set within the program based on the location of the resources. For instance, this program may set the flags for passing Builder rights to the next tile in its given column if all resources have been appropriately placed on the tile per the reference structure. Further, this program is critical for the detection of disturbances or faults in the system. For instance, if a resource previously placed is missing, a flag is triggered, setting the tile to Repair mode so that a set of actions may be taken to replace the missing resource. In regards to the hardware, the “Check Tile” program will use the feedback from the contact switches on the cradles by pinging the Phidget I/O board.

The “Determine Job” program observes the flags of “Check Tile”, and coordinates the jobs and completion states of the tiles. The jobs designed for in the CCL implementation are the following:

1. Builder
2. Passer
3. Repairer
4. Emergency Repairer

The completion states are as follows:

1. Floor Complete
2. Structure Complete

For more information on the jobs of a tile, or its states, refer to the MS3 Report.

The “Resource Control” program controls the behavior of a tile in its interaction with resources. It coordinates how nodes and trusses are placed within a tile, or coordinates the passing of a resource between neighboring tiles. The Resource Control program interfaces with the Python CKBot control for the actuation of the arms.

Simulation

The CCL framework is used in a simulator written in C++. The simulator emulates the hardware by providing definitions of the resources, and attributes of the tile to the CCL Testbed framework. The simulator does not strictly define cradles and the arm as entities to interact with resources; these components are instead abstracted away by providing additional resource definitions. For instance, there is a definition of a “Node” and a “Node_G”. The “Node” definition represents a node in a cradle, while the “Node_G” definition represents a node in a robot arm gripper. In this way, the passing of resources between positions can be captured. The simulator defines the following resources:

1. Node – node in cradle
2. Node_G – node in an arm's gripper
3. Truss_X – truss placed along the x-axis of a tile
4. Truss_Y – truss placed along the y-axis of a tile
5. Truss_Z – truss placed vertically on top of the node in a tile
6. Truss_G – truss in an arm's gripper

The definition of a tile has been modified slightly to better suit the simulation. A tile is composed of an arm, two truss cradles (one for x-axis placement, and one for y-axis placement), and a node cradle. Figure 1 below depicts a single filled tile with a node in the gripper of the arm. Figure 2 shows how the tiles fit together to form a piece of the Testbed.

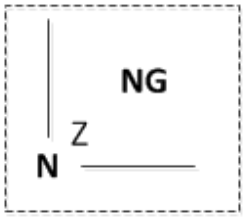

Figure 1: Simulation of Single Tile

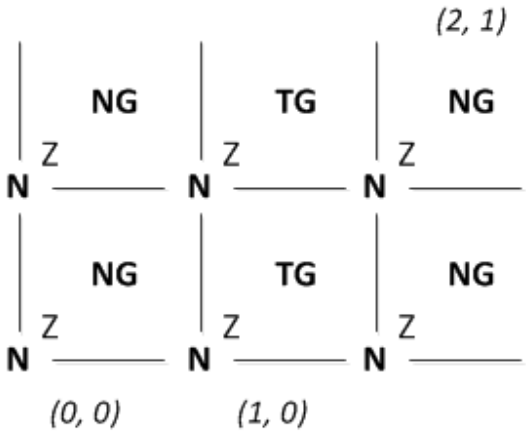

Figure 2: Grid of Simulated Tiles

Due to the separation of the CCL framework from the simulator, a simulated tile and a physically realized tile are entirely interchangeable, allowing both to operate concurrently. Thus, in the final simulation, one of the tiles in the outputs shown in Figure 2 above may be selected and used for a Hardware-in-the-Loop (HIL) simulation. The HIL will be the single physical Factory Floor Tile in our possession. The goal is to then show that the HIL can perform the tasks of building a physical structure (determined by its representation as a physical tile) and that it appropriately interacts with the surrounding tiles to do so (determined by its representation as a simulated tile in the terminal).

Low Level

The following section discusses the low-level software that is used to issue position and movement commands to the CKBots.

The CKBots receive position commands via the Robotics Bus, an interface developed by the MODLAB at the University of Pennsylvania that extends the CANOpen protocol for CAN bus. In order for the project to be successful and have a HIL demonstration, the Robotics Bus interface needs to be in a form that can be used by CCL. There is currently no such implementation; the only implementation of the Robotics Bus interface available to us is within the CKBot software library provided by the MODLAB and written in Python. In order to use this preexisting software library, CCL and Python need to be coupled together through a C-interface. CCL is able to call C-functions and C is able to call Python functions through the Python Development Tools. Implementing the C to Python interface should take less time than writing our own implementation of Robotics Bus in C, and it will allow use of all of the software and commands developed by the MODLAB.

Technical Obstacles

Many of the technical obstacles encountered in the previous weeks still exist at this time. As they have been well documented in the previous report, they are simply listed here. The previous problems still unresolved include:

Inconsistent operation – Same command, same resource, different path trajectory

Oscillation – The robot reaches a position and will oscillate indefinitely

C Interface for CCL/CKBot Comm. – Need C library for CCL to communicate with CKBots

The same problems exist for MS4 as were seen in MS3. Progress has been made in correcting the issues, however, and although inconsistent operation is inherent due to underpowered servos, this can be improved by the addition of the new, more power base module retrieved from the ICRA 2010 Planetary Contingency competition. In addition, we can help reduce the strain on the motors by implementing our torque minimized path planning formulated for MS2. Together, these remedies should help improve the consistency of the robot operation.

The requirement of a C library for the high level CCL to communicate with the low level Python is a technical obstacle which came up in MS3 and has yet to be resolved. Fortunately, a solution to this has not been required at this point since the CCL is not in a ready state to test. Still, progress has been made and a working library interface between CCL and Python should be completed by next week, allowing testing of the CCL program which is being written at the same time the library interface is being created.

Team Management

Overall, communication between the team members has been effective and project work has been smoothly distributed and well regulated between the three man team. Each member has always had a side project to work on individually or discuss as a group and each member has contributed in the project advancement since MS3. In general, Stefan Kristjansson has been involved in the algorithm design for high level resource management and structure assembly. In addition, at this phase of development he is working on utilizing the torque minimized path planning to increase placement efficiency. Richard Wood has taken the lead role in CCL implementation and algorithm design, programming the simulation which will be used in our final project demo. Andrew Lawrence has also helped with determining algorithms, but has primary focused on the development of a C library for the CCL to Python communication.

Bibliography

(1) *CCL: The Computation and Control Language.* Retrieved April 05, 2010, from University of Washington, Self Organizing Systems Lab website, http://soslab.ee.washington.edu/mw/index.php/Code
Primary reference for documentation and source code for CCL.

(2) *Phidgets.* Retrieved April 13, 2010, from Phidgets website, http://www.phidgets.com/
Used as the source for phidget I/O documentation and source code.

(3) Mason, Matthew. (2001). *Mechanics of Robotic Manipulation.* Massachusetts: The MIT Press.
Utilized as a supplemental reference for forward kinematic equations of the robotic arm.

(4) *Modlab CKBot Graphic User Interface Manual.* Retrieved April 10, 2010, from UPenn, Modular Robotics Laboratory website, http://modlabupenn.org/efri/
Used as the primary reference guide for interfacing with the CKBot modules in the Windows environment.

(5) M. Yim, P. J. White, M. Park, & J. Sastra, Modular Self-Reconfigurable Robots. 2009, pp. 5618-5631.
A pivotal paper on self-reconfigurable robots; one of the primary CKBot modular robotic design sources.

(6) Nurrat, Richard, & Li, Zexiang, & Sastry, S. (1994). *A mathematical introduction to robotic manipulation.* Florida: CRC Press.
Utilized for instruction on the derivation of torque equations for robotic arm systems.

(7) Craig, John J. *Introduction to Robotics: Mechanics and Control*.(1989) Reading, Mass.: Addison-Wesley.
Used as the primary source for kinematic equation derivation and vector equation manipulation.

(8) *IPython Documentation.* Retrieved April 12, 2010, from IPython website, http://ipython.scipy.org/moin/
Used for reference documentation on IPython documentation and for the source code for compilation. Ipython is used to interface with the CKBots.

(9) D. Gomez-Ibanez, E. Stump, B. Grocholsky, Vijay Kumar, & C. Taylor. The Robotics Bus: a Local Communications Bus for Robots. In *Proceedings of SPIE*, Volume 5690. 2005.
A paper describing how Rob[otics Bus is used over a CAN to communicate with robotic modules, used as a reference for the creation of a C-interface with the CKBots over PCAN.