EE449 – MS3 – TEAM 4

Factory Floor Testbed

Controller Design

Stefan Kristjansson, Andrew Lawrence, Richard Wood

5/14/2010

Table of Contents

Introduction

In Milestone 2 (MS2) we looked to develop a solution for inconsistent operation of the robotic arm. This problem was determined to be due to underpowered servos and the solution developed for MS2 was torque minimization path planning based on forward kinematics and a simulation was developed in Matlab. After competing in the ICRA 2010 Planetary Contingency Competition, however, and getting a chance to work with much more powerful modules, we have decided to shift our focus back to the original objective of the project, and move the new path planning as a final task. Consequently, with Milestone 3 (MS3) we look to design a high level control algorithm for structure assembly, resource management, inter-Testbed communication and disturbance rejection.

For this update, we present the updates to our project, including our shift from low level robotic arm control to high level distributed control algorithms. We present the reasons for this and introduce the controller design, logic and implementation. In addition, we comment on the benefits of this design and the current limitations and known problems that need to be addressed. Next we discuss how control will actually be implemented in hardware and why and finally, we comment on remaining technical obstacles and team management for this stage of the project.

Project Update

For Milsetone 2, our group focused on control of the robotic arm in an attempt to correct the inconsistent operation of the robot due to underpowered servo motors. Although the limitations of the servos and the inability to know their current actual location did not allow the application of an integral controller, we did pursue using forward kinematics for use in torque minimization path planning. While we still look to apply this technique, after competing in the ICRA 2010 Planetary Contingency Competition and getting a chance to work with new, more powerful CKBot modules, we realize our time is best spent on another area of the project. The new modules are powerful enough for our application and will be released shortly and thus at this time our team is currently working on high level resource management algorithms in CCL instead of trying to patch a problem that will be fixed with the new modules. Since the resource management algorithms are the main goal for the Factory Floor Testbed (FFT), we look to fully design and implement these algorithms as our high level control problem for this project. In addition, we will work on implementing the new path planning technique as a possible solution until we receive the new modules.

To complete the controller and test the simulation interface to the actual hardware, we must build a C language interface library for CCL to communicate with the CKBots. As no member of our group has direct experience in doing this, the task is a major hurdle and is being completed concurrently with the development of the high level control algorithms.

We believe we are on track to complete the high level CCL implementation in the next two weeks, leaving a final week to work on increasing the efficiency of the robotic arm placements

through the torque minimization path planning. In addition, we look to build simulations to test the system and look to characterize the system with many different reference structures and disturbance rejection tests.

Controller Design

The objective of our controller is to implement a high level distributed assembly algorithm to manage inter-testbed communication of multiple robotic tiles in the assembly of a larger structure. Each individual tile, as well as the testbed as a whole, have control procedures for resource management and structure assembly. This section describes the distributed assembly algorithms designed for high level control and provides detailed explanations of their logic and the specific tile tasks and roles utilized in the design.

Figure 1 below illustrates a block diagram of the controller design implemented for this project. Each individual tile has its own Large Scale Reference (LSR) structure input and feedback control, which in turn is broken down to a Small-Scale Reference (SSR) for each individual floor. There are NxN tiles in a testbed and NxN feedback control systems within a larger outer loop comprising the Global Reference Structure (GRS) as the input and the full testbed structure as the output.

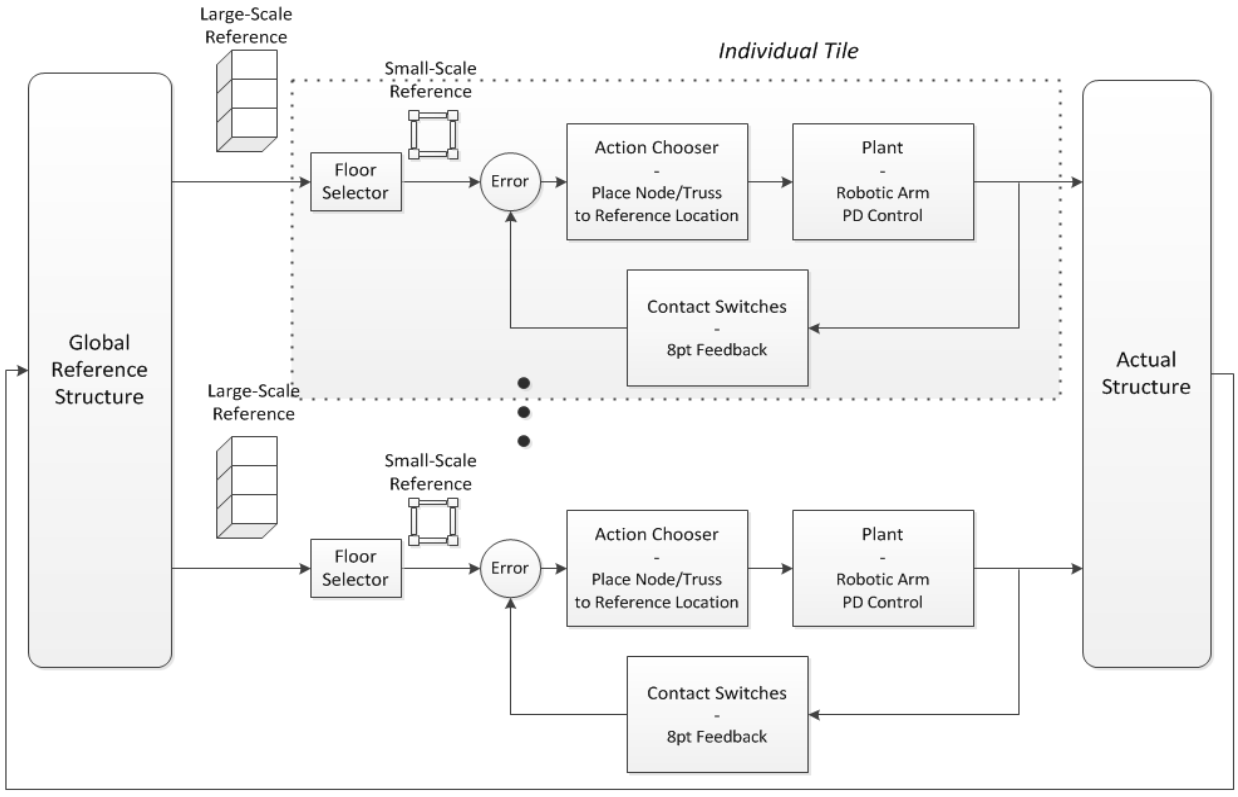

Figure 1: Testbed Feedback Control

All tiles within the testbed will have access to the overall GRS, which designates the placement of nodes and trusses on each tile for each floor. This will be used to coordinate resource management between each tile as well as provide a priority scheme in which tiles can be given jobs to complete individual tasks. From the global reference of the overall structure, an individual tile will extract relevant information for their structure as a LSR. This LSR is the node and truss placement for each floor of a given tile. Defined from the LSR is a SSR which is the node and truss information of the current floor being constructed by a tile.

Prior to construction of a structure a set of preconditions must be met. The first precondition is that the testbed must be complete. There cannot be an empty tile vertically enclosed within the testbed. This means that within a column there cannot be an empty tile between the first and last tile of a column of tiles. It is important to note that although a rectangular testbed is not required, for simplification the following discussion will assume such. The second precondition is that the GRS must be checked for physical attainability. For example, all trusses placed within the structure need to be terminated by a node at both ends. The final precondition is that resources enter the testbed at a single face (through each tile of said face).

After the preconditions are met, tiles are assigned priority by rows. The back row of tiles (opposite face to the entry of resources) is given the highest priority. Each following tile is given one priority lower than the next where the front row of tiles (the entry point of resources) is given lowest priority. An assignment system is then used to give individual tasks to each tile in the testbed, and each tile maintains a level of completion indicator.

The jobs and levels of completion are as follows:

Table 1: Job Designation and Completion Status

Jobs	Level of Completion
Builder (B) Passer (P) Repairer (R) Emergency Passer (EP)	Floor Complete (FC) Complete (C*)

To begin, all tiles with priority 1 are given Builder rights and all tiles of lower priority become Passers as shown in Figure 2.

The Passer job simply instructs an arm to pass resources across its own respective tile. All passes are directly between arms between adjacent tiles. This requirement is because trusses cannot be placed into a cradle without an adjacent node. Passers pass resources to tiles of higher priority. Further, direct passing between arms eliminates the removal of a resource that would have changed the state of the current tile as well as adjacent tiles.

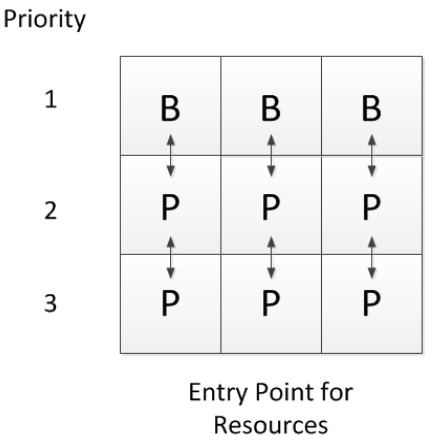

Figure 2: Job Initialization

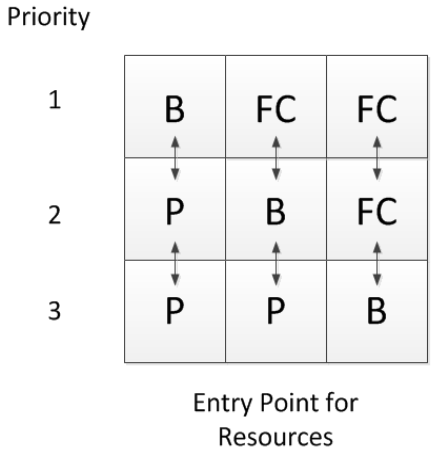

Figure 3: Floor Complete and Builder Passing

Removing a resource from cradles as a form of passing leads to a string of potential problems that direct passing easily resolves. The Builders receive resources from the Passers and place them into their respective cradles based on the SSR. When a Builder has completed the placement of all resources for the SSR, Builder rights are passed to the next priority tile within its column. The tile that passes its Builder rights becomes Floor Complete (FC) as shown in Figure 3. A tile cannot reach FC unless all tiles of higher priorities have reached FC to ensure that the necessary tiles remain available to pass resources. This pattern continues until all tiles of the testbed become FC. At this stage each tile confirms their status by verifying that the actual structure matches the SSR and then communicates the confirmed floor completion. When all floors are confirmed the elevators are used to simultaneously lift the floors of all tiles as one.

If upon confirmation a tile determines that a previously placed resource is not registered in its cradle that tile goes into Repairer mode (R). In this mode all tiles of lower priority are changed to Emergency Passers (EP) interrupting the previous job. Repairer mode requests the missing resource and the EPs retrieve the resource from the entry point. The EPs operate the same way as a standard Passer the only difference is that EPs have higher priority replacing any existing job with the EPs. Further, EPs have higher priority than a Repairer. If a tile is switched to EP that means a tile of higher priority is a Repairer, and thus requires resources to be passed. Also, a tile can become Repairer at anytime when in FC status. A FC simply checks its cradles for resources and verifies them to the SSR. See Figure 4 for an example of Repairer mode.

The Complete (C*) status for a given tile indicates that the tile has completed the placement of all resources in its LSR. Similar to FC, the C* status can only be achieved if all tiles of higher priority have reached C* to ensure that they remain active to pass resources.

In order to simplify the further explanation of how this algorithm works, consider a testbed composed of nine tiles in a 3x3 square configuration. On start-up the testbed checks preconditions. It then assigns priority and initial jobs for each tile. The Builder tiles within a row determine how many resources are required to complete their floor. The Builders then request the resources required as one large package. The Passers then begin retrieving resources. To explain how the Builders place their resources see the iterations depicted in Table 2 in reference to Figures 5.

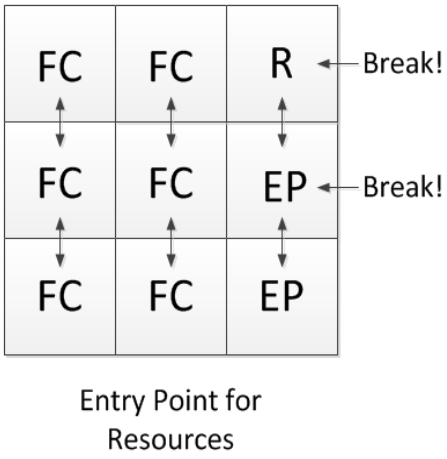

Figure 4: Disturbance Rejection - Repair Mode

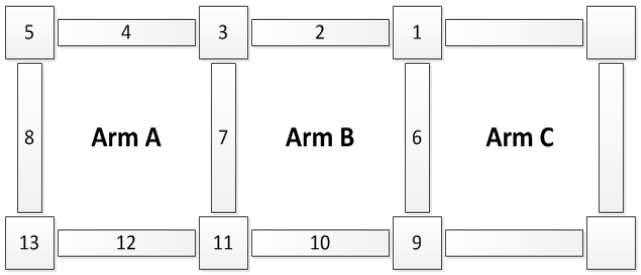

Figure 5: Organizational of Node and Truss Placement

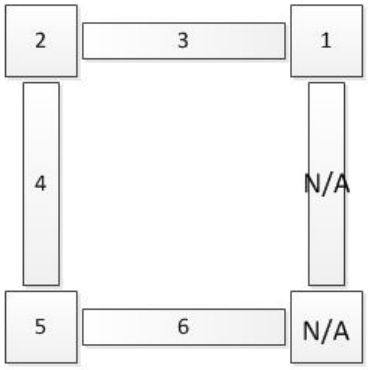

Figure 6: Order of Chronological Placement for Individual Tile

Table 2: Interative Steps for Node and Truss Placement

First iteration: Arm C places Node into 1. Arm B places Node into 3. Arm A places Node into 5.
Second iteration: Arm C places Truss into 6. Arm B places Truss into 2. Arm A places Truss into 4.
Third iteration: Arm C places Node into 9. Arm B places Truss into 7. Arm A places Truss into 8.
Fourth iteration: Arm C does nothing. Arm B places Node into 11. Arm A places Node into 13.
Fifth iteration: Arm C does nothing. Arm B places Truss into 10. Arm A places Truss into 12.

Following this iterative pattern, tiles that are not along the far right face will have an order of placement as shown in Figure6. This procedure has been chosen to reduce the effect of race conditions and to simplify the node sharing conflict between adjacent tiles. If the tiles did not coordinate within a given row, this could give rise to potential errors in the number of resources requested.

Implementation

The control algorithm will be implemented using the Computational and Control language (CCL). Each individual tile of the Factory Floor Testbed involved in the building of a structure will be provided with a copy of the CCL program, which will issue actions (such as ‘Pass’ or ‘Place’). These actions are communicated by the program to each individual robotic tile over a Controller Area Network (CAN) Bus using Robotics Bus, a local communication bus protocol for robots. The CCL program is able to estimate the state of the testbed surrounding each tile through the use of a Phidget I/O Board. Logical inputs for the Phidget Board determine the presence of nodes and trusses on the floor level currently being built through the activation of contact switches after resource placement. Since there is currently only one tile available for testing, the project will be implemented as a Hardware-in-the-loop (HIL) simulation. This

implies that the single physical tile may represent any of the tiles within a program and all other tiles will be handled by the simulation.

Technical Obstacles

All 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

Communication Failures – Communication freezes and requires software and hardware reboot

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

While inconsistent operation and oscillation still exist, we have determined this is a limitation of the servo motors, and as more powerful modules are being distributed soon, we are only addressing this issue with the new path planning after completion of the resource management algorithms. Communication failures still occur in both Windows and Linux environments, but communication with the MODLAB has shown that the repository of files we are using is out of date and we are currently working with the MODLAB to receive the newest code. Ultimately, our major technical obstacle with no current solution is the necessary C driver library files for CCL to communicate with the CKBots. Developing a driver is new territory for the members of our team, but we have been researching the problem. We believe we are close to completed code, and if further problems occur we will contact Albert Chiu, our CSE contact who may have experience in coding drivers and can point us in the right direction.

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 MS2. In general, Stefan Kristjansson and Richard Wood have been heavily involved in the design of the high level resource management and disturbance rejection algorithms. Andrew Lawrence has also contributed to the design as well as worked on the development of a C interface library for the CCL to CKBot 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.