EE449 – MS2 – TEAM 4

Factory Floor Testbed

System Model

Stefan Kristjansson, Andrew Lawrence, Richard Wood

4/23/2010

Introduction

Problems associated with the CKBot underpowered servos were discovered through initial path planning and highlight Milestone 2. This stage of project development was adjusted from the initial schedule to allow kinematic and dynamic system modeling to determine torque minimization path planning to reduce stresses on the CKBot modules. Consequently, with Milestone 2 we look to characterize the system introduced in Milestone 1, and address the controls implementation. In doing so, we describe the inputs, outputs and internal state of the system at both high and low levels of control, and introduce controller implementation for each level respectively. In addition, system modeling and parameters are discussed and a system simulation is described.

Project Discussion

High Level System

The high level control goal is for this project is for small Factory Floor "tiles" to build small structures such that the larger distributed system (the testbed itself) will construct a larger desired structure. A "tile" consists of a single robotic arm responsible for the placement of individual nodes and trusses. The scope of this problem revolves around the structure assembly of a single tile within the factory floor where the other tiles will be simulated.

All resources that a tile manipulates are given a reference expression defined as follows:

Horizontal Truss: Th (floor, location)
Vertical Truss: Tv (floor, location)
Node: N (floor, location)

Figure 1 depicts the layout of the expressed resources. The black octagon represents the arm itself, and the squares show the placement of the elevators used for lifting a constructed floor.

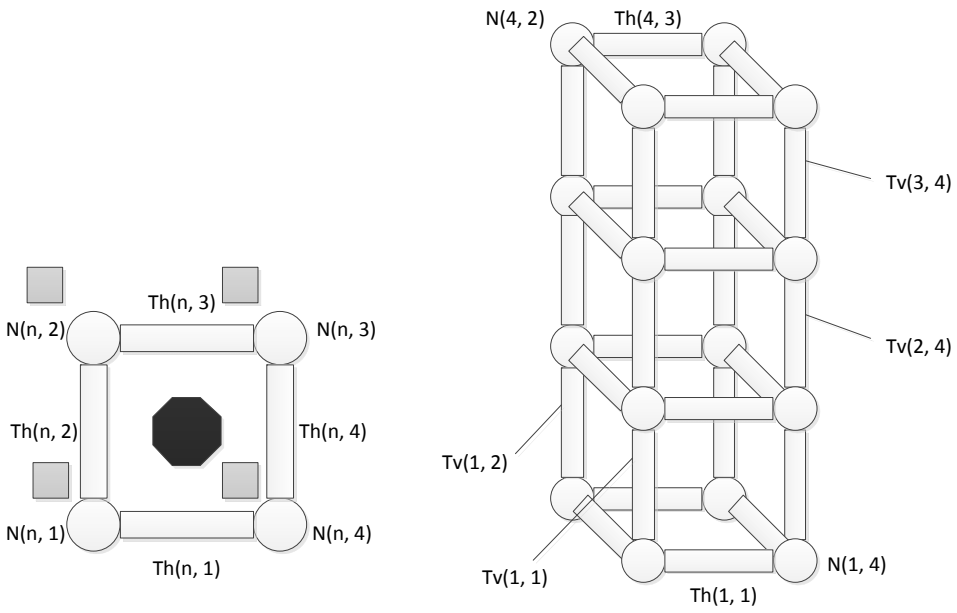


Figure 1: Node and Truss Defined

Due to the layout of each tile, the robotic arm is responsible for nodes 1-3, horizontal trusses 1-3, and vertical trusses 1-3, all for floor 1: the base floor. The fourth remaining node or truss is placed by the adjacent tile's arm. Once the base floor structure matches the desired structure, the elevators will lift the completed structure so that the next can be formed. Once a structure (generally a rectangle) has been formed underneath a structure held by the elevators, vertical trusses are then placed. After truss placement, the elevator lowers the above structure to connect to create a cube (or tower). The overall structure within that tile is subsequently lifted and the process continues until the overall desired structure is met. The high level system receives feedback on the placement of a resource from pushbuttons in the cradles for nodes and trusses.

The high level input is simply a reference structure. A binary representation of whether a node or truss should be placed at a given location within the previously defined framework. Input is provided from top down due to the nature of building up. The outputs of the system, in addition to the assembled structure itself, are the contact switches used to determine the completion of the actual structure, and are otherwise used to compare the reference structure to the actual structure. If there is a failure in the placement, the system will simply respond by performing the same action.

The controller of the high level system operates such that each action of the robotic arm reduces the difference between the physical structure and the ideal reference structure as represented by the following equation.

$$|ref - act_{n+1}| < |ref - act_n|$$

The high level system model is depicted in Figure 2. The "Action Chooser" uses the structure estimator to determine what action should be taken. The actions are retrieving and placing a node, horizontal truss, or vertical truss. The "Disturbance" incident on the Plant in this diagram represents the failure of a resource placement, or the removal of a resource previously placed. In the picture of the larger distributed system, the removal of a resource previously placed would be the interaction of a higher priority tile in the testbed requiring the resource for the construction of its structure.

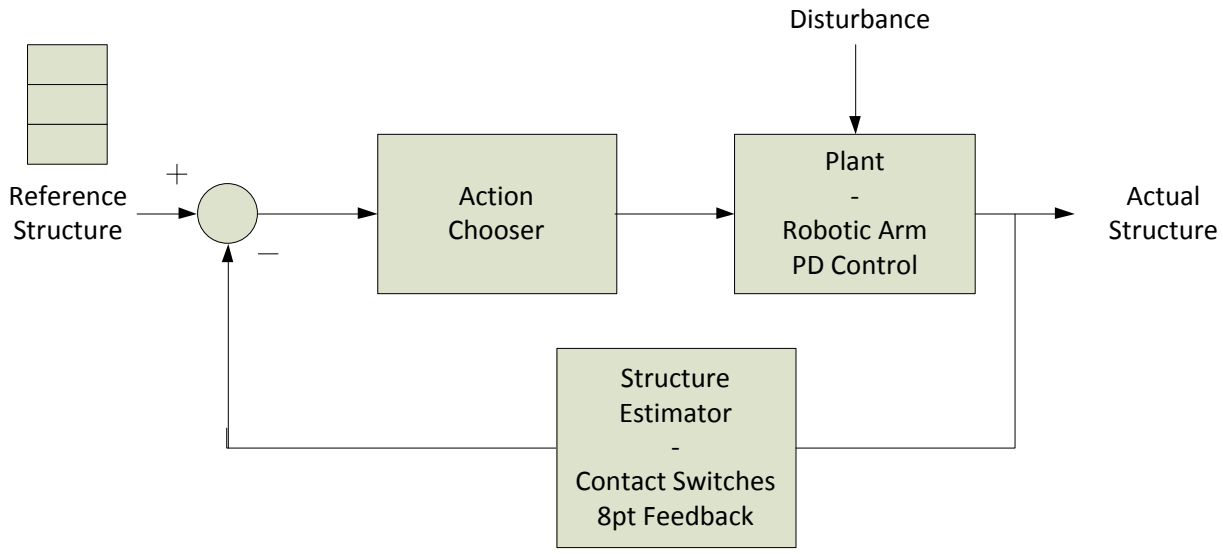


Figure 2: High Level Control Loop

Currently the characterization of the high level truss and node placement is as follows:

Node Placement
Total: 10 Success: 6 Failure: 4
Success Rate: 60%

Truss Placement (Horizontal)
Total: 10 Success: 8 Failure: 2
Success Rate: 80%

Truss Placement (Vertical)
Total: 10 Success: 7 Failure: 3
Success Rate: 70%

These success rates in the long term are not acceptable. To meet the design requirements a success rate of 95% or greater needs to be attained. To improve the success rate the low level control of the arm will be evaluated and improve through path planning.

Controllability of High Level System

The controls issue at hand of ensuring that a structure is properly assembled is not a classic controls problem; it is more an issue of controlling the flow and distribution of resources as well as implementing specific actions that work towards the assembly of the structure. As such, at this time, we are unsure how to model the system in a state-space where the rank of the 'A' matrix may be easily examined for controllability or observability. However, at this time, trusses and nodes are able to be placed, and the specific placements of these elements do work towards the achievement of some reference structure. Thus, by determining the actions of placing trusses and nodes, or moving trusses and nodes, in order to approach the structure goal, the system is controllable.

Low-Level System

The Arm is the low-level system that needs to be controlled in order to accomplish the high level task of building a structure. A block diagram of the system architecture for the Arm is shown below in Figure 3.

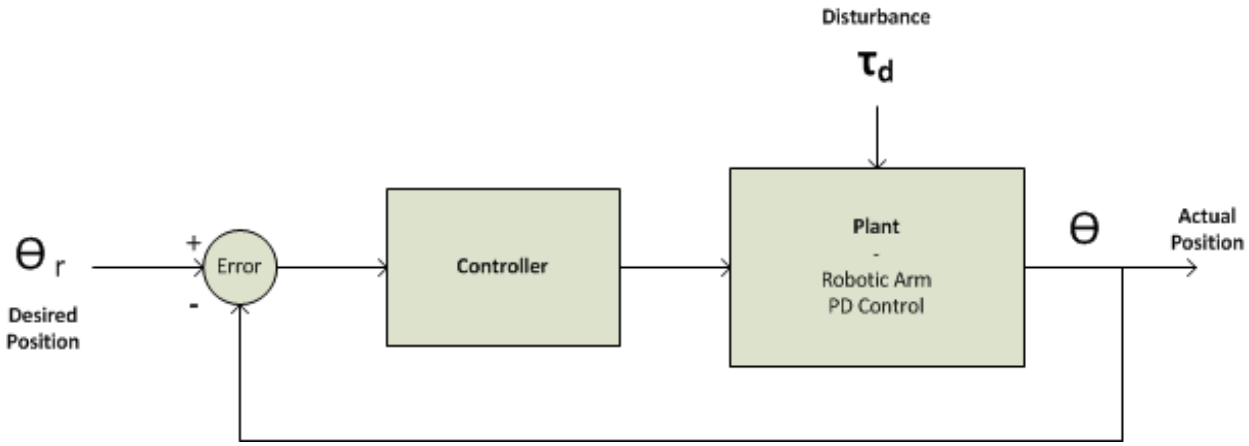
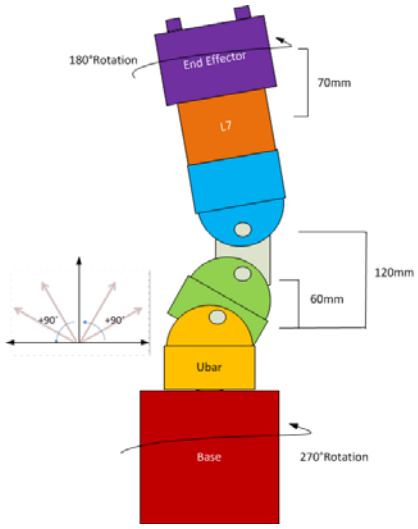


Figure 3: Lower Level Contrl Loop

The input to the Arm system is five reference angles—one for the base, three for each U-BAR joint, and one for the L-7 joint—as well as an open/close command for the truss-handling jaw on the end-effector. The outputs of the system are the five achieved angles of the Arm, creating a pose as shown in Figure 4 below. The state of the angle positions and velocities of the Arm are held in the vector *X*.



$$x = \begin{bmatrix} \Theta_1 \\ \Theta_1' \\ \Theta_2 \\ \Theta_2' \\ \Theta_3 \\ \Theta_3' \\ \Theta_4 \\ \Theta_4' \\ \Theta_5 \\ \Theta_5' \end{bmatrix}$$

Where θ_1 is the angle of the base rotation,
θ_2 is the angle of the bottom UBAR,
θ_3 is the angle of the middle UBAR,
θ_4 is the angle of the top UBAR,
and θ_5 is the angle of the L7.

Figure 4: Arm Internal State

The dynamics of the Arm may be modeled with the equation:

$M(\theta)\ddot{\theta} + C(\theta,\dot{\theta})\dot{\theta} + N(\theta,\dot{\theta}) = \tau$ [6],

which contains a momentum matrix, coriolis matrix, and friction matrix. The equation is set equal to a torque, but some algebra could relate the torque to an input angle per the current system inputs. A simplified model would be to use the set of equations:

$$\dot{\theta} = -k_p\theta + k_p u$$
$$u = \theta_r$$
$$y = \theta$$

Neither of these models have been developed for the Arm low-level system as of yet.
There are already PD controllers implemented for each servo motor controlling each joint of the Arm, but the system experiences disturbances, such as loading from moving a node or a truss, which cause torques greater than what the system may output in the setting of a position. These torques create huge errors in the position, pulling the Arm down toward the top of the tile, and can cause the Arm to fail in in the placing of a node or a truss.
To address the shortcomings of the Arm, there are two plans of action. One is to introduce an additional controller, such as a PI, outside of the already PD-controlled Arm with the idea that the integral component in this outer controller will eliminate steady-state errors as well smooth the movements of the Arm, making it move more fluidly. The second approach is to apply forward kinematics (and potentially inverse kinematics) to ensure optimal path planning for the Arm where the torques on the joints are minimized and do not exceed the maximum torques of the servo motors. Kinematics will be discussed in the following section.

Kinematics

Using forward kinematics, each joint of the Arm has been modeled so that the i^{th} joint position is known in relation to the $i\text{-}1^{th}$ joint, and by extension to a fixed origin and frame. Modeling the arm in this way creates a series of links, where each joint of the arm or point of interest (such as the end of the end-effector) is defined as a link.

A model of the Arm with defined frames for each joint is shown below in Figure 5 using the Denavit-Hartenberg notation with the measured dimensions and masses in Table 1 to the right [7]. The values for the masses of the base and end-effectors were estimated; in the future, more accurate numbers will be pursued.

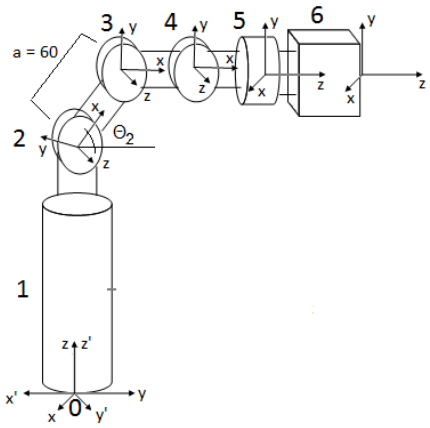


Figure 5: Arm Kinematics Framework

Table 1: Arm Parameters

Module	Mass (g)	Dimensions (mm)
Ubar	138	W60xL60xH60
L7	138	W60xL60xH60
Base	200	W60xL60xH85
End-Effector	130	W60xL67xH80
Node	129	W58xL58xH58
Truss	137	W35xL235xH35

The parameters relating a link, *i*, to the previous link, *i-1*, are defined below:

α_{i-1} = the angle between Z_{i-1} & Z_i measured along X_{i-1}
a_{i-1} = the distance from Z_{i-1} to Z_i measured along X_{i-1}
d_i = the distance from X_{i-1} to X_i measured along X_{i-1}
θ_i =the angle from X_{i-1} & X_i measured along X_{i-1}

The table below holds the link parameters for each link, *i*, according to the model of the Arm above.

Table 2: Arm Link Parameters

i	α_{i-1} (°)	a_{i-1} (mm)	d_i (mm)	Θ_i (°)
1	α_1	0	0	0
2	0	155	0	Θ_2
3	0	60	0	Θ_3
4	0	60	0	Θ_4
5	α_5	60	0	0
6	0	100	0	0

In the previous table, α_1 corresponds to the angle of the base rotation, θ_2 is the angle of the bottom UBAR, θ_3 is the angle of the middle UBAR, θ_4 is the angle of the top UBAR, and α_5 is the angle of rotation for the L-7.

Using the link parameters, the frame of link *i* is related to link *i-1* with the transformation matrix:

$$ {}^{i-1}_{i}\boldsymbol{T} = \begin{bmatrix} \text{Cos}(\theta_i) & -\text{Sin}(\theta_i) & 0 & a_{i-1} \\ \text{Sin}(\theta_i)\text{Cos}(\alpha_{i-1}) & \text{Cos}(\theta_i)\text{Cos}(\alpha_{i-1}) & -\text{Sin}(\alpha_{i-1}) & -\text{Sin}(\alpha_{i-1})d_i \\ \text{Cos}(\theta i)\text{Sin}(\alpha_{i-1}) & \text{Cos}(\theta_i)\text{sin}(\alpha_{i-1}) & \text{Cos}(\alpha_{i-1}) & \text{Cos}(_{\alpha i-1})d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}, $$

which contains information describing the rotation the rotation of the frame for link *i* relative to the frame for link *i-1* and a vector pointing from the origin of frame *i-1* to the origin of frame *i*. Lastly, the position and frame of each link are related to a fixed based frame for the system with the equation:

$$ {}^{0}_{N}\boldsymbol{T} = {}^{0}_{1}\boldsymbol{T}\,{}^{1}_{2}\boldsymbol{T}\,{}^{2}_{3}\boldsymbol{T} \ldots {}^{N-1}_{N}\boldsymbol{T} $$

The centers of mass for the Arm are shown in Figure 6 below.

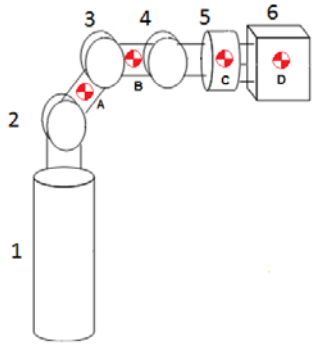


Figure 6: Center of Mass on Arm

Using the forward kinematics, the quasi-static torques for any position may be computed. The torques of interest are on the U-BARS, which are links 2, 3, and 4. The torques are computed using the cross product $\tau = F \times R$, where the torque on link 4 equals the torque applied by masses C and D on 4, mathematically written as $\tau_4 = \tau_{4C} + \tau_{4C}$. Similarly, $\tau_3 = \tau_{3B} + \tau_{3C} + \tau_{3D}$ and $\tau_2 = \tau_{2A} + \tau_{2B} + \tau_{2C} + \tau_{2D}$.

Each of the servos can exert a maximum of 2.94 N-m of torque. Using the kinematics, the goal is to ensure that none of the paths the Arm takes will cause one of the torques to exceed this limit.

Shown below is the MATLAB simulated position of the ARM holding a truss using forward kinematics. The static torques generated on the UBAR joints are shown below.

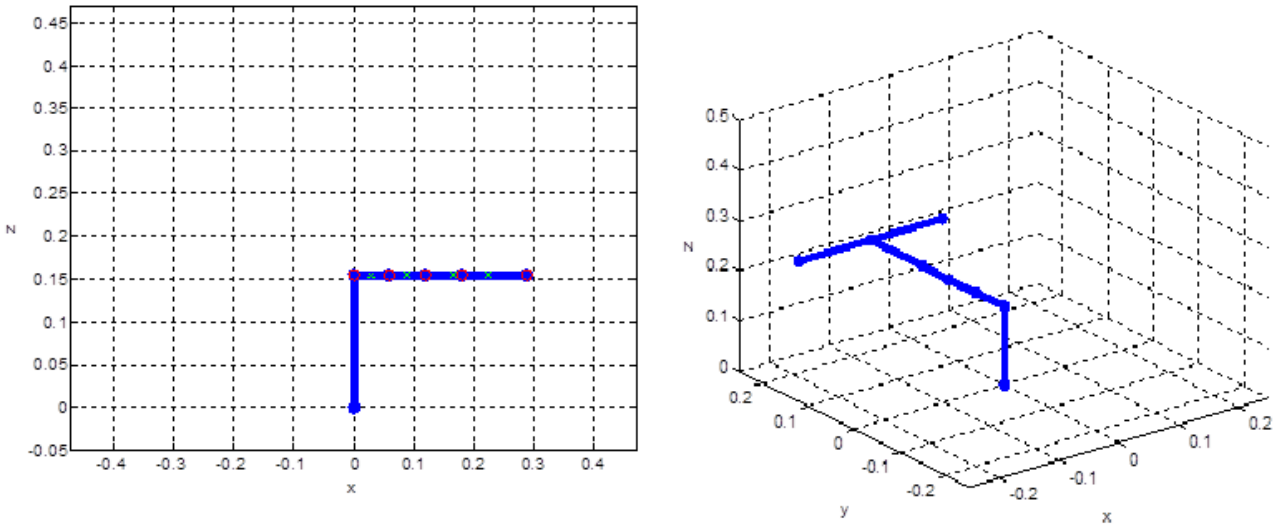


Figure 7: Arm Torque Simulation - Position

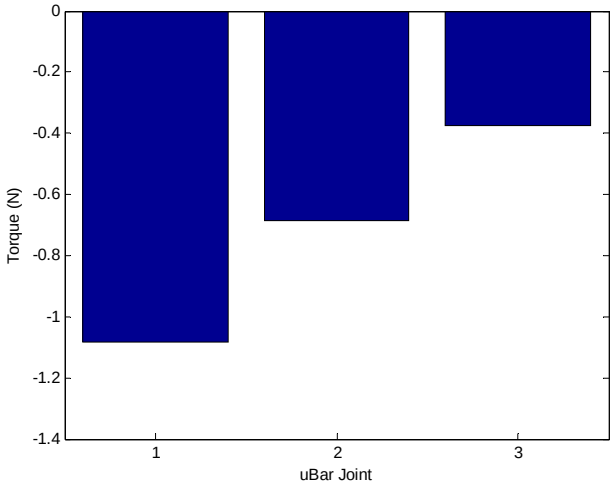


Figure 8: Arm Torque Simulation - Magnitude

UBAR joint 1 corresponds to the bottom UBAR, or link 2. In the above figure this joint is at a 90 degree angle. The following two joints are 2 and 3 (links 3 and 4).

Technical Obstacles

Acting as a catalyst for system modeling and simulation, the system has exhibited inconsistent operation, along with steady state error and oscillation during initial path planning. Because of this, the project goal has been updated since Milestone 1 due to the poor system control. To correct the steady state error and oscillation, the addition of an integral controller has been considered as a corrective measure due to the PD controller design of the individual CKBot modules. After consideration however, we have decided that the main problem of inconsistent operation is most likely an inherent limitation of the underpowered servo motors and consequently could not be corrected through the application of an integral controller. Instead, system model parameters were acquired for use in kinematic and dynamic equations characterizing the system as described in the project discussion section. These equations have allowed path planning simulation with an emphasis on torque minimization to be designed which will help correct the inconsistent operation of the motors due to excessive loading with specific motions. With this process we also hope to minimize the situations leading to oscillation.

Steady state error which was attributed to the effects of loading on the system and the lack of an integral controller was resolved through the calibration of each servo at predefined locations while loaded to maximize control while under operation. Slight error is induced while unloaded, but this does not negatively affect the placement of nodes and trusses as it would if the system exhibited the error while carrying a load.

In addition to the underpowered servos necessitating precise and calculated torque minimization path planning, there are a few other notable technical obstacles uncovered at this stage of development. Hardware wise, there are a few problems and limitations discovered. First, the contact switches do not all activate with the pressure of the nodes and trusses. This relatively simple problem can be fixed through spacers to ensure contact after placement. Second, the base of the robotic arm which allows rotation through 270 degrees has been lifting under the torque of the motor, allowing the gears between the motor and swivel base to slip. This problem was resolved through the liberal addition of tape between the motor housing and factory floor to keep the gears from misaligning.

With respect to software, there are still a few issues yet to be resolved. We recently moved from the Linux and IPython environment to the CKBot GUI in Windows. This transition allowed the calibration of the CKBot's and also allows easy path planning changes and communication. A few issues encountered however, are that commands are limited to only 14 per program, and the system crashes randomly at times. This requires a reboot at both the hardware and software ends to re-establish communication. We are currently actively searching for a solution, while at the same time considering returning the Linux environment after simple path planning with the torque minimization so that we can more easily transition to CCL when the time comes.

Finally, communication with UPenn MODLAB regarding robotic characteristics and control issues has been hit or miss. The simple fact that this is an experimental project with limited documentation and in a currently evolving state lends itself to certain difficulties. The most relevant unresolved issue pending response from UPenn is control of the servo motor velocity and pinging for the current location and state of the motor. For instance, at this time, there is no way for us to know the current position of the motors. This is required for our low level feedback control to work and is thus vital to the control system and our project goals.

Thus, our current open issues are as follows:

Inconsistent operation – Same command, same resource, different path trajectory
Oscillation – The robot reaches a position and will oscillate indefinitely
Contact Switches – Not all contact switches activate
Communication Failures – Communication freezes and requires software and system reboot
Motor State – How to get current position and control velocity in the Linux environment

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 MS1. In general, Stefan Kristjansson has primarily been involved with Phidget I/O integration and communication and converting from Linux to Windows for path planning and calibration procedures. Richard Wood has also been involved heavily in the Windows conversion, as well as path planning and helping with the torque minimization procedure and programming. Andrew Lawrence has been focused solely on the kinematic and dynamic equations and characterization and working to implement the torque minimization path planning simulation.

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.