Load Balancing for Multi-Robot Construction

Nils Napp, Eric Klavins
{nnapp,klavins}@uw.edu
University of Washington
Seattle, WA

Abstract— **In distributed multi-robot construction it is important that different building sites receive building materials at fixed, relative rates. Otherwise, subtasks finish at different times introducing unnecessary delays. We present a feedback algorithm to achieve robust load balancing in routing building materials for stochastic, distributed, multi-robot construction systems. We express global behavior in terms of local reactive behavior via *Guarded Command Programming with Rates* and prove correctness of the load-balancing controller for a wide range of conditions. We adapt a proof from earlier work on controlling *Stochastic Chemical Kinetic* systems and illustrate the algorithm on the Factory-Floor robotic testbed [1].**

I. Introduction

In this paper we examine feedback control to balance loads between subtasks in multi-robot systems. In particular, we present a controller to balance the feed rate of building materials between different construction sites in a larger distributed construction task. Modular, reusable programs for construction subtasks are important to managing complexity and achieving scalability in writing new construction programs. In this context a simple, robust way to accomplish load balancing between subtasks is important to composing large construction programs from smaller ones without introducing unnecessary delays. The work presented in this paper is another step toward the goal of building structures with autonomous, scalable, and robust robotic systems. Potential applications of such technology include building structures for space exploration, hazardous/inaccessible environments, or on a very small scale [2].

The load-balancing controller described here is robust to robot failures and other system changes, such as additional loads to the routing sub-task. We prove that the controller can balance loads in systems that are well-modeled by Markov processes and illustrate the approach on Factory Floor testbed (Sec. II) in the context of a particular way of representing behaviors (Sec. III).

In this paper, the behavior of robots is specified via *Guarded Command Programming with Rates* (GCPR), which specifies the local reactive behavior of robots and allows the system as a whole to be interpreted as a Markov process. Modeling the behavior of multi-robot system as a stochastic process is similar to [3], [4], [5], [6]. Modeling multi-robot system behavior as a Markov process restricts the type of timing behavior one can specify due to the inherent randomness. However, it allows the application of various analysis tools as well as reasoning about reliability. For GCPR the Markov process interpretation also enables reasoning about relative speeds and *concurrency*, the tendency of distributed system to work with a common resource at the same time.

This work is supported by NSF Grant 0735953: EFRI *Controlling the Autonomously Reconfiguring Factory* and AFOSR via the 2006 MURI Award *Specification Design and Verifcation of Distributed Embedded Systems*.

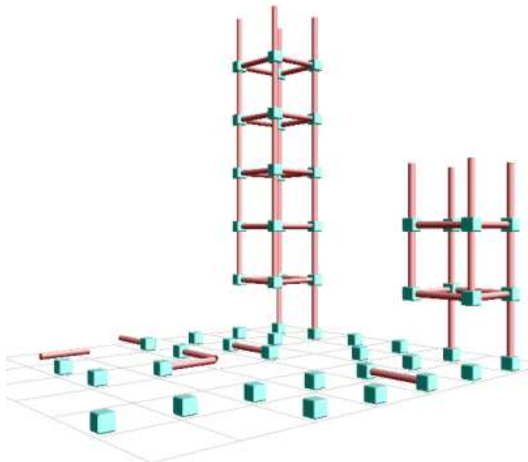

Fig. 1. Simulation of a decentralized multi-robot system, building two towers. Each square on the bottom layer represents a robotic module that can manipulate the two types of building materials trusses (red) and nodes (blue). Here, the construction area in the back receives building materials at twice the rate as the one in front. As a result the back tower is higher.

While using probabilistic models and behavior can seem counterintuitive for engineering robust, reliable systems, consider the exceedingly successful TCP/IP protocol. When analyzing the performance and reliability this protocol is often modeled as a Markov process [7]. That packets eventually arrive with high probability is more important than detailed behavior, such as a particular route being predictable or fixed. Similarly, GCPR specifications are meant to guarantee that behavior has a high probability of success [8]. In this paper we build on these previous results and try to guarantee not only that programs eventually succeed, but that several sub-programs can be balanced, resulting in robust predictable transient behavior.

The contribution of this paper is a load-balancing feedback controller that works for stochastic programs in multi-robot systems. We show how this approach can be used to balance loads between multiple construction sub-programs, taking full advantage of the compositional nature that local reactive programs allow. The mathematical approach is to reformulate load balancing into the equivalent problem of controlling the average species number in a Stochastic Chemical Kinetics (SCK) model [9] and to apply results we developed [10] (Sec. IV).

Section II describes the Factory Floor robotic testbed. Section III introduces mathematical notation used for programming, SCK, and summarizes some key results about Markov processes. Section IV describes a particular construction program in more detail and describes the load-balancing controller. It also gives some illustrative simulation results to accompany the proof. Finally, Sec. V contains some concluding thoughts and ideas for future research.

II. The Factory Floor Testbed

The Factory Floor testbed is a multi-robot system for developing scalable, robust, multi-robot construction algorithms [1]. It consists of identical modules arranged in an array, reminiscent of a factory floor with many workers. Together, these modules assemble building materials into a layer that is then lifted. By repeating this process, arbitrary lattice structures can be extruded from the Factory Floor, layer by layer. The structures are built from two different types of building materials, *nodes* and *trusses* (Fig. 2c and b). For the remainder of the paper, both types of building material are refered to as *raw mataerial*. Nodes have six faces, each of which can rigidly connect to the end of a truss. Together, the two raw materials can be used to construct arbitrary three dimensional lattices.

Each robotic module contains a manipulator, a lifting mechanism, a cradle to store nodes, and various structures to help with alignment (Fig. 2a). The end effector of the manipulator can pick up and release both trusses and nodes. In addition, the end effector can actuate a latching mechanism on the trusses, so that the ends become rigidly attached to nodes (Fig. 2b).

Each robotic module can communicate with its four neighbors and exchange information about the presence of raw materials and the state of the lifter. In this paper we program the system by describing processes that run each of the modules. Robots only talk to their neighbors to check the presence of building materials or exchange simple messages. No robot has access to the global system state and no robot tries to estimate it. In our opinion such restrictions about using global information are important to achieving scalable systems. The exception to this paradime is the integrator state (Sec. IV-C), which all robots have access to. Since the amount of information shared information is low we assume that there is a low-level, distributed coummunication scheme to share this information.

The programs for the example presented in Fig. 1 and the rest of the paper are written in the Command and Control Language (CCL) [11] (see Sec. III) combined with an external simulation library. It keeps track of the simulated physical state of the positions of lifting mechanism, nodes, and trusses. Figure 2de show the physical system and the same state represented in the simulator.

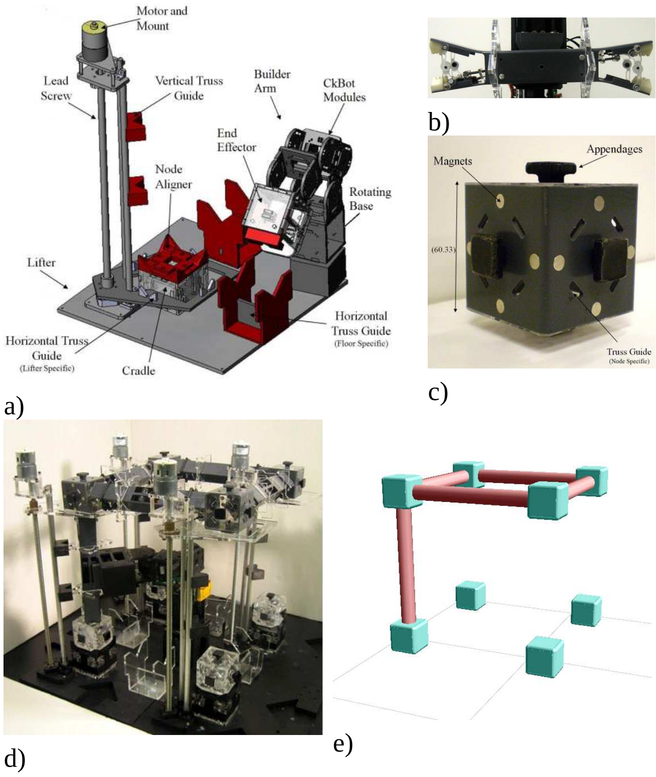

Fig. 2. The Factory Floor testbed. a) Schematic representation of a Factory Floor module. b) Picture of truss type raw material. c) Picture of node type raw material. d) Picture of four modules assembling a two layer structure. e) Picture of same structure in simulator. Module components are omitted for clarity.

In any case, we assume that the testbed has low level drivers that can arbitrate local resource conflicts so that the high level guarded command programs we write can treat the actions of Factory Floor modules as atomic operations between discrete states.

Example 1. For notational clarity consider only the lowest level of the Factory Floor testbed, and one type of the raw material, nodes for example. In this case, the *state space* S is the occupancy information of each module, which can conveniently be represented as a binary number. Using this binary notation a state $s \in S$ assigns a zero or one to each module. If the i-th digit $s_i = 1$ then module number i contains a node, otherwise if $s_i = 0$ it does not. Figure 3 shows an example layout of module indices. Even in this simplified model, the number of state is quite large, $2^{42} \approx 4.4 \times 10^{12}$. ■

The same approach can be used for an arbitrary, finite number of modules and occupancy states. For discussing routing programs this simplified state space suffices, but the following applies equally well to arbitrary, finite, discrete state spaces.

III. Notation

A. Guarded Command Programming with Rates

GCPR is a way to program reactive, concurrent systems, such as the Factory Floor testbed described in Sec. II. Programming systems with GCPR has the advantage of simple program *composition*. It is an extension of an approach to reasoning about parallel processes from the computer science literature [12]. Each robot works independently except sharing common resources, here, physical building materials and free space. GCPR and other concurrent languages are designed to make these dependencies and resource conflicts explicit, so that they can be formally reasoned about. The examples in this paper are written using the Command and Control Language (CCL) [11], a particular implementation of a GCPR language.

The idea is that each robot is represented by a *program*, or process that can make changes to a global state space S. In particular, we are interested in programs where these actions are *local*, which means that each robot only interacts with its neighbors. For a more detailed discussions of local behavior see, for example [13].

A program Ψ is a multiset of *rules*. A rule ψ is a triple (g, a, r) where $g \subset S$ is a called the *guard*, $a \subset S \times S$ is a called the *action*, and $r \in \mathbb{R}^+$ the rate.

Guards are conditions on the state space S; in the case of robotic systems these conditions often corresponds to a combination sensor information and internal variables. For example, a guard might be whether or not a robot is holding a raw material. An action corresponds to a change a robot can perform on the state space, like moving a raw material. For each $a \in S \times S$ the first coordinate of a corresponds to the state before the action and the second coordinate to the state after the action is performed. The rate associated with each rule determines the average frequency the rule executes when the guard is satisfied, in a way that is made precise in Sec. III-B.

Example 2. The guarded command (g, a, k) of module i passing a node to module j at a rate k is given by the guard

$$g \equiv \{s \in S \mid s_i = 1, s_j = 0\}$$

and the action

$$\begin{aligned} a \quad &\equiv \{(s, s') \in S \times S \mid s_i = 1, s_j = 0, \\ &\qquad s'_i = 0, s'_j = 1, \forall k \neq i, k \neq j \ s_k = s'_k\}. \end{aligned}$$

Only the occupancy of module i and j change and all the other digits of s remain unchanged. The action is local. ■

The notation in the above example is cumbersome and we borrow ideas from chemistry to simplify it. The chemical notation also highlights the local nature of guarded commands since parts of the state that remain unchanged do not appear.

Example 2 can be rewritten as

$$s_i \bar{s}_j \overset{k}{\rightharpoonup} \bar{s}_i s_j, \tag{1}$$

where s_i is used to denote the condition that $s_i = 1$ and $\bar{s}_i$ for $s_i = 0$ similar to boolean logic. In the chemical reaction notation the guard appears instead of *reactants*, the result of the action instead of *products*, and the rate over the arrow. Parts of the state that do not show up in the product remain unchanged. With this notation a guarded command program is essentially a reaction network with only unary reactions in the SCK model [9]. Both of these descriptions specify Markov processes on the state space as described in the next section.

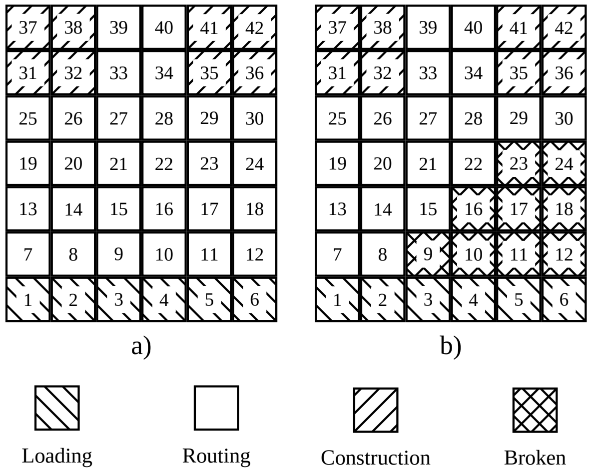

Fig. 3. Layout of construction program. The modules that run the loading program also run the routing program. Construction area 1 consists of modules 31, 32, 37, and 38, while construction area 2 consists of 35, 36, 41, and 42. The routing program is the same as for all modules except for the border modules of the two construction areas as shown in Fig. 5. a) Programs during normal operations. b) Failure scenario. Broken modules do not pass any raw materials.

Programs can be *scaled* and *composed* to create new programs. Together, these two operations are used in the remainder of the paper to build increasingly complex programs from simple sub-programs and to phrase desired program behavior as a control problem. The composition Ψ of two programs Ψ_1 and Ψ_2 is simply the union of their rules

$$\Psi \equiv \Psi_1 \cup \Psi_2. \tag{2}$$

A program Ψ scaled by a positive number $\alpha \in \mathbb{R}^+$ is a new program with rules that have the same guards and actions, but where each rate is scaled by α

$$\alpha\Psi \equiv \{(g, a, \alpha r) \mid (g, a, r) \in \Psi\}. \tag{3}$$

Note that in this construction the guard and action need to be consistent in order for the rule to specify transitions between states. Specifically, for guarded commands to induce transitions the intersection of the guard and the first component of the action has to be non-empty. Otherwise, the rule does not change the behavior of the associated Markov process.

B. The Connection Between GCPR And Markov Processes

Semantically, a *Guarded Command Program* (GCP) is interpreted as a continuous time *Markov process*, [14], X_t with state space S. Markov processes are characterized by their generators, which for finite or countable state spaces can be represented as a *generator matrix*. We briefly describe a mapping from GCPs to generator matrices and the correspondence of scaling (3) and composition (2) operations on programs to operations on the associated generator matrices. For a more detailed description of these operations and proofs see [8], [10].

To facilitate the description of the generator matrix $\mathbf{Q}$ we choose an enumeration of S. This enumeration is arbitrary but assumed fixed for the remainder of this paper. For a given program denote the set of rules in which the guard contains i and the action includes a transition from i to j, by $R_{i,j}$.

Given a guarded command program Ψ define the associated generator matrix $\mathbf{Q}$ element-wise as

$$\begin{array}{ll} Q_{i,j} = 0 & \text{if } i \neq j \text{ and } R_{i,j} = \emptyset, \\ Q_{i,j} = \sum_{\psi \in R_{i,j}} r_\psi & \text{if } i \neq j \text{ and } R_{i,j} \neq \emptyset, \\ Q_{i,i} = -\sum_j Q_{j,i}. & \end{array} \tag{4}$$

Probability distributions over states are denoted as row vectors $\mathbf{p}$, where the i-th element p_i denotes the probability of being in state $i \in S$. Functions of the sates are denoted by column vectors. If $y : S \to \mathbb{R}$ then $\mathbf{y}$ is a vector who's i-th entry is $y(i)$, the value assigned to state i by y. The *expected value* or *mean* $\mathbb{E}y$ of a function y is given by the inner product of the two vectors representing the function and the probability distribution $\mathbf{py}$.

The dynamics of the Markov process are determined by the generator matrix $\mathbf{Q}$. The evolution of probability distribution is given by

$$\frac{d\mathbf{p}}{dt} = \mathbf{pQ}, \tag{5}$$

which is called the *master equation* [15, Ch. 5]. When $\lim_{t\to\infty} \mathbf{p}(t)$ is unique it is known as the *steady state* distribution. A sufficient condition for a system to reach steady state is that S is finite and every state can be reached from every other state. In general, we look at sub-programs where these conditions are met, so that we assume the existence of a steady state distribution.

Scaling and composition operations on programs are related to operations on the generators in the following way

$$\mathbf{Q}(\alpha_1\Psi_1 \cup \alpha_2\Psi_2) = \alpha_1\mathbf{Q}(\Psi_1) + \alpha_2\mathbf{Q}(\Psi_2). \tag{6}$$

Due to the close relationship between programs and their generator matrices, we identify the guarded command program with the stochastic processes they induce. For example, we say that a program has a steady state distribution and use program to mean the discrete Markov process on the discrete, often very large state space S. For a more detailed description of the probabilistic interpretation of programs as well as other operations on programs see [10].

IV. A Robust Load-Balancing Program

There are three main tasks in construction, obtaining raw materials, routing raw materials, and processing the raw materials into structures. In the Factory Floor testbed the modules perform all three. Figure 3 describes the different

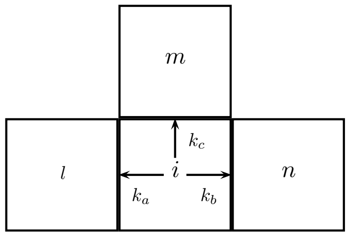

Fig. 4. Schematic representation of the routing program on a node. Each arrow represents a guarded command that passes raw materials in the direction indicates. The symbol next to the arrow is the associated rate. Tow of the rates are controlled by the input u.

regions of the factory floor and the different tasks they perform.

Obtaining raw materials in the following program is represented by the program Ψ_{load}. Modules that run Ψ_{load} randomly obtain raw materials at rate k_{load}. In general, the loading program could be replaced by a disassembly program that obtains raw materials by taking apart an existing structure of be the destination of a different routing program. Considering a single type of raw material as discussed in the Sec. II, the loading program can be written as

$$\overline{s}_i \overset{k_{load}}{\rightharpoonup} s_i \tag{7}$$

for each module i that is running Ψ_{load}.

The routing portion of a construction program is similarly simple. The spatial relation between different modules in the routing sub-program is shown in Fig. 4. Passing a raw material from one module i to a neighboring module m at a rate k_c, for example, can be written as

$$s_i\overline{s}_m \overset{k_c}{\rightharpoonup} \overline{s}_i s_m. \tag{8}$$

There are subtle differences between the routing sub-program and the loading sub-program. Firstly, because two different modules are involved, the same source modules could be involved in multiple passing operations so that the routing module has a choice on which way to pass the raw material. This choice is made probabilistically based on the relative rates of the passing rules (8). Secondly, physically passing raw materials between modules takes time, in the case when a module can pass a raw material to other modules we make sure that the rate out of each module is normalized to a maximum k_{pass}, corresponding to the maximum speed at which modules are physically able to perform the passing operation. Adjusting the rates for the different directions changes the relative feed rates to the two building sites in Fig. 3.

Building programs are more complicated to write due to coupled dependencies of raw materials, geometric constraints, and limited sensing of upper layers. Conceptually, these programs are just as simple though. In the programs presented here, each layer in the building program starts by placing a node in a specific location and then fills up the remaining positions with raw materials received from the routing sub-program until the layer is complete.

Building programs in the simulation use some simplifying assumptions. For example, in the simulator an individual module can lift a layer. In reality some coordination needs to take place. For example, a the module that decides it is time to lift a layer could send out a message with a time to count down, until starting to lift. Based on the size and inter module communication speed, the waiting time can be chosen conservatively so that all modules that need to lift together have received the message with high probability by the time the count down elapses.

Similar to our earlier work [8] this paper is mostly concerned with the routing portion of an overall construction program, specifically with robustly routing raw materials to different construction sites in a balanced way. The routing sub-program itself is the composition of three different sub-programs Ψ_C, Ψ_A, and Ψ_B. The program Ψ_C moves raw materials up into the direction of the construction areas, k_c in Fig. 4, and the other two program move raw materials sideways to the construction areas, program Ψ_A toward construction area 1 and Ψ_B toward construction area 2. The resulting routing program is given by

$$\Psi_{route} = \Psi_C \cup u\Psi_A \cup (1-u)\Psi_B, \qquad (9)$$

where the rules of each program have a rate of $\frac{k_{pass}}{2}$. The rules for border modules are are modified from Fig. 4 as shown in Fig. 5.

The overall program is given by

$$\Psi = \Psi_{load} \cup \Psi_{route} \cup \Psi_{tower1} \cup \Psi_{tower2}, \qquad (10)$$

where Ψ_{tower1} and Ψ_{tower2} are the construction programs in area 1 and area 2 respectively. The two construction programs are the same except that they are instantiated on different modules. Note, that this program has a tunable input u that determines how much of the raw material is routed to each construction site on average.

A. Disturbances

The two disturbances we consider here are failure of individual robotic modules and changing loads. For a more complete discussion of different disturbances and failures in the context of GCPR and the Factory Floor testbed see [10].

Balancing loads between different construction areas in the face of module failures, makes the overall construction program robust. For example, in Fig. 3b, 9 out of 34 routing modules are broken. As a result, both feed rates to the construction areas change because the routing and loading programs are much less efficient at transporting raw materials to the construction areas. However, the load-balancing controller regulates the average difference between the two feed rates to zero, see Fig. 6.

Rejecting the disturbance of new or changing loads is important to utilizing the compositional nature of GCPR. In the absence of a load-balancing mechanism routing sub-programs need to be specifically tailored to the construction

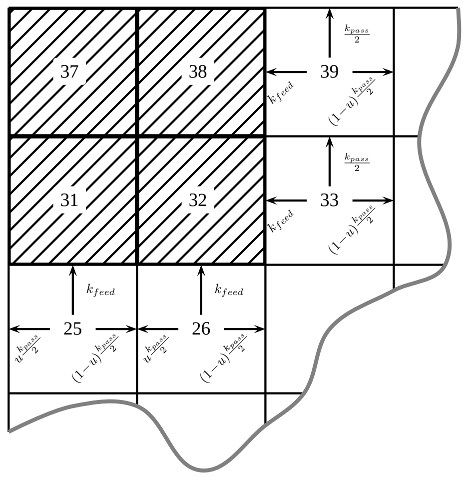

Fig. 5. Border modules of construction area 1. The expected value of the feed rate is a into the target area is a linear function of to the expected occupancy time for the border modules. The modules 25,26,33, and 39 passes a raw material (node or truss) into the target area at rate k_{feed}. If $\mathbb{E}s_i$ denotes expected value of module i possessing a raw material, then the expected feed rate is $k_{feed}\ \mathbb{E}s_i$.

sub programs they are feeding, which stands in the way of building up large programs form re-useable sub-programs.

Also, here we can consider robust behavior with multiple construction sites, while our previous results concerned creating robust open-loop routing programs that feed only a single construction area. While conceptually minor, the proof in [8] that the routing program is robust to a large class of failures relied on assuming a single construciton area.

B. Computing The Expected Feed Rate

The occupancy of the *border modules* to a construction area is directly related to the *feed rate* of raw materials into the area, see Fig. 5. Only the occupancy state of border modules matters, since raw materials can only enter a construction area through one of the border modules.

Let i be the location of a border modules that passes the raw material into the construction area at a rate of k_{feed}. For a given state $s \in S$, the i-th coordinate s_i denotes whether that particular module contains a raw material $(s_i = 1)$ or not $(s_i = 0)$. The expected feed rate from module i is given by

$$\mathbb{E}k_{feed}s_i = k_{feed}\mathbb{E}s_i.$$

Lemma 1. Given a construction site and a routing program where each module $i \in B$ in the set of border modules B passes raw materials into the construction site with rate k_{feed}, the expected feed rate is given by $\mathbb{E}y$ where $y : S \to \mathbb{R}$ by

$$y_B(s) = \sum_{i\ \in B} k_{feed}s_i.$$

Let y_1 be a function defined as in Lem. 1 for the border modules of construction area 1 $B = \{25, 26, 33, 39\}$, and similarly, let y_2 be defined for construction area 2 with $B = \{29, 30, 34, 40\}$. Due to linearity of the expected value, the mean difference of the two feed rates is give by the expected value of

$$y(s) = y_2(s) - y_1(s). \tag{11}$$

Load balancing between the two construction areas can thus be expressed as the set-point $\mathbb{E}y = 0$.

C. Feedback Control

The feedback controller is an integral controller, on the error of an output function $y : S \to \mathbb{R}$ of the discrete states compared to a set-point, y^*. The cumulative error is denoted by z and its dynamics are given by

$$\frac{dz}{dt} = \gamma(y(s) - y^*), \tag{12}$$

where γ is a integrator gain. The cumulative error is fed back though the saturation function

$$h(z) = \begin{cases} 0 & z \leq 0 \\ z & 0 < z \leq 1 \\ 1 & 1 < z. \end{cases} \tag{13}$$

Denote the steady state distribution of the open-loop system when $u = 0$ and $u = 1$ by $\mathbf{p}_m$ and $\mathbf{p}_M$ respectively. The closed-loop system dynamics are given by $u = h(z)$, where the dynamics of z depend on u because it changes the transition rates between states in S. The resulting closed loop system is a stochastic hybrid system with the following property.

Theorem 2 (From [10])**.** Let $\mathbf{Q}(u)$ be the generator matrix of a tunable reaction network and $\mathbf{y}$ the vector corresponding to an output function. The feedback controller described in (12)-(13) results in a closed loop system with a stationary distribution that has $\mathbb{E}y = y^*$ when y^* is in the *controllable region*, $\mathbf{p}_M\mathbf{y} < y^* < \mathbf{p}_m\mathbf{y}$.

To apply Thm. 2 to the load balancing problem let

$$\mathbf{Q}(u) = \mathbf{Q}(\Psi_{load} \cup \Psi_C \cup u\Psi_A \cup (1-u)\Psi_B \cup \Psi_{tower1} \cup \Psi_{tower2})$$

and the output function defined by (11). Choosing $y^* = 0$ and using the controller presented above results in steady state behavior where the mean feed rate to both construction areas is the same.

The simulation results of the open-loop and closed loop system are shown in Fig. 6. In both cases the system is operating correctly according to Fig. 3a for the first 1000*sec* at which time them modules marked broken in Fig. 3b stop passing raw materials.

The input u is chosen such that the open-loop system with all modules working correctly has a balanced feed rate, Fig 6a $t < 1000sec$. However, when the failure scenario shown in Fig. 3b occurs, more raw material arrives at construction area 1 on average. The load-balancing controller rejects this disturbance and even when modules break the mean feed rate to both construction areas is the same.

The controllable region depends on the loading rate, the detailed geometry of the routing program, and the location of broken modules. In the absence of bottlenecks in the routing area, the difference in loading rates is limited by the total available flux of raw materials arriving at the loading area. However, when bottlenecks are present as in the failure scenario shown in Fig. 3b they limit the total available flux difference, see Tab. I.

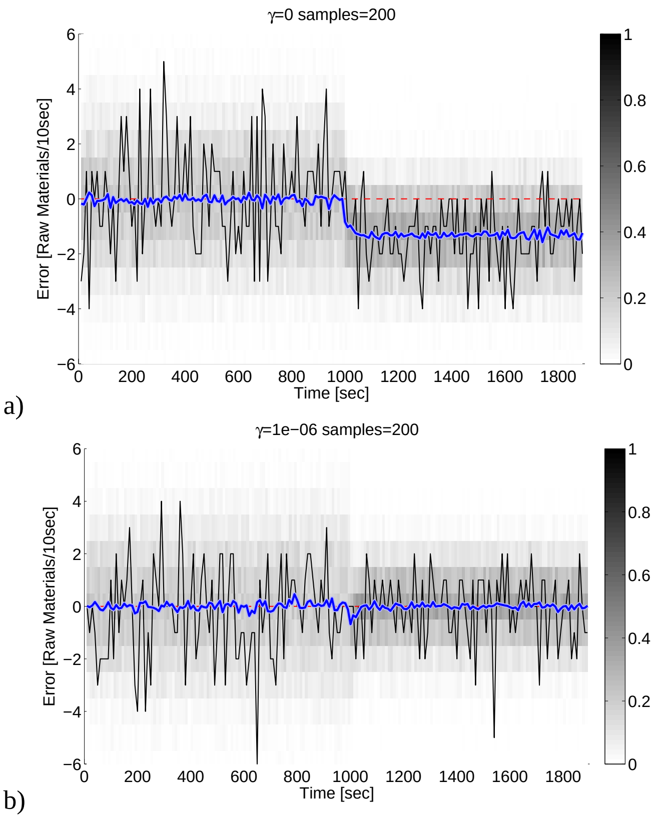

Fig. 6. Routing raw materials with the failure shown in Fig.3b at 1000sec. a) Difference in routing rates in open-loop. The fraction of trajectories at a particular value and time is indicated by the grey shading, showing the noise in the system. A sample trajectory highlighted in black and the thick blue line is the empirical ensemble mean. b) Difference in routing rates for the closed-loop controller with $\gamma = 1e - 6$.

The controllable region for a given geometry is easy to compute. All that is needed is an estimate for the steady state occupancy distribution, which can be done analytically for smaller problems or thorough sampling techniques like the stochastic simulation algorithm [16] even for large state spaces. Table I shows estimates of the controllable region based on statistics gathered from sample trajectories to compute $\mathbf{p}_m\mathbf{y}$ and $\mathbf{p}_M\mathbf{y}$. Note that when modules are broken the region shrinks significantly.

	$u = 1$	$u = 0$
Working	-3.90 ± 0.24	3.56 ± 0.23
Broken	-2.30 ± 0.18	0.71 ± 0.18

TABLE I

ESTIMATES OF THE CONTROLLABLE REGION (CI 0.95)

By Thm. 2 the load-balancing controller will work as long as the set-point (zero) is inside the controllable region. Using different programs can change this region, but as long as the new controllable region contains the set-point the balancing controller will balance the feedrates. As a result, the load-balancing controller is robust to a wide class of disturbances. For example, if the program also passes raw materials to additional construction sites.

V. Conclusion

The contribution of this paper is to adapt a feedback mechanisms from earlier work [17] to balance loads between different sub-programs in distributed, stochastic, multi-robot systems. The key connection is given in Lem. 1. We demonstrate the controller in a robust load balancing application for routing raw materials to different construction sites at the same feed rate.

The ability to robustly balance loads between different sub-programs is important to a modular approach to writing programs for multi-robot construction. This work is an extension of [8] which focuses on robust routing algorithms for a single construction site. Allocating the flow of raw materials to different sub-programs allows for building complex behaviors from sub-programs that have predictable behavior.

In the current analysis the feedback controller requires a global error and global feedback signal. However, the amount of information that needs to be shared is minimal. As a result, this controller can readily be implemented even in robots with very limited communication ability. In the future we plan to investigate distributed implementations as well. For example, each module could run a local copy of the feedback controller and only error signals need to be computed and broadcast.

Creating robust, scalable construction sub-programs is significantly more difficult than creating robust routing programs, but necessary to reach our goal of creating robust, autonomous multi-robot construction platforms. We plan to address this problem by carefully designing a set of robust basic construction programs, which can then be scaled and composed to build complex structures. However, to take full advantage of such robust construction sub-programs robust routing and load balancing are essential. As such, we believe that the controller presented in this paper is a significant step toward robust, distributed, multi-robot construction.

References

[1] K. Galloway, R. Jois, and M. Yim, "Factory floor: A robotically reconfigurable construction platform," in *IEEE International Conference on Robotics and Automation (ICRA10)*, (Anchorage AK), pp. 2467–2472, 2010.

[2] M. Yim, W. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. Chirikjian, "Modular self-reconfigurable robot systems," *IEEE Robotics Automation Magazine*, vol. 14, pp. 43 –52, march 2007.

[3] I. Chattopadhyay and A. Ray, "Supervised self-organization of homogeneous swarms using ergodic projections of Markov chains," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 39, pp. 1505–1515, Dec. 2009.

[4] L. Matthey, S. Berman, and V. Kumar, "Stochastic strategies for a swarm robotic assembly system," in *Proc. IEEE International Conference on Robotics and Automation ICRA '09*, pp. 1953–1958, 2009.

[5] S. Berman, Á. Hálász, and A. Hsieh, "Optimized stochastic policies for task allocation in swarms of robots," *IEEE Transactions on Robotics*, vol. 25, pp. 927–937, Aug. 2009.

[6] E. Klavins, S. Burden, and N. Napp, "Optimal rules for programmed stochastic self-assembly," in *Robotics: Science and Systems II*, (Philadelphia, PA), pp. 9–16, 2006.

[7] J. P. Hespanha, "Stochastic hybrid modeling of on-off TCP flows," in *Stochastic Hybrid Systems: Recent Developments and Research Trends* (C. G. Cassandras and J. Lygeros, eds.), no. 24 in Control Engineering Series, pp. 191–219, Boca Raton: CRC Press, Nov. 2006.

[8] N. Napp and E. Klavins, "Robust by composition: Programs for multi robot systems," in *ICRA Proceedings*, (Anchorage), pp. 2459–2466, 2010.

[9] D. A. McQuarrie, "Stochastic approach to chemical kinetics," *Journal of Applied Probability*, vol. 4, pp. 413–478, Dec 1967.

[10] N. Napp and E. Klavins, "A compositional framework for programming stochastically interacting robots," *In Review*, 2011. http://soslab.ee.washington.edu/nnapp/ijrrRev.pdf.

[11] E. Klavins and R. M. Murray, "Distributed algorithms for cooperative control," *Pervasive Computing, IEEE*, vol. 3, no. 1, pp. 56–65, 2004.

[12] E. W. Dijkstra, "Guarded commands, nondeterminacy and formal derivation of programs," *Commun. ACM*, vol. 18, no. 8, pp. 453–457, 1975.

[13] B. Smith, A. Howard, J. McNew, and M. Egerstedt, "Multi-robot deployment and coordination with embedded graph grammars," *Autonomous Robots*, vol. 26, pp. 79–98, 2009.

[14] D. W. Stroock, *An Introduction to Markov Processes*. Graduate Texts in mathematics, Springer, 1st ed., 2005.

[15] N. V. Kampen, *Stochastic Processes in Physics and Chemistry*. Elsevier, 3rd ed., 2007.

[16] D. T. Gillespie, "Stochastic simulation of chemical kinetics," *Annual Review of Physical Chemistry*, vol. 58, no. 1, pp. 35–55, 2007.

[17] N. Napp, S. Burden, and E. Klavins, "Setpoint regulation for stochastically interacting robots," in *Robotics: Science and Systems*, 2009. http://soslab.ee.washington.edu/nnapp/rss09.pdf.