2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

# The Process Information Space: The Importance of Data Flow in Robotic Systems

Aaron Morris
Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
Email: acmorris@andrew.cmu.edu

***Abstract*— Data flow describes the motion of information through a robotic system. As this work investigates, data flow can also provide system-critical information about the robot that is useful for failure recognition, fault recovery and improved robot dependability. The approach developed in this paper applies a planning formulation to the observation and control of data flow through the reconfiguration of robot process connections. Simulated results are presented and a subterranean robot application is discussed.**

## I. Introduction

Data flow describes the motion of information through a system: a temporal data stream that moves, collects, and transforms from one computational process to another. In a robot, this flow begins at the sensory level and finishes with actuation. As such, the tendency when programming a robot is to focus intently on the causal relationship between sensory data and action. For example, how does a particular pixel intensity or range measurement effect whether or not a mobile robot moves forward? While the link between sensor data and actuator is undeniable, this research is less concerned about the actual data and more interested in observing the effects that data induce upon a computational system. Thus, this paper describes how process information can be extracted and exploited to improve robot decision-making in uncertain working conditions.

For this work, the term process denotes the running instance of a program within a robot's computational framework. A robot will have a number of processes, which are divided into three classes: *sensors*, *inters* and *motors* (Figure 1). Sensor processes interface with sensor hardware, inter-processes interface with other processes and motor processes interface with hardware devices such as actuators, speakers or displays. Each process may have multiple inputs for a single output, although this output can take the form of a data structure with multiple values. Data flow is therefore identified as the creation of new output along a chain of processes that stretches from sensor to motor.

To achieve this end, multiple, redundant variations of these process chains are defined to channel the robot's sensory data streams. These streams are thereby observed and controlled through *virtual* sensors and actuators placed inside the robot's computational system. A virtual sensor is sensitive to particular computational stimuli just as external sensors are

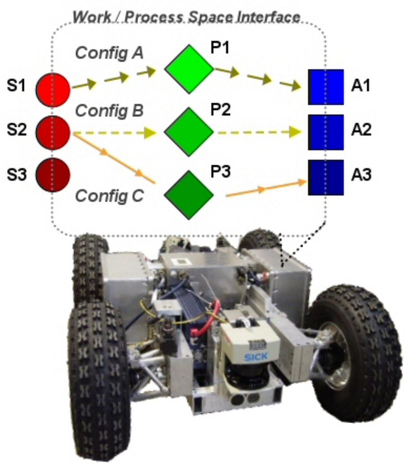

Fig. 1. Robots and data flow. Inside the computational framework of every robot resides three classes of processes: sensors ($S_i$), inters ($P_i$), and motors ($A_i$). These processes can be linked to form process configurations, which direct data flow from sensor to motor.

receptive to physical stimulus. In a similar context, a virtual actuator analogously acts upon a computational system as an electro-mechanical actuator would create action in the physical world. Together, these virtual mechanisms form the basis of a scheme that regulates data flow throughout a robotic system.

Why is data flow regulation important? Flow regulation is a way to estimate and change the state of the robot without dealing directly with large streams of raw and processed sensor information. For example, imagine that a scanning laser-range sensor feeds a series of processes responsible for navigating a robot. If this sensor should fail, every process upstream from this data source can no longer function as expected. In the best case, each upstream process may recognize the corrupted or absent data and shutdown; however, a system shutdown is not the preferred option for a robot operating beyond the reach of human assistance. A robot in such a situation requires a redirection of sensor information: that is, an alternative source of sensory data that can sustain the navigation system until the robot is recoverable. By watching data flow, identification and mitigation of such problems is feasible without the overhead of direct data analysis.

Data flow regulation also offers potential utility for improving the selection of robot behaviors over time. If, for example, the aforementioned mobile robot has a process that identifies obstacles from laser-range measurements, knowing that this process is not producing output could indicate no obstacles lie ahead of the robot along its intended path. Such knowledge may be useful for selecting between a cautious path-planning process and a fast reactive-steering process.

The application for this work extends to all robots regardless of the operational scenario; although, such a scheme is particularly valuable when a robot becomes unrecoverable upon failure. Environments such as space and the subterranean domain specifically fit within this profile. The subterranean world is of interest in this research since underground operation has tremendous application for robotic utility; however, the vast distance and limited communication between robot and human necessitates trustworthy robot autonomy [1].

## II. The Problem

The core problem addressed in this paper is in determining *when* and *how* to switch a robot's operational context. An operational context is a set of processes actively contributing to the robot's performance. These processes establish the data streams that flow through a robot's decision-making facilities. An alternative way to speak of this context is in terms of process configuration. Process configuration represents a linked series of processes that are running in the computational system at a certain instance in time. When a robot switches its operational context, this translates into a reconfiguration connections between processes.

The trigger for this switch can take any number of forms during robot task fulfillment. As mentioned in the introduction, the failure of sensor hardware is one instance that necessitates reconfiguration. Other triggers could include corrupted sensor data, data that violates assumptions made by a process, problematic code, or changes in the environment. Most of these triggers are related to an even more common problem among robotics: decision-making under uncertainty and incomplete information.

In order to clarify this problem, the following scenario is described for a mobile robot engaged in an exploration task. To align with the themes of subterranean robotics, imagine this robot is operating in an abandoned coal mine.

A robot, similar to the one pictured in Figure 1, is tasked to explore and map an abandoned mine corridor. This corridor, unmaintained for many years, stretches beyond the reach of radio communication and is filled with mud, mine equipment, explosive gases, and mine debris. Outfitted with scanning lasers, inertial sensing and gas detectors, this mobile robot navigates and maps the rugged mine terrain until a large obstacle blocks further progress into the mine, forcing the robot to return to the mine entrance. Soon after beginning its return, the robot encounters what seems to be another large obstacle, this one obscuring its path to the exit. Did something change in the environment? Did a sensor fail and allow a phantom obstacle to appear? Did a process fail or misinterpret the sensed world?

Without additional information regarding the robot's true state, the navigation system has no chance of disambiguating these scenarios. Even if the navigation system possessed the capacity to assess the situation, such knowledge would not achieve the robot's goal to exit the mine. What the robot can ascertain from its predicament, however, is that the current operational mode (e.g. the current set of active processes) is not facilitating the robot to escape the mine.

In this problem scenario, knowledge of data flow is key in diagnosing the situation and contributing to a possible solution. For example, if the problem were truly a laser failure, the sensor process along with all downstream inter and motor processes would essentially cease to generate valid output. On the other hand, if the laser remained operational, the data flow would restrict downstream toward a path-planning or obstacle-avoidance process. As a result, the first case requires alternative sensing or blind navigation to exit the mine whereas the second case requires switching to an alternative path-generation process or temporarily disabling the obstacle avoidance procedure.

Once a robot has encountered a "bad" configuration, how will it measure the quality of alternative configurations? History and prior information are inevitably needed to guide the selection process. For this reason, process information from virtual sensors must be collected and compiled into a format that can govern the process reconfiguration.

## III. Related Work

In general, sensory data decreases uncertainty regarding the robot's state; however as has been discussed in the problem scenario, there are some situations where sensor data or invalid assumptions can lead to increased uncertainty. Approaches to address uncertainty with regards to robot decision-making span everything from action without sensing to action based on constructed histories of all the information available to the robot. Research in sensor-less manipulation by Akella, Erdmann, and Mason [2] have shown how action models can be applied to reduce uncertainty without sensor feedback. A more classic approach to reasoning about action under uncertainty employs (POMDPs) [3], [4], as demonstrated by Pineau and Smith[5], [6] on robotic systems. Such methods attempt to utilize all available sensor, state, and action information; however, for this intense dependency on information, such methods tend to become intractable in practice, Roy [7].

More recently, many of the methods that deal with uncertainty are converging into unified theory of planning with uncertainty, as described by LaValle [8]. Known as the *information space*, this formulation is tailored for problems that involve reasoning under ambiguous sensing. Evidence to support this claim is demonstrated in the information space's ability to represent a large class of problems including those that estimate robot state as well as those that need no state information whatsoever. As such, the theoretical formulation of the information space presents a natural way to describe problems that involve uncertainty and will be utilized in this work.

If dealing with uncertainty is the problem, data stream observation and manipulation is the solution offered in this paper. Popular approaches to exploit reconfiguration among processes are the ASyMTRe-D architecture by Parker and Tang [9], Networked Robotics by McKee and Schenker [10] and OCP by Wills [11]. ASyMTRe-D is a multi-robot approach for coalition formation by sharing sensor information across teams of robots. In this respect, ASyMTRe-D can be viewed as a single robotic entity that reconfigures its process structure to meet the specifications of its tasks. Networked Robotics is also an architectural paradigm designed to share resources across robotic teams through explicitly define resource configurations where resources represent sensor, inter and motor processes. OCP stands for *open control platforms* and is designed to coordinate complex interaction among software components in real-time.

All of these paradigms describe the benefits of reconfiguration (robustness to failures, architecture abstraction, adaptability, etc.) thereby supporting the process reconfiguration concept of this work. Unlike this research, however, the key research problem they address relates to task description as configuration, rather than observation of configuration for failure diagnosis and recovery. In addition, ASyMTRe-D and Network Robotics are not designed for an individual robotic systems and none relate the nature of reconfiguration to a problem of uncertainty.

Finally, prior research in the field of subterranean robots has significantly contributed to the development of this paper's concepts. In particular, a mobile robot named Groundhog utilized a small set of operational modes (i.e. process configurations) to autonomously map an abandoned mine near Pittsburgh, PA[12]. These operating modes included a path planning navigation scheme, a wall following behavior, and a risky, albeit affective, scheme that blindly engaged the robot's motors. Such simple and diverse schemes proved to be essential in recovering the robot from trouble. The approach presented in this paper generalizes this scheme into a broader framework.

## IV. Approach and Formulation

Process observation and reconfiguration is formulated as a search/planning problem over the space of possible process configurations. That being said, a process configuration is defined as the specific binding of individual processes to one another. Recall from the introduction that processes are divided into three categories: a sensing and/or perception process $S$, an inter process that takes the form of a planning, behavioral or reactive process $P$, and a controller or actuator process $A$. Each process will have a specific interface that will allow them to connect to particular classes of processes $S, P, A$ or the environment $E$.

Figure 2 provides an example of process configuration and illustrates how configurations project actions into the

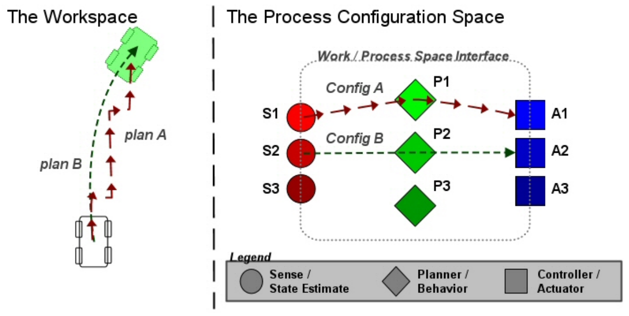

Fig. 2. A simple example of process configuration. Sensor processes are circles, inter processes are diamonds, and motor processes are squares.

workspace of a robot. On the right side of this figure, three processes of each class define the robot's configuration building blocks. The upper row of each class represents processes capable of handling large data streams whereas the lower processes handle thinner data streams. For this example, assume that configurations are constrained to single data paths every time step of the robot's operation. Also, assume that class $S$ processes can only bind to $P$ and $P$ to $A$. Altogether, this robot has 27 possible configurations.

On the left side of Figure 2 is a mobile robot defined in a 3D workspace (e.g. 2D position and heading). The processes configurations, as seen of the right, enable the robot to move in this workspace. Depending on the configuration, however, a variety of behaviors is possible. For example, $Config_A = S_1 : P_1 : A_1$ describes a heavy data flow where $S_1$ outputs all the robot's sensory data to a sensor integration/planning process $P_1$, which then provides a plan to controller $A_1$. The resulting path created from this configuration is therefore choppy due to the sampling required to efficiently handle the data load. Another configuration, $Config_B = S_2 : P_2 : A_2$, describes a sensor-feed back configuration that relies on much less information to drive the vehicle. The resulting path from this configuration is therefore smoother.

On a cautionary note, while $Config_B$ seems like a better choice for vehicle control, $Config_A$ will likely perform better than $Config_B$ when obstacles or rugged terrain are introduced. Why? $Config_A$ retains memory of prior information while $Config_B$ is only concern with immediate sensor values. $Config_B$ is therefore more susceptible to local minima than $Config_A$.

With a basic outline of process configuration and their relevance to robot behavior, the following definitions describe the formulation of process reconfiguration. The notation and formulation is adapted from the information space [8].

**Processes**. Let $p^i \in P$ define a process of any class for $N$ robot processes. Although not formalized in this paper, each process has a set of inputs and a single output that defines if two process can bind.

**Process Sensors**. Let $y^i \in Y$ define a process observation from a set of process sensors for each $p^i$. Note: these observations reflect process status as seen by a virtual sensor such as memory and processor consumption; process state (i.e. active, blocked or sleeping); or even algorithmic feedback such as percentage of algorithm completion.

**Process Configuration State**. Let $x^i \in X$ define the state of all processes interactions for $M$ configurations. One can imagine this being a snapshot of the robot's internal memory and, similar to the robot's physical environment, is much too large and complex for explicit representation. Each $x_i$ is defined by a unique connection pattern of $\{p^h : p^i : p^j : \ldots\} \subset P$

**Process Reconfiguration**. Let $u_i \in U$ define a reconfiguration over the set of possible configuration. Here $i$ is bounded such that $0 \geq i \leq M$. If $i = 0$, $U$ allows no reconfiguration, which implies a static system. If $i = M$, $U$ allows switching from one configuration to any other configuration. Note: constraints on reconfiguration may be defined to limit these configuration switches. For example, one possible constraint could be to only allow reconfiguration if the future configuration has a subset of the processes from the current configuration, or $x_i^k \cap x_i^{k+1} \neq \emptyset$ for stage $k$.

**Process Sensor and Reconfiguration Histories**. Let $\tilde{y}_k = (y_1, y_2, \ldots, y_k)$ denote a history of observations and $\tilde{u}_{k-1} = (u_1, u_2, \ldots, u_{k-1})$ denote a history of reconfigurations up to stage $k$.

**Process Configuration Cost Functional**. Let the cost of reconfiguration be

$$L(\tilde{x}_{K+1}, \tilde{u}_K) = \sum_{k=1}^{K} l(x_k, u_k) + l_F(x_{K+1}) \qquad (1)$$

where $l(x_k, u_k)$ is the cost to reconfigure and $l_F(x_{K+1})$ is the total cost of all prior configurations.

**Process Information State**. Let $\eta_k = (\eta_0, \tilde{y}_k, \tilde{u}_{k-1})$ be all the process information up to stage $k$.

**Initial Process Information**. Let $\eta_0$ define the start configuration of the robot at $k = 0$, which implies an initial start configuration $x_0$ and sensor observation $y_0$.

With the general problem formulated, two additional components are necessary for the problem definition to be complete: a definition of data stream sensors and the characterization of goal states. Although a variety of virtual sensors are possible, consider:

- A stimulation sensor. Every process has a set of inputs that must be present and valid as a precondition for output generation. For example, an obstacle avoidance process requires range data in order to generate an output signal. As such, $y_k^{stim_i} = 1$ if an output has been generated for $p^i$ at stage $k$; otherwise, $y_k^{stim_i} = \max(0, y_k^{stim_i} - \epsilon)$ where $\epsilon$ is a constant decay rate.
- A satisfaction sensor. Every process has a rating that signifies the quality of its output. For example, a localization process that is confident in its position estimate (i.e. low reported error) would output a high satisfaction rating whereas an estimate of low confidence will receive a poor degree of satisfaction. As such, let $y_k^{sat_i}$ fall within the range $[-1, 1]$ for $p^i$ at stage $k$.

The second component under consideration is goal expression. Unlike traditional planning/search problems that express goals explicitly in terms of the state space, process reconfiguration goals require additional information. The reason for this condition is that achieving a particular process configuration does not imply the robot will operate properly. As such, the information space formulation of this problem becomes important. Robot goals can be expressed in terms of information states and these states include sensor and action information. A desired goal is therefore expressed in terms of observations: let $\eta_{goal} \subset \tilde{y}_k$ such that $y^{sat_i} \approx 1 \in \tilde{y}_k$. This means that the robot's goal is to achieve a high satisfaction observation (within $\epsilon$ of 1) for process $p^i$ by the stage $k$.

The interesting characteristics of defining process sensors and goals in these terms is that (1) sensor observations are not necessarily predictable and (2) a variety of possible solutions exist for a goal that extend beyond the robot's "real" task specification. To elaborate on point (1), configuration is measured by stimulation sensors, which become stimulated if all the process inputs are present and valid. Sensor processes take input from the environment and thus will become stimulated by changes in the environment. As such, the observed process configuration can fluctuate even without reconfiguration. Point (2) means that prior to the goal stage $k$, the process configuration at stage $j$ where $j < k$ is not required to include the observed goal process $p^i$. As an example, a robot involved in a navigation task may be required to satisfy a process that compares its current position to a goal position. During this navigation task, the robot may need to call for help, open a gate, or recharge its batteries, which are not typically within a navigation problem domain; however, all of these situations do in fact reside along a solution path expressed in the context of process information states.

## V. Simulated Results

For these experiments, a simulator with a simple differential-drive robot was used to explore process observation and reconfiguration. This robot was provided five sensors: a range sensor, a bump sensor, a global position sensor, a sensor to translate user input, and a sensor that directs the robot to a particular landmark (i.e. a flag). The robot also has four mapping processes and five behaviors. These include ($x^{map}$): a prior map handler, a null mapper (has no map), an evidence grid mapper and a reflectance mapper; and ($x^{behave}$): two different grid planners, a control law drives the robot directly to the flag, a behavior that dodges obstacles and a behavior that bumps into obstacles and chooses a random direction to head afterward.

The robot is allowed to choose one mapper ($x_i^{map}$) and one behavior process ($x_j^{behave}$) to define a configuration. As such, $X$ is every possible map-behavior pairing. This paring yields twenty selectable configuration states. A stimulation decay rate of $\epsilon = 1$ is used to force stimulation to be either 1 or 0. Thus, for the observed configuration states, three of the five robot's sensors (e.g. the range sensor, bump sensor

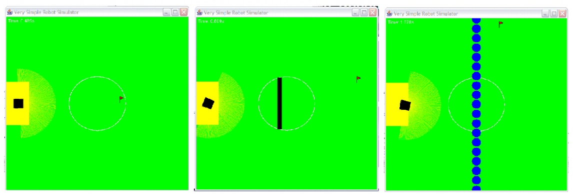

Fig. 3. The three game boards for this experiment. The robot is the square object, which is trying to capture a flag denoted as the small triangular object. The left board has no obstacles (World 1), the middle board has a wall in the middle (World 2), and the right board has a line of balls that can be moved if the robot pushes them (World 3).

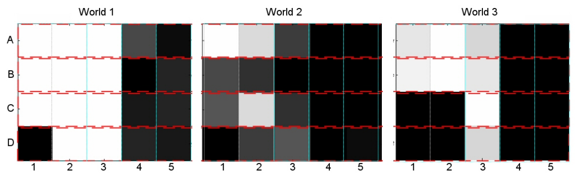

Fig. 4. These three images show a ratio of success-to-trials for each map-behavior pair. White cells indicate an averaged high percent of success over the trials whereas black cell indicate no success. Key: (A) Prior map, (B) No map, (C) Evidence grid, (D) Reflectance mapping, (1) Brushfire search, (2) A* search, (3) Servo control law, (4) Obstacle avoider, (5) Bump and random motion

and proximity sensor) can be stimulated by the environment yielding $2^3 = 8$ observed stimulation configurations. Each of the 20 process pairs can also be stimulated such that when multiplied by the number of sensor stimulation configurations yields 3280 unique stimulation observations. In reality, this number is far less since many processes will always be stimulated when active.

The robot is given the task to drive to a randomly placed flag using each possible configuration. This task is to be carried out in three different worlds: one with no obstacles, one with a fix obstacle and one movable obstacles (Figure 3). The task is repeated 100 times for each configuration on each world. The robot is timed and if the robot does not succeed in stimulating its flag proximity sensor within the time limit, the task is labeled a failure.

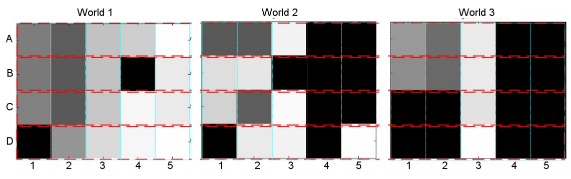

Fig. 5. These three images show a normalized average time for task completion given the configuration was successful. The key from Figure 4 applies also to this figure.

Figure 4 presents a series of gray-scaled images denoting the success-to-trial ratios for each static configuration pair for each world. As shown, configurations with heavy data streams and planning processes consistently perform the best when all the available information is correct as seen in cells A1 and A2. Processes that use smaller data streams (i.e. less information) consistently preform the worst (i.e. generate the most failures) as indicated by columns 4 and 5. Aside from these intuitive observations, there are three noteworthy occurrences in these plots. First, due to randomness in the environment, "mindless" routines that do not actively seek the objective, occasionally manage to be successful as seen in the last two columns of World 1. Second, what would seem to be a "good" map-behavior pairing (cell D1 that corresponds to a map generation routine and path planner), did not form a compatible data stream, which caused the robot to freeze. The third observation shows that when "good" algorithms obtain "good" data but the algorithmic assumptions are violated, the best configurations perform with a success ratio equal to the worst configurations as seen in cells C1, C2 and D2 of World 3. This situation is caused when active mapping registers the balls as obstacles, which forces the planner to declare the goal unobtainable. The control law in column 3 of World 3, which does not use a map or laser data, pushes through the obstacles because it is unaware of these data streams.

Figure 5 shows the performance times over all $X$ given the map-behavior pairing successfully obtained its objective. As presented, these results agree with intuition. On the random chance the flag was in the right place at the right time, the unreliable configurations of columns 4 and 5 achieved the goal fastest. The data-heavy streams performed consistently slower than the simpler behaviors as seen in columns 1 and 2. The middle ground, which employed the minimal amount of data to accomplish the goal, was the servo control law seen in column 3.

Equipped with these configuration priors, can data flow observation and reconfiguration improve the robot's chance at obtaining the flag? To answer this question, a cost functional and configuration search routine is required. For the next set of results, the following cost functions and selection routines were employed.

**Experiment 1**: No cost function or search routine is used. The static configuration data is analyzed to emulate the systematic trial of every possible configuration. **Experiment 2**: No cost function is used. After $k/3$ stages, where $k$ is predetermined last stage, select a new configuration from a uniform distribution. **Experiment 3**: Let $L(\tilde{x}_{K+1}, \tilde{u}_K) = E(K_x)$ be the expected success stage for each configuration. Order configurations by ascending expected stages and choose each configuration for its number of expected stages. **Experiment 4**: Compute a joint probability distribution from prior data. At each stage, choose the configuration that maximizes $P(Success|\tilde{y}_K)$, the probability of success given the observation history up the current stage $K$. To simplify this computation, the Markov assumption is made, making the calculated value $P(Success|y_K)$, which is computed using the Bayesian recursive algorithm.

The results from these experiments is reported in Table I. For Experiment 2 through Experiment 4, each reconfiguration approach was applied to each world for 100 trials to be consistent with the prior experiment. The metrics addressed in this table are the number of unique configuration observa-

TABLE I

PRELIMINARY RESULTS

| Key | Sys | Random | Expected | Bayesian |
|---|---|---|---|---|
| Observed | 109 | 191 | 162 | 110 |
| $AVE(Success)$ | 39% | 54% | 76% | 77% |
| $P(Success)$ | 0.46 | 0.59 | 0.79 | 0.8 |
| $AVE(t)\|S$ | 170s | 203s | 356s | 341s |

tions: Observed, the averaged success-to-trial ration across all worlds and configurations: $AVE(Success)$, the probability of success: $P(Success)$, and the average task completion time given the trial was successful: $AVE(t)|S$.

Starting with the first row, the data shows that as the configuration search becomes more sophisticated, a fewer number of unique observations are made. The underlying cause for this phenomenon is related to the switching routine. Random configuration selection will pick poorer performing pairs, thereby allowing a variety of previously un-encountered observations to be made. The second and third rows reflect the success rates of reconfiguration. Row 2 reflects how, on average, a reconfiguration would perform given one of the three worlds. This average, however, may be somewhat misleading since only a few particularly poor configurations (such as the incompatible pair) bring down the rest. The third row puts perspective on this value by showing that, on average, a configuration drawn from this distribution should perform better than the average. The difference between these values, however, becomes less significant the more optimized the selection routine becomes. Finally, the last row is a relative gage of system performance. Similar to what was observed Figure 5, the more specialized the selection routine, the longer it takes for reconfiguration to be successful.

Overall, these results present a clear case in favor of data flow regulation through process reconfiguration over static process configuration. Reconfiguration clearly leads to systems that can handle a larger class of problem in a variety of working conditions. On the other hand, one conclusion that seems to resonate with these results is that simple and specific statically configured data flows will work and, in general, outperform a reconfigurable system if most algorithmic assumptions hold true. Although, these results are still in a preliminary form - more conclusive formalism and experimentation are necessary to reinforce these claims.

## VI. CONCLUSIONS AND FUTURE WORK

The described system has shown promise in simulation, demonstrating that even simple restructuring strategies have great impact upon the success of robot to perform a specified task. The question that remains is to what degree is such a system technically feasible on a real robotic platform? While implementation and evaluation on real robots is underway, process reconfiguration has been achieved through a modular software design using standard interprocess communication utilities such as shared memory, message queues and sockets to enable data-flow routing. The actual cost and reconfiguration strategies will be no different that the simulated system since both reside at the computational level. Therefore, the greatest potential downfall of this system resides in the quality of the underlying processes.

The three key directions important to the future of this work are (1) a continuation of the formalism and theoretical principles of this work (2) investigation of alternative process sensors and (3) a demonstration of this approach on a robotic system in a real operational scenario. First, the graphical nature of data flow lends itself well to a number of analytical modeling methods. One of particular interest is a Bayesian network model since the inputs of one process form dependencies on other processes' outputs. Under this model, the system could be asked to compute the probability that a planning process is causing a failure verses a laser failure. In addition, the ability to automatically label failure configuration states verses successful configuration states opens the possibility of applying on-line learning methods to process reconfiguration.

## ACKNOWLEDGMENT

This work acknowledges Red Whittaker, Scott Thayer, Chuck Whittaker, and Zachary Omohundro. Special thanks also extends to Jame Kuffner and Steve LaValle for providing insight and assistance.

## REFERENCES

[1] A. C. Morris, D. Ferguson, Z. Omohundro, D. Bradley, D. Silver, C. Baker, S. Thayer, C. Whittaker, and W. R. L. Whittaker, "Recent developments in subterranean robotics," *Journal of Field Robotics*, vol. 23, no. 1, pp. 35–57, January 2006.

[2] S. Akella, W. Huang, K. Lynch, and M. Mason, "Sensorless parts orienting with a one joint manipulator," in *IEEE International Conference on Robotics and Automation (ICRA '97)*, vol. 3, April 1997, pp. 2383 – 2390.

[3] K. Astrom, "Optimal control of markov decision processes with incomplete state estimation," *Journal of Mathematical Analysis and Applications*, vol. 10, pp. 403–406, 1965.

[4] E. Sondik, "The optimal control of partially observable markov decision processes," Ph.D. dissertation, Stanford University, 1971.

[5] J. Pineau, "Tractable planning under uncertainty: Exploiting structure," Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2004.

[6] T. Smith and R. Simmons, "Heuristic search value iteration for pomdps," in *Proc. of UAI 2004*, Banff, Alberta, 2004.

[7] N. Roy, "Finding approximate pomdp solutions through belief compression," Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, September 2003.

[8] S. M. LaValle, *Planning Algorithms*. Cambridge University Press (also available at http://msl.cs.uiuc.edu/planning/), 2006, to be published in 2006.

[9] L. E. Parker and F. Tang, "Building multi-robot coalitions through automated task solution synthesis," *Proceedings of the IEEE, special issue on Multi-Robot Systems*, 2006.

[10] G. T. McKee and P. S. Schenker, "Networked robotics," in *Proceedings of SPIE - Sensor Fusion and Decentralized Control in Robotic Systems III*, vol. 4196. SPIE, October 2000, pp. 197–209.

[11] L. Wills, S. Kannan, S. Sander, M. Guler, B. Heck, V. Prasad, D. Schrage, and G. Vachtsevanos, "An open platform for reconfigurable control," 2001. [Online]. Available: citeseer.ist.psu.edu/wills01open.html

[12] C. Baker, A. Morris, D. Ferguson, S. Thayer, C. Whittaker, Z. Omohundro, C. Reverte, W. Whittaker, D. Hähnel, and S. Thrun, "A Campaign in Autonomous Mine Mapping," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, New Orleans, LA, 2004.