遠隔操縦ロボットのためのネットワーク分散型可変構造システムアーキテクチャに関する研究

Study on Network Distributed Variable Structure System Architecture for Tele-operator

電気電子情報通信工学　原　佑輔
Yusuke HARA

1. はじめに

近年，災害現場におけるレスキュー [1] や極限環境 [2] などにおける調査，観測作業や地雷除去 [3] に関わるロボットの実用化が始まり，遠隔操縦ロボットに対する期待の高まりと共にさらなる開発，運用の加速化が求められている．しかし，実際に運用することを考えると以下の点を考慮する必要がある．まず，遠隔操縦ロボットの運用では非可視運用や状況によっては通信容量の制限や通信時間遅れなどが発生するため，オペレータがロボット自身の情報やその周辺の環境情報をリアルタイムに理解することは難しい．そのため，周辺環境の認識や目的地の決定，周辺状況による行動決定など多くの自律機能が付加されている．しかし, このようなロボットの自律機能は開発者が開発時において想定した状況下において有効に機能するものの, 想定外の状況下に対してはリスクが発生する．通常, 自律機能が何らかの機能不全に陥った場合, 制御ゲインなど各種パラメータの変更での対応や, 根本的なアルゴリズムの変更で対処し, 障害を乗り切る必要がある. また，遠隔操縦ロボットの運用では運用環境の厳しさからハードウェアのトラブルが発生する確率が高い．しかし，災害救助や極限環境下での調査などにおいてはそのようなトラブルがあったとしても，もともと人が立ち入ることの出来ない遠隔地で運用されているため，普通のロボットのように回収して修理することは出来ない．このような場合においてミッションを継続するためには，ハードウェアの障害をソフトウェアで吸収する必要がある．このように遠隔操縦ロボットには運用中のシステム変更が要求される場面が多いため，その効率化が求められる．しかし，多くの場合, システムは一度配置すると運用中にシステムの機能や構成を変更することが難しく，さらにネットワークを介してのシステム変更ということもあり，システム変更が長期化する傾向にある．その一方で災害現場や極限環境下での環境調査などでは時間の有効利用が要求されるため，遠隔操縦ロボットにはシステムを停止することなく，一部の機能を変更できるような仕組みが求められている．

以上より，本研究では遠隔操縦ロボットにおける運用上の問題点を解決することを目的とし，ハードウェア，ソフトウェアの両方から柔軟なシステム変更を可能にする設計論の検討とアーキテクチャの提案を行う．

2. 基本概念及び設計論

システム変更を効率的に行うためにはモジュール化とネットワーク化によるシステム構築が有効である．モジュールとは機能の塊のことを指すが，モジュール化を行うと，システムのそれぞれの機能を明確に出来るため，システム変更の際の効率化に繋がる．更に，これらをネットワーク化することにより，モジュールやハードウェアの接続関係をソフトウェア的に切り替えることが出来る．モジュールを作る際は，システムの機能や従属関係を整理し，機能を分割することが重要であり，更にはモジュールのインターフェースを規格化することによって，モジュールの独立性を確保することが出来る．以上の概念に基づき，具体的な設計を考える．

2·1 設計のアプローチ

目的とするシステムはシステムの変更を柔軟に行えるシステムであるが，そのロジックを設計する前にハードウェアアーキテクチャやモジュールのシステムへの接続方法を具体的に考える必要がある（Fig.1）．以下で各層に関する説明を行う．

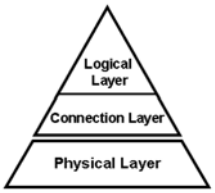


Fig.1 3 Layer Struction for System Architecture

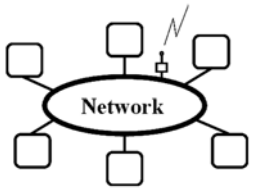


Fig.2 Star Type Network Topology

Physical Layer

ハードウェアのアーキテクチャを定義する層．ハードウェアアーキテクチャはソフトウェアの構築に多大な影響を与え，場合によってはソフトウェアの大きな制約となってしまう．ソフトウェアの制約にならないためにも，ハードウェアアーキテクチャは柔軟性が高くなるように組まなければならない．

Connection Layer

モジュールがシステムに接続されるときの方法を定義する層．システムへの接続方法は多数の方法が考えられるが，接続方法によっては，Logical Layer の構築方法に大きな影響を与えるため，出来るだけ自由度が高くなるようにしなければならない．

Logical Layer

最終的にモジュールが接続された後のシステムとしてどう振る舞うかを定義する層．Physical Layer，Connection Layer から受ける制約が少なければ少ないほど，この層で自由な設計が可能になる．

以下，Physical Layer と Connection Layer について，具体的な提案を行う．

2·2 Physical Layer の構成

遠隔操縦ロボットにはシステムを動かすための CPU はもちろんのこと，モータなどのハードウェアを動かすために複数の CPU が使用される．このように複数の CPU を持つシステムにおいてはその接続方法が上のレイヤーの仕様に大きな影響を与えるため，ここの議論が非常に重要になる．遠隔地で活動するシステムを運用するためには, 常にシステムの状況を把握する必要がある．特に，変更を要する不具合が発生した場合, その原因特定や危険回避及び機能変更のため，ハードウェア, ソフトウェアを問わず, 全てのシステムの状況の把握とアクセスが欠かせない．このため，システムには透過性の確保が重要な課題となる．しかし，通常，システムには接続関係が存在し，ハード的な接続方式によっては情報の直接的取得が難しい場合がある．これに対し，CPU の接続をスター型のネットワークベース (Fig.2) とし，接続関係をネットワーク上に維持することで, 必要に応じ各機能に直接アクセスしたり, 接続関係のソフト的に切り替えることが可能になる．その結果, 透過性を維持することができる．

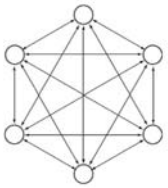


Fig.3 Custom Interface

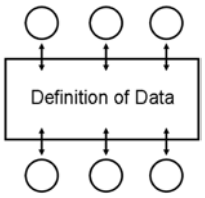


Fig.4 Universal Connection

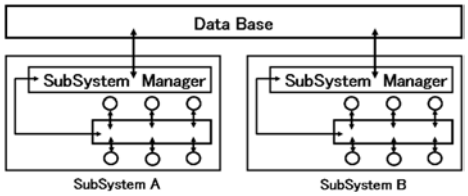


Fig.5 Class Composition of Modules

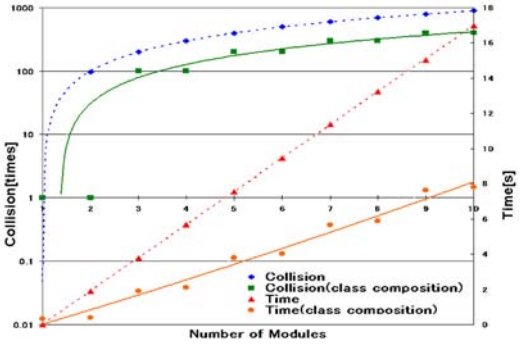


Fig.6 Relation among the number of modules,the number of times of a collision,and computation time

2·3 Connection Layer の構成

モジュールの組み合わせによって目的とするシステムを構築することは，システムに柔軟性を持たせる上で，非常に重要である．この時，モジュールにインターフェースの規格化がなされていれば，他のモジュールに影響を与えることなく，モジュールの入れ替えを行うことができ，部分的なシステム変更が可能になる．しかし，一般的な規格化ではモジュール同士の柔軟な接続に対応しきれない．一般的なインターフェースの規格化はデータの大きさ，データの種類，接続相手に関する定義の総合で決まるため，このうちどれか 1 つでも異なるだけで，全く違うインターフェースになってしまう．これがシステム変更を難しくしている原因のひとつである．例えば Fig.3 のようにモジュール同士を接続してシステムを構築している場合，同じ場所にモジュールを入れ替えることは出来ても，モジュールを追加することは出来ない．なぜなら，モジュールのインターフェースが既に 1 つの相手に固定され，規格化されているためでる，このような，モジュール同士を直接接続してシステムを構築している場合，新たなモジュールを追加するには既存のモジュールに新しいインターフェースを追加する必要がある．この作業は大変手間がかかるため，当初の目的を達成できない．そこで，本研究ではどのようなモジュールでもシステムに接続できるようにするために Universal Connection を提案する．

2·3.1 Universal Connection

Universal Connection は 1 つの規格で様々なデータを扱い，更には複数のモジュールと関係を持つためのインターフェースに関する概念である．従来，インターフェースの設定はデータの大きさ，データの種類，接続相手の定義であった．そのため，一度インターフェースの規格化を行うと，これらの情報のうち 1 つでも異なる場合はモジュールの接続が行えなかった．本研究で提案する，Universal Connection の概念ではモジュールの接続に柔軟性を持たせるために，接続の定義とデータの大きさやデータの種類の定義を分離させ，データの定義をモジュールの内部ではなく，モジュールの外部で行う．このことによって，同じインターフェースでも様々なデータや相手とやり取りをすることが可能になり，モジュール接続の自由度は飛躍的に高まるといえる．本研究では Universal Connection の実装例として，Fig.4 のようにモジュールをデータベースに接続する体系を提案する．このことにより，モジュールを追加する際，新規モジュールも既存モジュールもデータベースとのインターフェースだけを持てばよいため，既存モジュールへの影響がなくなり，機能の追加を容易に行うことができる．

3. システムの実現

ここではより実装に近い問題を解決していく．下層の問題は上層の設計に大きな制約を与えるため，Logical Layer の検討に入る前に Universal Connection を実現する上での問題点を解決する．

3·1 Universal Connection の問題点

前節で議論したアーキテクチャを構築する際，遠隔操縦ロボットのシステム規模が大きな問題となる．遠隔操縦ロボットは通信，行動計画，運動制御，認識技術，人とのコミュニケーションなど幅広い技術分野を持ち，それが複雑に絡み合って動作しているため，必然的に機能モジュールが多くなる．多数の機能モジュールがデータベースにアクセスする場合，以下の理由からシステム全体の性能が低下する．まず，モジュールの数だけ通信トラフィックが増加する．そのため，データベースにアクセスする際に通信の衝突が起き，待機状態のモジュールが発生するために遅延が発生する．更に,1 つのシーケンス動作を実行している場合は遅延の伝播が起こる．一方，モジュール間で協調動作をしている場合は，同期合わせが必要になるため，片方のモジュールの遅延がもう片方に伝播する．このような理由でシステム性能が低下すると，緊急停止など急を要する動作が間に合わなくなるため，ロボットに致命的な損害を引き起こす可能性がある．

3·2 アクセスの分散

システム性能が低下する一番大きな問題は同じデータベースに対し，多数のアクセスが集中することである．多数のアクセスが集中すると，データベースに書き込みできないモジュールが続出するため，遅延が大きくなる．この問題を解決するためにはモジュールのアクセス回数自体を少なくすることとアクセスするモジュールの数を減らすことが考えられるが，アクセス回数はモジュールのアルゴリズム等に依存するため，機能によっては削減できない．よって，全ての機能モジュールに対応できるように，後者の考え方に基づき，対応策を考える．

遠隔操縦ロボットはタスク別に使用するモジュールが決まるため，全体システムが機能別に分けられたいくつかのサブシステムから構成されるのが一般的である．よって，データベースのアクセス数を削減するために，サブシステムを階層化することで解決を図る．各サブシステムごとにデータベースを用意し，各サブシステムのモジュールは各サブシステムに存在するデータベースにのみ，アクセスを行う．それを各サブシステムに設置したサブシステムマネージャが監視し，サブシステムマネージャが全体のデータベースにのみアクセスを行う．概要図を Fig.5 に示す．このように構成することによって，全体のデータベースへのアクセスを減らすことができる．Fig.6 にモジュールを 1 〜10 個用意し，同期をかけずにデータベースを介して 1000 回演算させ，その時に生じる衝突回数と計算時間を示す．階層化によって両者とも改善が見られるため，この手法が有効であるといえる．

3·3 Logical Layer の構築

下位層の設計が決定したので，Logical Layer を構築する．ここではモジュールがシステムに接続された後，システムとしてどのように振る舞うかを定義する．本研究の目的はシステム変更の効率化であり，そのためには全体のシステムを停止することなく，一部のシステムを変更できること，また，モジュールを接続するだけでシステム変更が

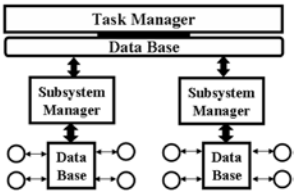


Fig.7 Model Architecture

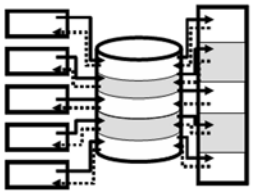


Fig.8 DataBase of Image

出来ること，この二つが重要である．前者を満たすためには逐次モジュール情報を把握できることが重要である．なぜなら，変更したいモジュールが使用されていない状態で，且つ今後もしばらく使われる予定がなければ，前者の要求を満たせるからである．また，後者は変更の手間を考えれば当然の要求である．本研究では以上の要求を TaskManager と SubSystemManager の二つの Manager を用いることによって実現する．

3·3.1 基本要素

Fig.7 に変更効率化を実現するためのシステムアーキテクチャを示す．このアーキテクチャの各データベース上にジョブテーブル，サブシステムテーブル，モジュールテーブルの 3 つのテーブルを設ける. ジョブテーブルはオペレータから送られてくる要求を遂行するために必要なサブシステムやモジュールの情報がジョブごとに格納されているテーブルである. モジュールテーブルは各サブシステムごとのモジュール情報がすべて格納されているテーブルであり，サブシステムテーブルは各サブシステムの情報（モジュール情報を含む）が全て格納されているテーブルである. システム起動時にこの 3 つのテーブルは生成され，モジュールがシステムに接続されると自動的に更新される (Fig.8). 本研究ではこれらのテーブルと TaskManager や SubSystemManager を用いて，当初の目的を達成する．

続いて，システムの振る舞いを一般化するためにモデル化を行う．モデル化に用いる要素と変数について以下を定義する．

オペレータ:O，要求:R，タスクマネージャ:T,
ジョブの数:N^J，ジョブ:$J_j(j=1,\cdots,N^J)$,
全ジョブ集合:$J=\{J_1,\cdots,J_j,\cdots,J_{N^J}\}$,
サブシステムマネージャの数:N^S,
サブシステムマネージャ:$S_k(k=1,\cdots,N^S)$,
S_k が管理するモジュール数:N_k^C，S_k が管理するモジュール:$C_{ki}(k=1,\cdots,N^S,i=1,\cdots,N_k^C)$,
ジョブに対するモジュール情報:M_j，システム情報:L，サブシステム情報:$l_{S_k}(k=1,\cdots,N^S)$
モジュール情報:$l_{C_{ki}}(k=1,\cdots,N^S,i=1,\cdots,N_k^C)$,
モジュールの仕事順:$\tau_{kx}(k=1,\cdots,N^S,x=1,\cdots,N_k^C)$,
最終結果:$F_{\tau_{kx}}(k=1,\cdots,N^S,x=1,\cdots,N_k^C)$

3·3.2 システム構成の遷移

モジュールが追加された際，システム構成がどのように変更されていくかを以下に示す.

1. モジュールはシステムに接続されると自らの情報をモジュールテーブルにアップする (モジュール情報とはプロセスモデル，状態，ID の 3 つである. プロセスモデルとはそのモジュールが仕事を行うのに要する時間であり，状態とはそのモジュールが稼動状態か待機状態かを指す). 新規に追加されるモジュールのみ新しいジョブ情報を持っており，接続される際に，その新規ジョブ情報をジョブテーブルに書き込む.
2. モジュールテーブルに情報がアップされると，各サブシステムマネージャはそこに自分の情報 (プロセスモデル，状態，ID) を加え，サブシステムテーブルの情報を更新する (サブシステムマネージャにとってのプロセスモデルとは各ジョブに所要する時間のことである).
3. タスクマネージャは各サブシステムテーブルを参照し，システム全体の構成を把握する (この 1〜3 のプロセスはモジュールが追加される度，自動的に行われる).
4. オペレータから要求を受けたタスクマネージャはジョブテーブルを参照し，各サブシステムにジョブを振り分ける (Fig.10 ①).
5. タスクマネージャからジョブを振られたサブシステムマネージャはジョブテーブルを参照し，ジョブに必要なモジュールをセレクトする (Fig.10 ②).
6. モジュールをセレクトしたら，モジュールテーブルを参照し，各モジュール情報をピックアップし，部分的なスケジュールを作成する (Fig.10 ③).
7. タスクマネージャは各サブシステムマネージャから部分スケジュールを受け取ると，サブシステムテーブルを参照し，全体のスケジュールを決定する (Fig.10 ④).
8. オペレータにスケジュールが承認されると，サブシステムマネージャに渡され，それぞれのスケジュールが実行される (Fig.10 ⑤).

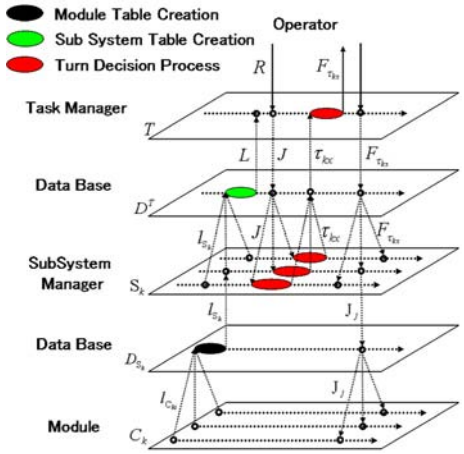


Fig.9 Data Flow Model

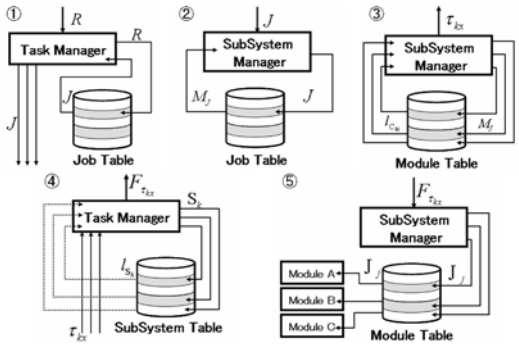


Fig.10 System Configuration Automatic Change Algorithm

実際にモジュールを追加して, システム構成を変えたい場合, オペレータはモジュールをシステムに接続（1〜3 が自動的に行われる）するだけでよい. あとはタスクマネージャとサブシステムマネージャにより，4〜8 が自動的に行われ，新しいシステム構成での運用が可能になる．全体のデータフローの外観を Fig.9 に示す．これにより，システムの状態を常に把握できるようになるため，オペレータはシステムを停止することなく，システムの一部を変更することが可能となる．

4. システム構築と稼動実験

以上の理論に基づき，本研究室で研究対象である遠隔操縦ロボット [4] のシステムを構築した (Fig.11)．更に，変更の際のユーザの負担を減らすために，RT ミドルウェア [5] を用いて，実装した (Fig.12)．続いて，実機による稼動実験を行った．実験内容はカメラによる地形データの取得及び DEM 生成，地形評価，パスの生成，走行というローバの基礎ミッションである．ローバが得た地形とオペレータが指定したゴールを Fig.13 に示す（右図の円で囲って

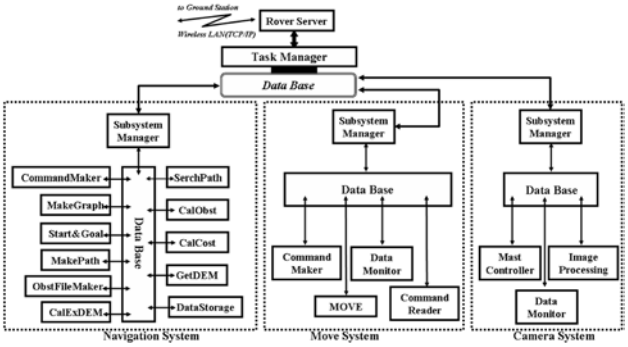


Fig.11 System Architecture for Tele mobile-robot

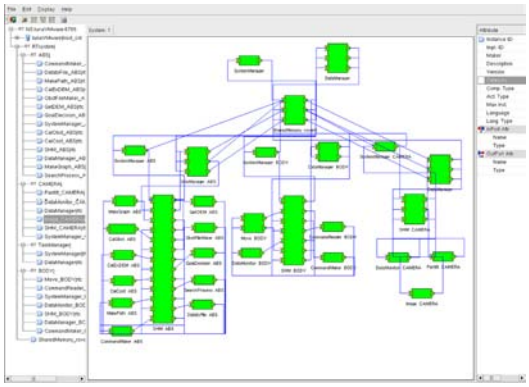


Fig.12 Rover System Over View Based on RTMiddleware

ある旗が指示したポイント）．その結果 Fig.14 が得られ，システムが正常に動作することが確認できた．

5. システム評価

当初の目標である，システム変更の効率性を検証する．Fig.3 のような多くの遠隔操縦ロボットで採用されているアーキテクチャと本研究で提案したアーキテクチャにおいて，実際にシステム変更をした際に伴う，作業量と作業時間の比較を行う．本論文では次のようなケースを想定した．

～ケース～
問題なく障害物の認識が出来ていたが，カメラがトラブルを起こし，画像にノイズがのるようになってしまった．そのため，物体の抽出が出来ない．

物体の抽出が出来ないと，ラベリングやマッチングなどの抽出後の処理が全て出来なくなるため，一刻も早く問題を解決しなければならない．そこで，既存システムにノイズを除去する機能モジュールを追加して，問題の解決を図る (Fig.15：普段の画像処理は撮像，物体抽出，ラベリング … と続くが，ノイズを除去するためにノイズの除去モジュールを追加し，処理順序を撮像，フィルタリング，物体抽出，ラベリングというように変更する.)

まず，従来のアーキテクチャの場合を考える．このケースでは撮像モジュールとエッジ抽出モジュールの間にノイズ除去モジュールを追加するため，既存モジュールにインターフェースの設計とコードの追加を行う必要がある．その結果，インターフェースの変更とコードの追加に約 5 分かかった．デバックなどの作業時間を含めるとシステムを再稼動させるまでに 6 分の時間を要した．

次に提案アーキテクチャの場合を考える．この場合，オペレータが行う作業は機能モジュールをシステムに追加することだけである．今回，システム変更までの時間を計ったところ 5 秒程度だった．作業時間は筆者が行ったものなので個人差があるが，この結果より，本研究で行った提案の有効性が確認できたと言える．今回は単純な例だったが，従来のアーキテクチャでは，変更が複雑になるとそれに伴う作業が非線形で増えていくので，変更が複雑になるとこの差はより開くと予想される．

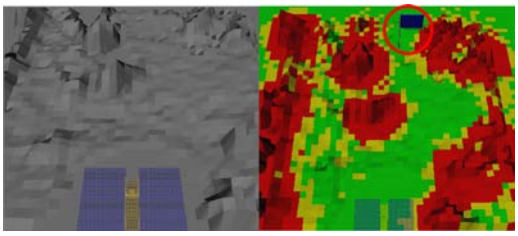


Fig.13 Geographical Feature Evaluation Result

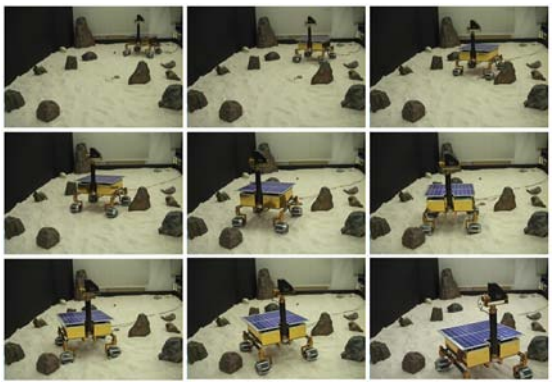
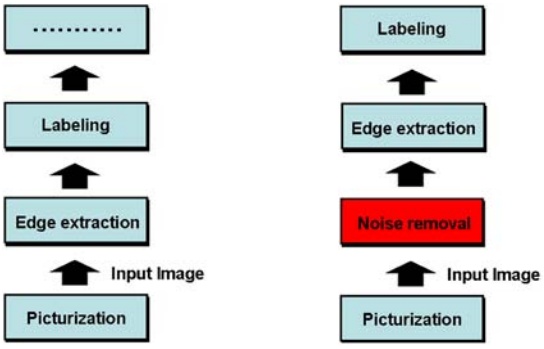


Fig.14 Experimental Result



(a) Before Flow　(b) After Flow

Fig.15 Addition of a Functional Module

6. まとめと今後の課題

本研究では遠隔操縦ロボットにおける運用上の問題点を解決することを目指し，ハードウェア，ソフトウェアの両方から柔軟なシステム変更を可能にするアーキテクチャの検討と設計論の提案を行い，ＲＴミドルウェアで実装した．最後に実験を行い，提案したものが有効であることを示した．今後は，このシステムの評価を更に複数の項目別に行い，有効性を証明すると共にそこで明らかになる問題点の抽出とその改善策について検討する．更にフェイルオーバーなど信頼性設計の導入を目指す．

参考文献

[1] 田所諭: “地震災害救助ロボット”, 日本損害保険協会, 予防時報,*214* 号,*pp.8-13,2003*
[2] Ari K.Jonsson, ConorMcGann, Liam Pedersen, Michael Iatauro, Srikanth Rajagopalan: “AUTONOMY SOFTWARE ARCHITECTURE FOR LORAX”,*i-SAIRAS,2005*
[3] 野波建蔵: “地雷探知ロボット”, 映像情報メディア学会誌, *Vol.57,No.1,pp87-90,2003*
[4] 國井康晴, 永塚真吾, 染谷智樹,”ローバシステム開発に向けた仮想ローバシミュレータの構築”, 第 *21* 回日本ロボット学術講演会
[5] http://www.is.aist.go.jp/rt/