卒業論文

論文タイトルは４６文字で２行に収まります１２３４５６７８９０１２３４５６７８９０１２３４５６

公立はこだて未来大学
システム情報科学部　情報アーキテクチャ学科
知能システムコース　1020259

姓 名

指導教員　姓 名

提出日　2024 年 1 月 25 日

BA Thesis

Title in English within two lines: Lorem Ipsum Dolor Sit Amet, Consectetur Adipiscing Elit, Sed Do

by

Firstname Lastname

Information Systems Course, Department of Media Architecture
School of Systems Information Science, Future University Hakodate

Supervisor:　Firstname Lastname

Submitted on January 25th, 2024

Abstract–

(Abstract should be about 150–200 words. Following is a sample text.) Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urna. Nunc viverra imperdiet enim. Fusce est. Vivamus a tellus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin pharetra nonummy pede. Mauris et orci. Aenean nec lorem. In porttitor. Donec laoreet nonummy augue. Suspendisse dui purus, scelerisque at, vulputate vitae, pretium mattis, nunc. Mauris eget neque at sem venenatis eleifend. Ut nonummy. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urna. Nunc viverra imperdiet enim. Fusce est. Vivamus a tellus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin pharetra nonummy pede. Mauris et orci. Aenean nec lorem. In porttitor. Donec laoreet nonummy augue. Suspendisse dui purus, scelerisque at, vulputate vitae, pretium mattis, nunc. Mauris eget neque at sem venenatis eleifend. Ut nonummy.

Keywords: Keyword1, Keyword2, Keyword3

概 要： ロボットソフトウェア開発において，ROS の利用が増加している．ROS を活用したロボットソフトウェアの構築ではクラウドを用いた分散ロボットシステムが注目されているが，ノード配置の柔軟性が課題となっている．特に，クラウドとロボットの CPU アーキテクチャの違いから，動的なノード再配置が難しい．先行研究での動的配置機構はオーバーヘッドの増加が問題とされ，mROS 2-POSIX を用いたアプローチも提案されているが，性能評価はネットワークスループットのマイクロベンチマークに留まり，実アプリケーションにおける有用性が十分に評価されていない．本研究では，mROS 2-POSIX と ROS 2 の性能を比較評価する．実験結果によって得た通信性能やメモリサイズをもとに，mROS 2-POSIX が複雑なアプリケーション上でも期待される性能を発揮できることを示す．

キーワード： クラウドロボティクス,Robot Operating System,WebAssembly

目次

第 1 章

はじめに

さまざまな産業向けやエンターテイメント関連のロボットシステム，自動車の自動運転技術や IoT システムのソフトウェア開発をサポートするフレームワークとして Robot Operating System（以下，ROS）の普及が増加している [1]．ROS のプログラミングモデルは，システムの各機能を独立したプログラムモジュール（ノード）として設計することにより，汎用性と再利用性を向上させて，各機能モジュール間のデータ交換を規定することで，効率的かつ柔軟なシステム構築を可能にしている．たとえば，カメラを操作して周囲の環境を撮影するノード，画像からオブジェクトを識別するノード，オブジェクトのデータをもとに動作制御を実行するノードを連携させることで，自動運転車の基本機能の一部を容易に実装できる．

ROS のプログラミングモデルは，ロボット/IoT とクラウドが協力する分散型システムにおいても有効である．ロボットシステムのソフトウェア処理は，外界の情報を取得する「センサー」，取得した情報を処理する「知能・制御系」，実際に動作するモーターなどの「動力系」の 3 要素に分類できる [2]．クラウドロボティクス [3] において，主に知能・制御系のノードを高い計算能力を持つクラウドに優先して配置することで，高度な知能・制御処理の実現を促進できる．さらに，ロボットが取得した情報や状態などをクラウドに集約・保存することで，複数のロボット間での情報共有と利用を容易にする．一方で，現行の ROS 実装では，各ノードの配置をシステム起動時に静的に設定する必要があり，クラウドとロボット間の最適なノード配置を事前に設計する必要がある．しかし，実際の環境で動作するロボットは，ネットワークの状況やバッテリー残量の変動など，システム運用前に予測することが困難な状況変化に対応する必要があり，設定したクラウドとロボット間のノード配置が最適でなくなる可能性がある．このような状況変化への対応として，ノードを動的に再配置するライブマイグレーション技術があるが，多くの場合でクラウドとロボット間の CPU アーキテクチャが異なり，命令セットがそれぞれ違うため，実行中のノードをシステム運用中にマイグレーションすることは技術的に困難である．

菅らは，WebAssembly（以下，Wasm）を用いることで，クラウドとロボット間での実行状態を含む稼働中ノードの動的なマイグレーションする手法を実現した [4]．Wasm とは

Web 上で高速にプログラムを実行するために設計された仮想命令セットアーキテクチャのことで，1 つのバイナリが複数のアーキテクチャで動作するため，異種デバイス間でのマイグレーションに適しているといえる．課題として，ROS 2 を Wasm 化したことでライブマイグレーション後のファイルサイズのオーバーヘッドが増大し，ノードの実行時間が大幅に増えてしまう問題が残った．柿本ら [5] は，組込みデバイス向けの軽量な ROS 2 ランタイム実装である mROS 2-POSIX を採用し，ROS ランタイムの Wasm 化にともなうオーバヘッド増加に対処した．しかし，採用された mROS 2-POSIX 上で指定されたメッセージを往復させるシンプルなアプリケーション上でしか評価実験はされていない [6]．そのため，ライブマイグレーション後のオーバーヘッド増加を解決するロボットソフトウェア基盤として，アプリケーションが複雑化した場合の動作が明らかでない．

本研究では，クラウドとロボット間での実行状態を含む稼働中ノードの動的マイグレーションの実現に向けた mROS 2-POSIX の性能評価を行い，mROS 2 が ROS 2 と比べて動的配置機構ロボットソフトウェア基盤としてどの点で優位性があるのか明らかにすることを目指す．

第 2 章

使用した技術

2.1 ROS 2

ROS 2 は，ROS の後継であり，ROS 2 は ROS 1 と比べて，分散型のロボットシステムに対応している．ROS 1 では，主に UDP を使用したメッセージ型の通信を行っていたが，ROS 2 では，DDS（Data Distribution Service）[?] と呼ばれる通信ミドルウェアを採用しており，RTPS（Real Time Publish Subscribe）プロトコルを用いたメッセージ型通信を行っている．これによって，個々のロボットが独立して動作するだけでなく，複数のロボットが協調して動作することが可能になった．現在は ROS 2 が主流であるが，ROS は，スタンフォード人工知能研究所の研究プロジェクトとから移管された Willow Garage 社によって開発が始まっているロボットソフトウェア開発基盤である．最初の正式なディストリビューション版は，2010 年 3 月にリリースされた Box Turtle である．ROS の利点は，分散型ロボットシステムの実現に向いている通信ミドルウェア，プロジェクト管理やデバックおよびシュミレーションなどのためのツール群，再現性の高い豊富な OSS のパッケージやライブラリ，世界規模の活発なオープンソース開発コミュニティという４つの側面にある [?]．すでに１０年以上の歴史を持っており，ロボット工学の発展に ROS は大きく貢献してきた．日本の企業ではソニーの aibo の事例が ROS を使った製品として代表的であり，企業でも商用商品に採用されることも多い．ロボット開発を開発を取り巻く環境や ROS が研究用から商用にも活用され始めるという変遷を受け，2014 年より第 2 世代バージョンである ROS 2 の開発が始まった．ROS 1 の思想を踏襲しながらも，その基本設計から見直し，一から実装しなおされている．2015 年 8 月には最初の distribution である Ardent Apalone がリリースされた．現在は ubuntu22.04 や windows，macOS などにも対応した ROS 2 humble ディストリビューションがリリースされている．

2.1.1 パブリッシャーとサブスクライバー

ROS では基本的なノード間のデータ通信としてパブリッシューサブスクライブ通信型の非同期な通信方式がある．データの送信側をパブリッシャー（Publisher，出版者）とよび，受信側をサブスクライバー（Subscriber，購読者）という．通信経路として，定義であるトピック（Topic）を介した通信が行われる．トピックで送受信されるデータはメッセージ（Message）と呼ばれ，車輪の角速度や回転量や現在位置の 3 次元座標など，基本型を組合せた任意の型を定義することができる．同じ名前のトピックに対して，様々な個数や種類のノードが任意のタイミングで登録や変更，削除ができる．さらに，メッセージの型が一致していれば，メッセージの通信が行われる．パブリッシュ側は自由なタイミングで動作でき，非同期に動作するサブスクライバー側は，パブリッシュ側では対応するコールバック関数が実行される．ノードはパブリッシャーにもサブスクライバーにもプログラム次第で変化できるため，ユーザがオリジナルのノードを作成することができるため，ノード同士の依存が少なくなり，ロボットシステム全体の機能の追加や削除が容易であるため，柔軟なロボットシステムを構築できる．また，ROS 2 ではパブリッシャーは rclcpp::Publisher クラスを，サブスクライバーは rclcpp::Subscription クラスを使用して実装する．

ROS 2 の通信ミドルウェアである DDS と通信プロトコルである RTPS は，通信相手の探索および通信経路の確立を自律的に行う．この機能の実現は，RTPS の SPDP（Simple Participant Discover Protocol）というプロトコルによって行われる．および SEDP（Simple Endpoint Discover Protocol）が備えられている．ここで, RTPS では, ROS 2 のノードに相当するものを Participant（参加者）と呼ぶ．通信相手を探索するためには自身の情報を送信するモジュールを Writer，ほかの Participant から情報を受け取るモジュールを Reader と呼ぶ．SPDP は, Participant の情報を送受信するためのプロトコルであり, SEDP は, Topic の情報を送受信するためのプロトコルである．この通信のエンドポイントは，Participant 同士のパブリッシュ，サブスクライブである．

RTPS を OSI 参照モデルに例えると transport 層に位置し，UDP/IP の上に実装されている．UDP 通信にはパケットの到着に関する保証がないが，ROS 2 の DDS ではこれを補助する QoS（Quality of Service）制御の機能がある．QoS 制御は，サブスクライバ，パブリッシャごとに設定でき，厳格な条件であれば RELIABLE（信頼性の高い通信）を，緩やかな条件であれば BEST EFFORT（ベストエフォート通信）を選択することができる．ROS 2 のデフォルト DDS である FastRTPS では，QoS 制御の機能が実装されており，各ノードが通信するときの QoS 設定は RELIABLE である．

2.1.2 ノードの作り方とパッケージ

ROS 2 においてノードを作成する場所は決まっており，一般的には ros2_ワーキングスペース（ws）というディレクトリを作った後，その中に src というディレクトリを作成し，その中にノードを作成する．src ディレクトリを作成した後，ros2_ws ディレクトリで colcon build というコマンドを実行することで，ros2_ws の中でノードを作成することがきる．ROS 2 の場合，ノードの集まりのことをパッケージと呼ぶ．パッケージは ros2 pkg create というコマンドを実行することで作成することができ，src ディレクトリの直下で実行することで，パッケージを作成することができる．パッケージを作成する前にそのパッケージの中で使用する言語を決める必要がある．ROS 2 は C++ と Python の 2 つの言語をサポートしており，ユーザーの好みに合わせて変更できる．

2.1.3 オーバーレイとアンダーレイ

また，ROS 2 には重要な概念にアンダーレイとオーバーレイがある．アンダーレイは，完成されたパッケージをインストールするワークスペースであり，安定した環境をオーバーレイに提供するためにある．オーバーレイは，ユーザー自身で作成したパッケージを扱うワークスペースであり，先ほどの ros2_ws はオーバーレイにあたる．ROS 2 ではユーザーが作るノードやパッケージをオーバーレイに作成し，必要に応じてアンダーレイのパッケージを参照して使用するのが一般的である．このオーバーレイ上でユーザは colcon build を実行し，パッケージをビルドする．ユーザーはパッケージをビルドした後，すぐに実行することができない．ROS 2 では source コマンドを用いてオーバーレイ環境を読み込むことでアンダーレイがオーバーレイより優先されることなく，開発することができる．ROS 2 においてオーバーレイとアンダーレイは，複雑化するロボットシステムに柔軟性と拡張性をもたらす概念であることがわかる．ROS 2 のデバック方法として様々なコマンドが用意されており，ros2 topic list というコマンドを実行することで，現在実行されているトピックの一覧を表示することができる．また，現在実行されているノードを視覚的に確認できるように rqt_graph と呼ばれるものが用意されている．開発者は rqt_graph を用いて，目に見えない ROS 2 のノード間の通信を確認することができる．こうしたアンダーレイ機能の充実によって，ROS 2 は ROS 1 よりも柔軟性と拡張性を持つことができた．

2.1.4 ROS 2 が対応している DDS

ROS 2 にはデフォルトで FastRTPS という DDS が実装されている．DDS とは，OMG（Object Management Group）が定めたデータ交換のための仕様である．これによって分散型ネットワークでも効率的に通信が可能になっている．FastRTPS のほかにも，RTI

Connext DDS や eProsima Micro XRCE-DDS という DDS が ROS 2 に対応している．Ubuntu22.04 の ROS 2 humble では，FastRTPS はもちろんのこと，デフォルトで Ecripse Cyclone DDS，Gurum DDS，RTI Connext DDS がインストールできる．

2.1.5 サービスとアクション

ROS 2 では，パブリッシャーとサブスクライバー以外にもサービス通信とアクション通信がある．サービス通信は，サーバーとクライアントの 2 つのノード間でリクエストとレスポンスをやり取りする通信である．これは既存のクライアントーサーバーモデルによく似ており，Subscribe するのではなく，ノードがメッセージの値を欲しいタイミングでリクエストする仕組みになっている．そのため，ロボットに対する命令やデータの取得，計算結果を受け渡しに適しており，サービス通信はリアルタイム性や連続的なデータには向かないが，確実に応答するというロボットシステムにおいて重要な役割を果たす．さらに，サービス通信は同期的な通信であるため，サービス通信を行うノードは，サービス通信が完了するまで他の処理を行うことができない．そのため，クライアントの処理を間接的にブロックすることができる．一方，アクション通信は，サービス通信と同様にサーバーとクライアントの 2 つのノード間でリクエストとレスポンスをやり取りする通信である．しかし，アクション通信はサービス通信と異なり，リクエストに対するレスポンスを即座に返すのではなく，リクエストに対するレスポンスを返すまでの間に，進捗状況を返すことができる．この通信方式によって，長時間のタスクやフィードバックが可能なタスク，中断可能なタスクに適しているため，開発者は柔軟性を保ちながら，ロボットシステムを構築することができる．さらに，アクション通信は非同期的な通信であるため，アクション通信を行うノードは，アクション通信が完了するまで他の処理を行うことができる．これによって，複雑なタスクを行うノードでもアクション通信を用いることで同時に処理することが可能であり，多くのリソースがあるマシンで高度なノードを動作させることができる．実装方式として，サービス通信は，ROS 2 では rclcpp::Service，Client クラスを用いて実装する．アクション通信は，ROS 2 では rclcpp::ActionServer，Client クラスを用いて実装する．

2.2 mROS 2-POSIX

2.2.1 mROS 2

mROS 2 は，ROS 2 ノードの軽量実行環境である．ROS 2 を採用するロボットシステムにおいて通信方式とメモリ軽量な実行環境を確立することができる組み込み技術を導入することにより，分散型のロボットシステムにおける応答性やリアルタイム性の向上，消費電力の削減が可能になる．mROS 2 は汎用 OS 上で実行される ROS 2 ノードと通信できることを目的として実装された．そのため，パブリッシューサブスクライブ通信の相手と経路を自

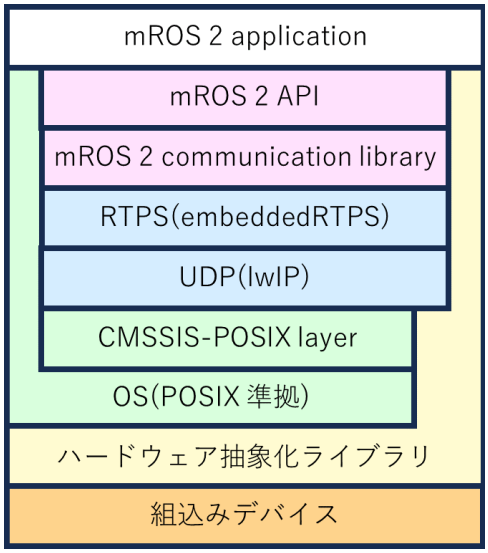

図 2.1: mROS 2-POSIX の内部構成

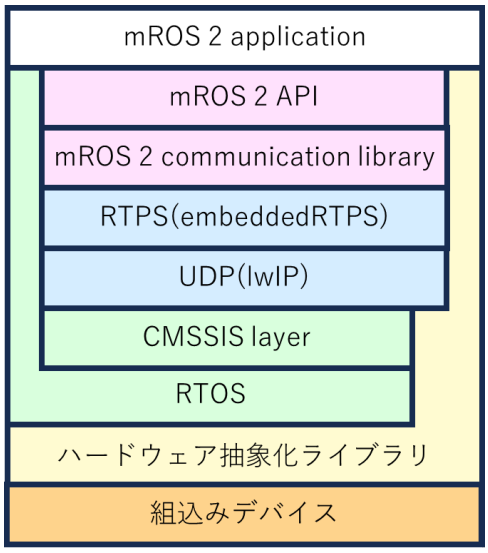

図 2.2: mROS 2 の内部構成

律的に探索できるように設計され，ROS 2 および RTPS の利点が，組み込み技術導入時に損なわれないようになっている．ROS 2 に対応している DDS の種類として，FastRTPS, RTI Connext DDS，eProsima Micro XRCE-DDS を挙げたが，既存の組込みデバイス向けの ROS 2 ノード実行環境である micro-ROS[?] がある．micro-ROS は RTPS の軽量規格である DDS-XRCE（DDS For Extremely Resource Constrained Enviroments）[?] の実装である Micro XRCE-DDS[8] を採用している．Micro XRCE-DDS は，ホストとして ROS 2 ノードを実行するデバイスと通信する際に，Agent ノードと呼ばれるノードの稼働が必要となる．通信の仲介の役割を Agent ノードはになっており，Agent ノードは，ホストデバイス上の ROS 2 ノードと RTPS に則った通信，組み込みデバイス上のノードと XRCE に則った通信がそれぞれ行われるため，応答性とリアルタイム性の低下が懸念される．また，複数の組み込みデバイスを用いる場合，Agent ノードは分散型システム全体の通信を仲介するため，Agent ノードの数が増えると通信の遅延が増加する．そのため mROS 2 は，DDS として XRCE-DDS を採用せず，embeddeRTPS[?] を採用している．embeddeRTPS は，一つの Domain クラスから Participant，Writer，Reader の 3 つのインスタンスが生成される．つまり，初期化処理の段階で embeddedRTPS の提供するパブリッシューサブスクライブ通信が利用できるようになる．これによって XRCE-DDS のように Agent ノードが必要なく，組み込みデバイス上での通信の遅延を抑えることができる．この embeddeRTPS が採用されたのは通信の遅延を抑えることができるだけではない．この RTPS は，SPDP と SEDP が実装されており，通信の宛先や受け手として自立性を確保できる RTPS であること点である．また，ROS 2 で代表的な FastRTPS と通信の確認ができているため親和性が高い点も上げられる．以上の設計思想により mROS 2 は，計算資源の限定的な組み込みデバイス上で

の稼働を想定した組込みデバイスのリアルタイム性の向上および消費電力の削減ができるソフトウェア基盤である．

2.2.2 mros2 の内部構成

図 1（a）に mros2 の内部構成を示す．ユーザーアプリケーションからの階層順で，mROS 2 通信ライブラリ，通信プロトコルスタック，RTOS，ハードウェア抽象化ライブラリによって構成される．mROS 2 通信ライブラリは，ユーザアプリケーションに対して，ROS 2 通信のトピックに関する基本的な API を提供している．主な API として void mros2::init()，void mros2::Node::create_node()，void mros2::Publisher::create_publisher()，void mros2::Subscriber::create_subscriber() がある．通信プロトコルスタックには，C++ で実装された embeddedRTPS を採用している．先ほど述べたようにこの RTPS には SPDP と SEDP が実装されており，計算資源の限定的な組込みデバイス上での稼働を想定した設計であるかつ，ROS 2 の代表的な RTPS である FastRTPS と通信の確認できているという理由がある．UDP については組込み向けの C による軽量実装である lwIP[?] が採用されている．RTOS には，TOPPERS/ASP3 カーネル [?] が採用されており，高分解能大麻やティックレスの低消費電力な処理遅延機能など，高いリアルタイム性と安全性が求められる軽量な組込みシステムに適した設計がなされている．lwip は CMSIS-RTOS API に依存しているため，それぞれの API 差分を吸収するラッパが用意されている．

2.2.3 mROS 2 の通信機能

mROS 2 の通信機能は，mROS 2 の通信ライブラリにある init task（初期化処理）と RTPS/UDP と通信ライブラリを介する write task（パブリッシュ処理）と reader task（サブスクライブ処理）の 3 つのタスクに分けられる．またアプリケーション層にある user task（ユーザーアプリケーション）という開発者が実装するタスクがある．user task は，ROS 2 のノードに相当している．パブリッシューサブスクライブ通信を行うアプリケーションである．mROS 2 の API を介してパブリッシュやサブスクライブを行うことができる．init task は ROS 2 としてのノードの情報の初期化を行う．APImros 2::init() が呼ばれたときに，対象の組込みデバイスを RTPS の Participant として登録する．writer task と reader task に関しては，パブリッシュおよびサブスクライブに関する処理を担う．これらのタスクは user task からの依頼を mROS 2API を介して受け，RTPS の該当機能を立ち上げる．writer task はパブリッシュの依頼を受けて起動し，reader task はサブスクライブの依頼を受けて起動する．

2.2.4 mROS 2-POSIX の内部構成

そして，mROS 2 が POSIX[7] に対応したのが mROS 2-POSIX である．

図 1（b）は，mROS 2-POSIX のソフトウェア構成を示す．mROS 2-POSIX アプリケーション層は，ユーザが実装する ROS 2 ノードに相当する．つまり，ROS 2 におけるオーバーレイに相当する層である．mROS 2-POSIX API 層および通信ライブラリ層は，メッセージを非同期にパブリッシュやサブスクライブするためのコミュニケーションチャネルである ROS 2 の Topic に相当する API および通信機能を提供する階層である．本階層は，ROS 2 のネイティブなクライアント通信ライブラリである rclcpp と互換性を保つように設計されている．mROS 2 通信ライブラリでは，rclcpp のうち pub/sub 通信の基本的な機能のみ実装されている．利用可能な機能は制限されているものの，組込み技術を導入する ROS 2 開発者は，汎用 OS 向けのプログラミングスタイルを踏襲しながら C++ によって mROS 2 のアプリケーションを実装できる．そのため mROS 2-POSIX はサービス通信やアクション通信には対応していない．

RTPS プロトコルスタックには UDP でパブリッシャとサブスクライバ C++ 実装の embeddedRTPS[8] が採用されている．UDP については組込み向けの C 実装である lwIP が採用されている．通信層の embeddedRTPS および lwIP は CMSAIS-POSIX に依存しており，図 1（b）に示す mROS 2 の CMSIS-RTOS を互換した層になっている．最下層にはハードウェアを抽象化したライブラリがある．

mROS 2-POSIX は図 2 に示す実行方式を採用している．リアルタイム OS では，組込みマイコンを実行資源の管理対象として，タスク単位でアプリケーションが実行される．POSIX においてはタスクに相当する概念はプロセスであり，そこから生成されるスレッドを実行単位として処理が進行している．しかし，mROS 2-POSIX は実行単位であるノードに POSIX のスレッドを対応づけ，組込みマイコンでの通信処理におけるイベント割込みについては，POSIX 準拠 OS におけるブロッキング API の発行に相当させて処理している．これらの方式によって，mROS 2-POSIX は POSIX 準拠 OS 上で仮想 ROS 2 ノードとして軽量環境下で実行することができる．

2.2.5 mROS 2-POSIX の動作フロー

mROS 2-POSIX の動作は以下のようになっている．

- netif_posix_add(NETIF_IPADDR, NETIF_NETMASK) によって，lwIP のネットワークインターフェースを初期化する．
- osKernelStart() によって，RTOS のカーネルを起動する．
- mros2::init(0, NULL) によって，mROS 2-POSIX の初期化を行う．

- mros2::Node::create_node(ノード名) によって，ノードを生成する．

mROS 2-POSIX は通常の mROS 2 と同様に lwip を利用して UDP stack を実装している．異なる点として，mROS 2 ‐ POSIX は POSIX レイヤが設けられていることが挙げられる．このように明確なレイヤを設けたことによって，netif_posix_add(NETIF_IPADDR, NETIF_NETMASK) のような実装が追加されている．引数に現在 mros2-posix をビルドしているマシンの IP アドレスとサブネットマスクを設定することによって，lwIP のネットワークインターフェースを初期化する．この NETIF_IPADDR と NETIF_NETMASK は，mROS 2-POSIX のヘッダファイルである mros2_posix_netif.h に定義されている．そのためビルド前に ip a などで自分の IP アドレスを確認し，その IP アドレスとサブネットマスクを設定する必要がある．この設定をを行わないと，ROS 2 やほかの mROS 2 ノードと通信できなくなり，ros2 topic list などのデバックコマンドにも表示されない．

次に，osKernelStart() によって，RTOS のカーネルを起動する．その後，mros2::init(0, NULL) によって，mROS 2-POSIX のノードの初期化を行う．mros2::init() は mROS 2-POSIX のノードの初期化，つまり，ROS 2 としてノード情報を初期化するということである．これは対象の組込みデバイスを RTPS の Participant として登録する．このタスクを行った後，mros2::init() は休止状態に移行する．

そして，mros2::Node::create_node(ノード名) によって，ノードを生成する．この命令に紐づけられたインスタンス変数を介して，パブリッシャーやサブスクライバーを生成する．

2.2.6 mROS 2-POSIX のパブリッシュ処理

mROS 2-POSIX においてパブリッシュ処理は mROS 2 とほとんど変更がない．まず，mros2::Node::create_publisher() によって mROS 2 ノードをパブリッシャーとして登録する．mros2::Node::create_publisher は関数テンプレートとして実装されており，第一引数にトピック名，第二引数にパブリッシュするメッセージのキューのサイズを設定する．例として，mros2::Node::create_publisher¡型¿(トピック名,10) という形で記述する．¡型¿の中に入るのは，パブリッシュするメッセージの型である．ROS 2 同様に様々な種類の型を設定でき，ユーザーが自由に定義することができる．同様に”トピック名”にはトピック名が，10 はパブリッシュするメッセージのキューサイズ（履歴の長さ）が入る．定義された publisher がどのようにパブリッシュされるのか以下にフローを示す．

- mros2::Publisher.publish() を呼び出す．引数にはメッセージ情報が格納されたオブジェクトのポインタを渡す．
- mROS 2-POSIX の通信ライブラリ内でメッセージのシリアライズを行い，RTPS に則った形式に変換する．
- embeddedRTPS の提供する機能によって UDP パケットを作成し，タスク間通信で

mROS 2-POSIX の Participant の writer にパブリッシュ処理を依頼する

- writer のタスクが実行され，RTPS の SPDP によって，パブリッシャー情報を送信する．

mROS 2 - POSIX のパブリッシュ処理は，mROS 2 と同様に UDP パケットを作成し，RTPS の SPDP によって，パブリッシャー情報を送信する．mROS 2 通信ライブラリの通信処理効率化のために writer task は，CPU とネットワーク通信を平行に行うことができる．このため，embeddedRTPS や lwIP におけるメッセージ，UDP パケットの送信処理に関して，user task から分離して writer task で実行されるようになっている．

2.2.7 mROS 2-POSIX のサブスクライブ処理

mROS 2-POSIX においてサブスクライブ処理もパブリッシュ処理と同様にほとんど変わらない．mros2::Node::create_subscription() によって mROS 2 ノードをサブスクライバーとして登録し，メッセージをサブスクライブできるようにする．サブスクリプションも同様に関数テンプレートとして実装されており，テンプレートの仮引数にはメッセージ型，引数にはトピック名と RTPS における QoS のキャッシュに加えて，サブスクライブ時に呼び出されるコールバック関数を指定する．例として，mros2::Node::create_subscription¡型¿(トピック名,10, コールバック関数) という形で記述する．これによって embeddedRTPS に紐づけられたため，サブスクライブ機能を利用できる．定義された subscriber がどのようにサブスクライブされるのか以下にフローを示す．

- ノードの Participant の reader が UDP パケットを受信する．
- 初期化タスク時に Participant 情報として登録されていた embeddedRTPS 内のコールバック関数が実行され，RTPS に則った形式に変換する
- mROS 2 通信ライブラリ内で RTPS パケットのデシリアライズを行い，メッセージに変換する．
- タスク間の通信によって登録されていたコールバック関数およびサブスクライブされたメッセージのオブジェクトポインタを渡す．
- メッセージのオブジェクトのポインタを引数としてコールバック関数を実行する．

メッセージのパブリッシューサブスクライブ通信は非同期的に行われるため，reader task がサブスクライブを待ち受けるには CPU 使用権を専有する必要があり，メッセージサブスクライブに対するコールバック関数を実行する必要がある．その為，この処理時間が長くなる場合に，次のメッセージのサブスクライブを待てないという問題が発生する．UDP パケットの到着を高い優先度で定期的に監視する役目を reader task が行うことで到着時にメッセージのサブスクライブ処理を行うようにしている．

2.2.8 mROS 2-POSIX のメッセージ生成

ROS 2 では様々なメッセージ型を定義して通信することができるが，mROS 2-POSIX もオリジナルのメッセージファイルを用意してそのメッセージ型で通信することができる．そもそも mROS 2 にはデフォルトで sensor_msg/msg/image.hpp,std_msgs/msg/bool.hpp, byte.hpp，char.hpp，float32.hpp，float64.hpp，header.hpp，int16.hpp，int32.hpp，int64.hpp，int8.hpp，string.hpp，uint16.hpp，uint32.hpp，uint64.hpp，uint8.hpp というメッセージ型が用意されている．これらのメッセージ型は，mROS 2-POSIX でも利用することができるが，ほかのメッセージ型も mros2/mros2_header_genera

第 3 章

検討手法

3.1 Sphero sprk+

Wii Fit Board と Raspimouse を用いたロボットの動作を制御するアプリケーションを実装する．Raspimouse とは，ロボット開発キットである Raspberry Pi Mouse のことで，Raspberry Pi 上で動作する ROS 2 のノードで制御することができる．Wii Fit Board は，任天堂が発売した Wii の周辺機器で，体重移動を検知することができる．このアプリケーションは，Wii Fit Board からのセンサデータを Raspimouse に Pub-Sub 通信を用いて送信し，Raspimouse は受信したセンサデータをもとに動作を制御する．ユーザの体重移動によってロボットが動作するため，エンドツーエンドのアプリケーションである．また，Raspimouse 上でノードを立ち上げ動作しているため，組み込みデバイス上で動作するアプリケーションという条件も満たしている．評価実験に用いるアプリケーションとして，Wii Fit Board と Raspimouse を用いたロボットの動作を制御するアプリケーションを実装するが，本来は，Wii Fit Board からのセンサデータを Sphero sprk + に Pub-Sub 通信を用いて送信し，Sphero sprk + は受信したセンサデータをもとに動作を制御するアプリケーションを実装する予定だった．Sphero sprk + は，Sphero 社が発売した球体のロボットで，ユーザーがスマートフォンから操作することができる．ROS 2 側では Wii Fit Board からのセンサデータを受け取り，Sphero sprk + に送信する 2 つのノードを実装することに成功したが，mROS 2-POSIX 側では Wii Fit Board からセンサデータを送信するノードを完成させることができたものの，Sphero sprk + を受信したセンサデータをもとに動作させるノードの作成が難しく，実装を断念した．実装を断念した理由として，ROS 2 側の実装では Sphero sprk + は Python のライブラリを用いて制御しており，mROS 2-POSIX は Python の Support はないため，動作させるには，Python のライブラリを C++ 言語に変換する必要があった．しかし，Python のライブラリを C++ 言語に変換するためには，bluez を用いて Sphero sprk + を BLE サーバーと見立てたクライアントのスクリプトを実装する必要があり，その実装が間に合わないと判断したため，実装を断念した．また，Sphero sprk + を

動作させるライブラリは go 言語にもあったため，cgo と呼ばれる go 言語から C 言語の関数を呼び出すための仕組みを用いて，go 言語から Sphero sprk + を動作させることも試みたが，mROS 2-POSIX の cgo 移植になってしまい，実装範囲が広くなってしまったため，実装を断念した．

3.2 Raspberry Pi Mouse

実装アプリケーションは ROS 2 と mROS 2-POSIX の 2 つの環境で動作する．ROS 2 側のアプリケーションは，C++ 言語で Wii Fit Board のセンサの値をパブリッシュするノードを実装した．センサの値をパブリッシュするためには，Wii Fit Board を端末と Bluetooth 接続する必要がある．接続後は/dev にある/event ○○を開くことで Wii Fit Board のセンサ値を取得できる．しかし，/event はネットワーク環境によってランダムな/event 番号に振り分けられるため，実行するネットワーク環境に変更があるたびに修正しなくてはならない．また，ノード実行前に/dev/uinput に chmod で a+rw の権限を与えておく必要がある．Wii Fit Board はそれぞれの角に 4 つのセンサがあり，センサの値を/event から 4 つの値が取得できる．この値をこのままパブリッシュすると，サブスクライブする側で 4 つの値を処理しなくてはならないので，この 4 つの値を x,y の 2 つの値に変換する．x,y はユーザの重心点を表しており，Board に乗っているユーザの体重に応じてその大きさが変化する．次に，x,y の値をサブスクライブするノードではパブリッシュされた x,y の値を受け取り，x,y の値に応じて Raspimouse の動作を制御する処理を実装した．x,y の値はそれぞれユーザの体重によって大きく増減するため，適切な閾値の値はユーザーによって変化する．今回の場合は，評価実験するユーザーとして開発者のみになるため，開発者の体重に合わせた閾値の値を設定した．ロボットが動作する処理は，x,y の値が閾値を超えた場合に Raspimouse が動作するように実装した．ユーザーがロボットを前進させようとした時，y の値がマイナスに大きく傾くため，y の値がマイナスの閾値を超えた場合に Raspimouse が前進するように実装した．同様に，y の値がプラスに傾くとき，ロボットを後退させ，x の値がマイナスに傾くときにロボットを右に移動させ，プラスに傾くときにロボットを左に移動させるように実装を行った．ロボットを移動させる処理はロボットのデバイスドライバを直接 write 命令を用いてたたくことで実装を行っている．mROS 2-POSIX 側のアプリケーションは，ROS 2 側のアプリケーションと大きく変更されていない．ROS 2 側のアプリケーションと同様に Wii Fit Board のセンサの値をパブリッシュするノードを実装し，その値をサブスクライブするノードを実装した．パブリッシュするノードでは ROS 2 側と同様の実装になっている．異なる点は ROS 2 側では rclcpp と呼ばれる ROS 2 のノードを作成するためのインターフェースを使用しているが，mROS 2 ではオリジナルのインターフェースが用意されている．

3.2.1 実装に際しての課題

実装に際しての課題として Wii Fit Board の値をパブリッシュするノードとその値をサブスクライブし，Raspimouse を動作させるノードそれぞれであった．Wii Fit Board の値をパブリッシュするノードでは，mROS 2-POSIX 側のノードで実行後に Segment Fault が発生した．mROS 2-POSIX を利用していると，ビルドが通って実行時に Pub-Sub 通信の準備が完了した後，Segment Fault が発生することがある．これによって，ノードが突然動作しなくなる問題があったが，再度ビルドすることで修正することができた．Wii Fit Board のセンサの値をパブリッシュするノードからサブスクライブするノードでは，Raspimouse を動作させる部分で課題があった．Raspimouse を制御する方法として ROS 2 を利用する方法が一般的である．方法としては，Raspimouse の制御ノードを立ち上げ，制御ノードがパブリッシュしているトピックを別のノードでサブスクライブしたり，制御ノードがサブスクライブしているトピック宛に値をパブリッシュすることで Raspimouse を操作する．すでにキーボードやゲームコントローラを使って Raspimouse を操作できるノードも配布されている．この Raspimouse の操作だが ROS 2 側では操作できるノードが配布されているため，比較的容易に動作させることができた．しかし，mROS 2-POSIX 側で Raspimouse の制御ノードである ROS 2 のトピックに対してパブリッシュを動作させることができなかった．制御ノードがパブリッシュしている値のサブスクライブは問題なく mROS 2-POSIX 側で可能であった．制御ノードに対して mROS 2-POSIX からのパブリッシュが通らない理由として，QoS 設定の問題がある．

第 4 章

評価

mROS 2-POSIX と ROS 2 の性能を比較評価するにあたって，mROS 2-POSIX に実装できるアプリケーションと同様のアプリケーションを ROS 2 に実装し，比較評価する．本章では，比較評価に用いるアプリケーションの選定基準と，比較評価に用いるアプリケーションの詳細について述べる．

4.1 アプリケーションの選定基準

本研究では，mROS 2-POSIX と ROS 2 の性能を比較評価するにあたって，以下の基準を設けた．

- Pub-Sub 通信のみを使用したアプリケーションであること
- 組み込みデバイス上で動作できるアプリケーションであること

4.1.1 Pub-Sub 通信のみを使用したアプリケーションであること

mROS 2-POSIX は，embeddedRTPS を利用して RTPS を実装している都合上，ROS 2 の Pub-Sub 通信のみの実装となっている．そのため，評価に用いるアプリケーションは Pub-Sub 通信のみを使用したアプリケーションである必要があり，この条件を満たすことで，mROS 2-POSIX と ROS 2 の通信性能を比較評価できる．

4.1.2 組み込みデバイス上で動作できるアプリケーションであること

先行研究での動的配置機構はオーバーヘッドの増加が問題とされ，mROS 2-POSIX を用いたアプローチも提案されているが，性能評価はネットワークスループットのマイクロベンチマークに留まり，実アプリケーションにおける有用性が十分に評価されていない．そのため，評価アプリケーションの実装条件として，動的配置機構が実現される組み込みデバイス

上で動作するアプリケーションでなくてはならない

4.1.3 エンドツーエンドのアプリケーションであること

ユーザー入力によってロボットが動作するアプリケーションを評価に用いることで，より実アプリケーションに近い状況での評価を行うことができるため，エンドツーエンドを条件の一つとした.

4.2 評価手法

第 5 章

実装

本章では，mROS 2-POSIX と ROS 2 の性能を比較評価するにあたって，mROS 2-POSIX と ROS 2 に実装するアプリケーションの概要と，これまで実装予定だったものについて説明する．

5.1 実装アプリケーションの概要

5.2 実装アプリケーションの詳細

5.3 実装アプリケーションの課題と解決策

5.4 実装アプリケーション変更に伴って生じる影響

第 6 章

関連研究

第 7 章

おわりに

謝辞

謝辞を記入する.

参考文献

[1] Reference1
[2] Reference2