OpenELが変える組込みシステム開発

OpenELでIoTデータをクラウドと連携！

2024年11月22日

技術本部　副本部長　兼
プラットフォーム構築委員会　OpenEL活用WG 主査
アップウィンドテクノロジー・インコーポレイテッド
中村憲一

一般社団法人
組込みシステム技術協会
Japan Embedded Systems Technology Association

© Japan Embedded Systems Technology Association 2024

目次

- 組込みシステム開発における課題
- OpenELとは？
- OpenELの実装例
  - 対応デバイスの増加
  - 他システム（ROS2）との連携
- 令和4年度の活動
  - 対応デバイスの増加
  - 仮想シミュレーション環境との連携
- 令和5年度の活動
  - クラウドへの対応（Microsoft Azure IoT Hub）
- 令和6年度の活動
  - 対応デバイスの増加

© Japan Embedded Systems Technology Association

組込みシステム開発における課題

組込みシステム開発における課題

1. 品質保証（Quality guaranteed）
2. 開発コスト(development Cost)の削減
3. 納期(Delivery time)の短縮

■ これらの課題を解決するためにモデルベース開発が採用されているが、モデルベース開発にも課題が存在する。

モデルベース開発における課題

モデルベース開発ツール

アプリケーションの設計

↓

ソースコードを自動生成

↓

~~デバイスドライバのソースコードの自動生成は不可能~~

デバイスドライバのAPIが
各社、各デバイスでバラバラ

↓

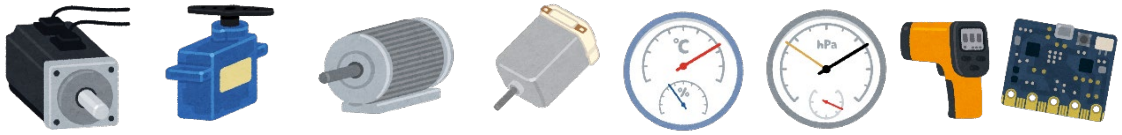
手作業で
1. デバイスドライバを新規開発
   品質の保証は？
2. 各種デバイスドライバと結合
   品質の保証は？

↓

実機試験：不合格

↓

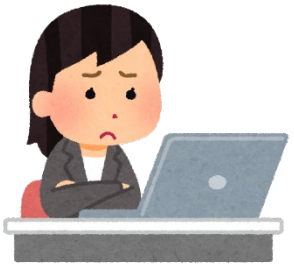
設計・実装見直し
開発コストの増加
納期の遅延



A社製　B社製　C社製　D社製　・・・

一般社団法人
組込みシステム技術協会
Japan Embedded Systems Technology Association

モデルベース開発における課題



- 組込み用の自動コード生成の品質に問題がある。
- 品質を維持するためのノウハウの蓄積が難しい。
- ドライバのコードは生成されないため、ドライバを用意する必要がある。
- なるべくプログラムを書きたくない。
  - 瑕疵担保責任を負いたくない

組込みシステム開発における課題

■ メーカーの課題

- ソフトウェアの開発効率
- ソフトウェアの品質
- 自社製以外のソフトウェアにも保証義務

■ ベンダーの課題

- 他社製のソフトウェアとの相性は保証できない
- ベンダーごとに異なるインターフェース仕様
- ハードウェアならプラグフェストで相互接続試験を実施する機会があるが、ソフトウェアは？

解決策の提案

- アプリケーションプログラミングインターフェース（API）とドライバ用のテンプレートを標準化すれば良い。
  → 開発効率UP！
  → ソースコードの可読性が向上 → レビューが容易
  → バグが入りにくくなる → 品質UP!
- テストパターンを自動生成し、シミュレーターによるテストの実施 → 論理などの単純なバグはここで排除
- テスト後の実行コードを実機にデプロイ
- 実機では実機特有のテストを重点的に実施→ 工数短縮!

- 品質と効率を上げる組込み用のモデルベース開発を実現！

OPENELとは？

OpenEL活用WG

■ 目標

1. OpenELの国内外における普及
2. OpenELの仕様の強化
3. OpenELの国際標準への提案

■ 活動

- WGの開催（毎月）
- OpenELに関連しそうな技術や規格の講演会の開催（年2回）
- ユーザーの増加が見込まれるデバイスやプラットフォームへの対応
- 開発成果の一般公開
- 上流から下流まで一気通貫した開発を実現するための活動

■ メンバー

- 会員(7)：アップウィンドテクノロジー、エヌデーデー、チェンジビジョン、東洋大学、UCサロン、アフレル、金沢工業大学
- 非会員(7)：京セラ、東京大学、静岡大学、シマフジ電機、永和システムマネジメント Knowledge & Experience、日立産機システム

OpenEL®(Open Embedded Library)とは？

■ 目的

- ハードウェアの抽象化を実現し、QCDの向上および上流から下流まで一気通貫した開発を目指す

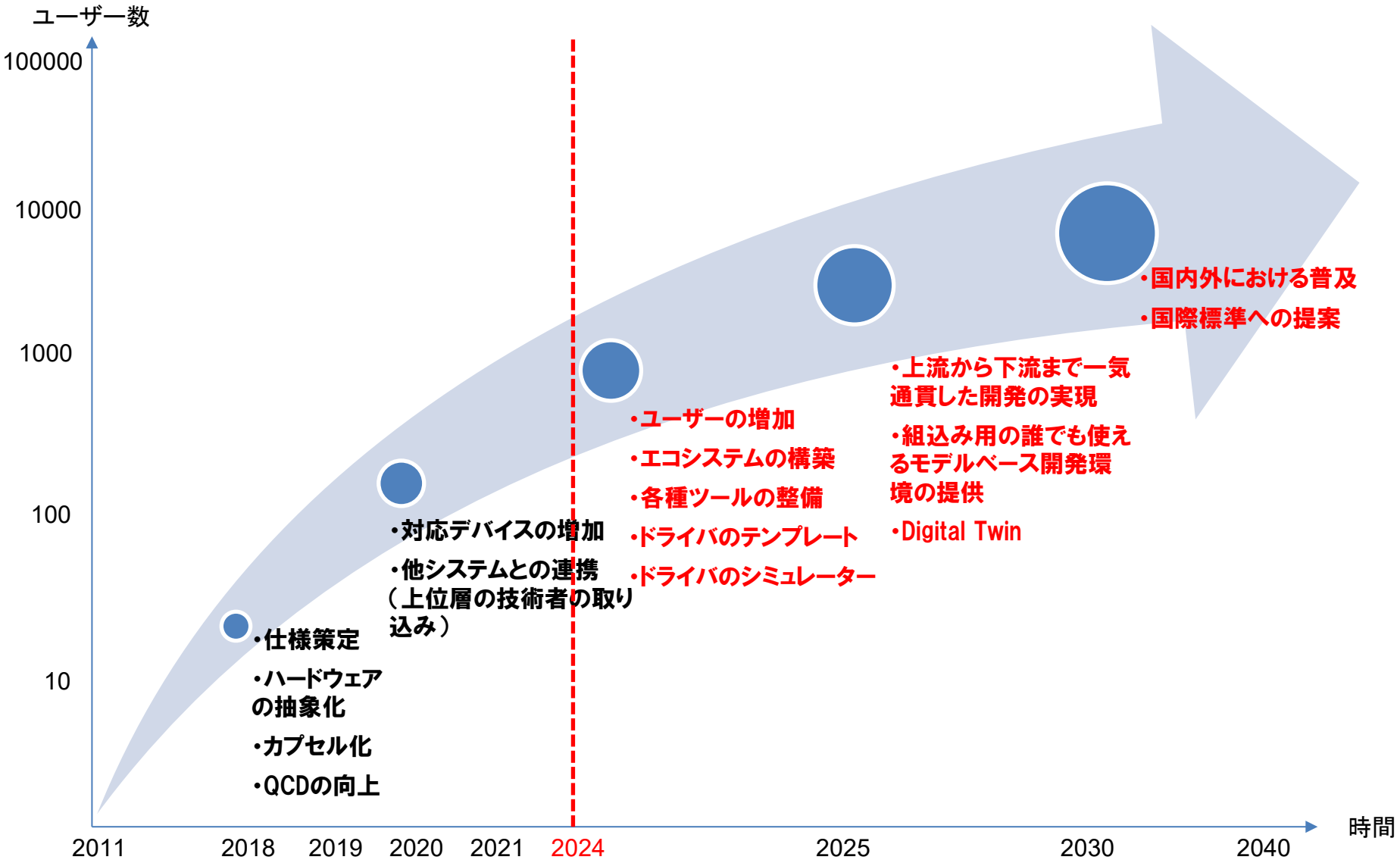
■ 背景

- 組込みソフトウェア技術者の不足
- 製品やデバイスの多様化（多品種少量生産）
- 上位層との標準インターフェースの欠如

■ 課題

- 品質保証（Quality guaranteed）
- 開発コスト(development Cost)
- 納期(Delivery time)

OpenEL®のロードマップ

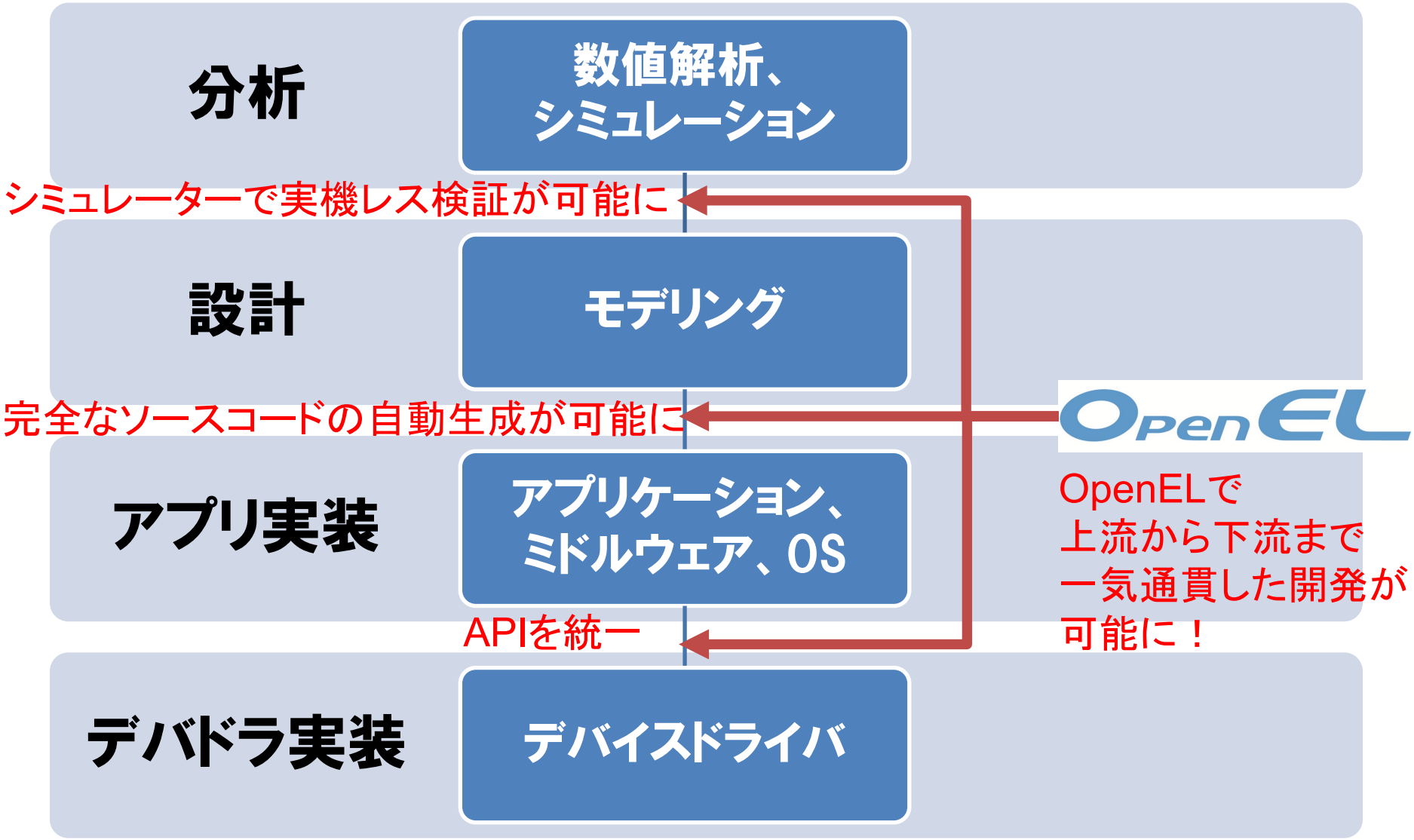


一般社団法人
組込みシステム技術協会
Japan Embedded Systems Technology Association

OpenEL®の何が嬉しいのか？期待される効果

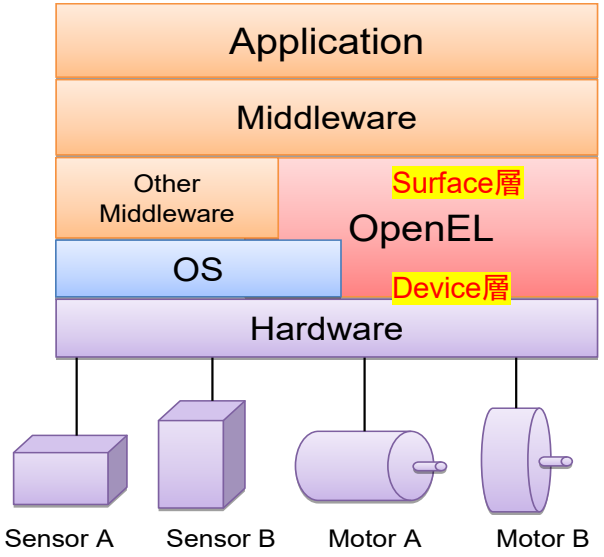
- 品質（Quality ）の向上
  - テスト済みのソフトウェア部品（コンポーネント）を再利用することにより品質を保証！
- 開発コスト（Cost）の削減
  - デバイス毎に異なっていたAPIを学習する期間を削除！
  - ソフトウェア部品（コンポーネント）の新規開発の削減！
  - テスト済みのソフトウェア部品（コンポーネント）を再利用することにより開発期間を短縮！
  - 車輪の再発明を排除！
- 納期（Delivery）の短縮
  - APIが標準化されているため、異なるプラットフォーム、異なるベンダーのデバイスでもソースコードの変更が不要
  - テスト済みのソフトウェア部品（コンポーネント）を組み合わせることにより開発期間とテスト期間を短縮！
- ソフトウェア開発力の強化
  - アクチュエーターやセンサーを専門としない組込みソフトウェア技術者による制御システムの開発が可能に
  - エンタープライズシステムの技術者によるIoTシステムなどの開発も可能に
- 応用範囲(シミュレーターやWebとの接続)の拡大
- ユーザーの増加

OpenEL ®で何がどうなるのか？



OpenEL®(Open Embedded Library)とは？

- 制御システムやIoTデバイスなどのソフトウェアの実装仕様（API）を標準化する組込みシステム向けのオープンなプラットフォーム
- 特徴
  - デバイスの制御に特化し厳選された19個のAPI
  - Surface層とDevice層の複数層によるハードウェアの抽象化を実現
    - デバイスを交換してもアプリケーションのソースコードの修正は不要！
  - Device層によるハードウェア制御のノウハウの隠蔽化を実現
- 実装言語
  - C/C++/C#
- 対応プラットフォーム
  - Non-OS、Embedded RTOS、Linux、Windows、macOS
- 対象ユーザー
  - 組込みシステム技術者から エンタープライズシステム技術者まで



OpenEL®(Open Embedded Library)とは？

- デバイス（センサーやモーター）の接続先が異なっていてもアプリケーション層のソースコードは変更不要！
  - GPIO
  - UART
  - SPI
  - I2C
  - CAN
  - Ethernet
  - Internet
- 実デバイスが存在しなくても仮想シミュレーション環境でテストが可能！
  - 箱庭
- シミュレーション環境と実環境でアプリケーション層のソースコードは同一！
  - HAL ID、OpenEL未対応のデバイス用のコードを除く

OpenEL®の歴史

2011年5月、プラットフォーム研究会ロボットWGにより開発着手

2012年5月、OpenEL 0.1を公開

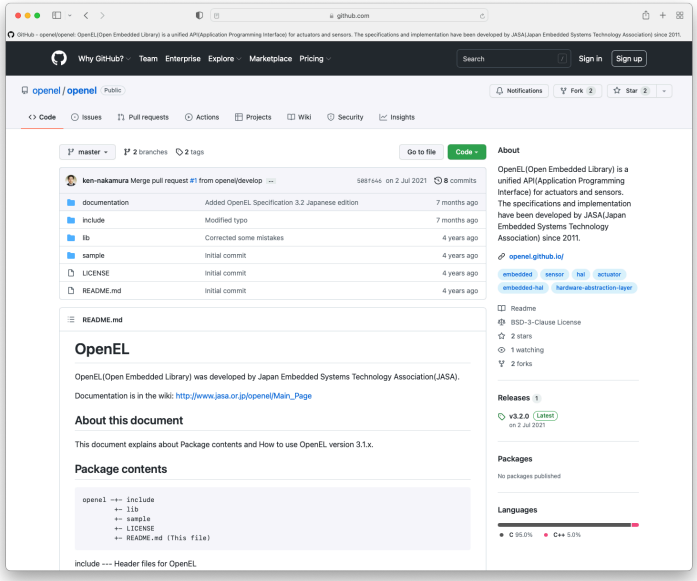
2013年5月、OpenEL 1.0を公開

2015年4月～2018年2月、
経済産業省の国際標準開発事業に採択

2015年7月、OpenEL 2.0を公開

2018年6月、OpenEL 3.1をGitHubで公開

https://github.com/openel/openel



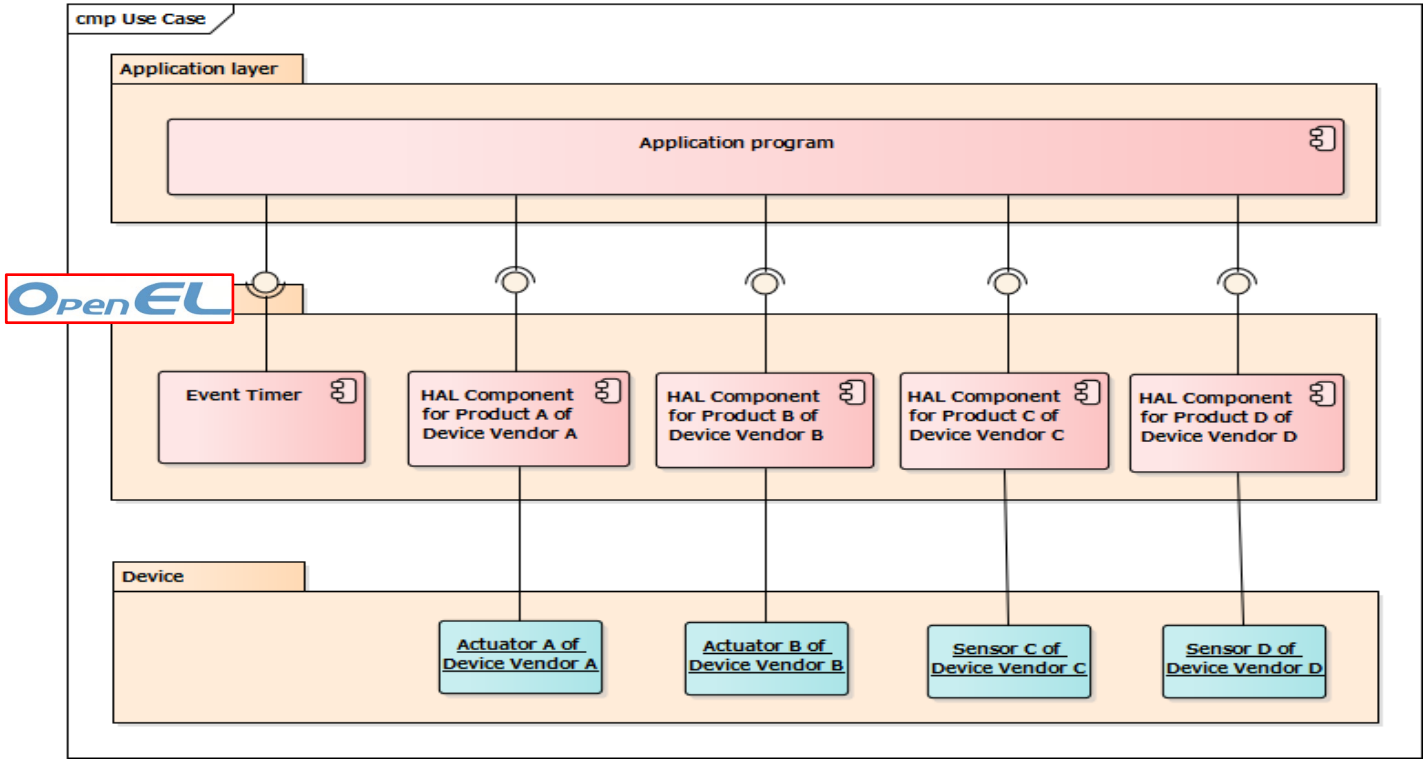
2020年10月～2021年2月、
経済産業省「地域分散クラウド技術開発事業」に採択

2021年7月、OpenEL 3.2（C#版）を公開

2021年7月～2022年2月、
経済産業省「次世代ソフトウェアプラットフォーム実証事業」に採択

2023年5月、OpenEL 3.2(C++版)を公開

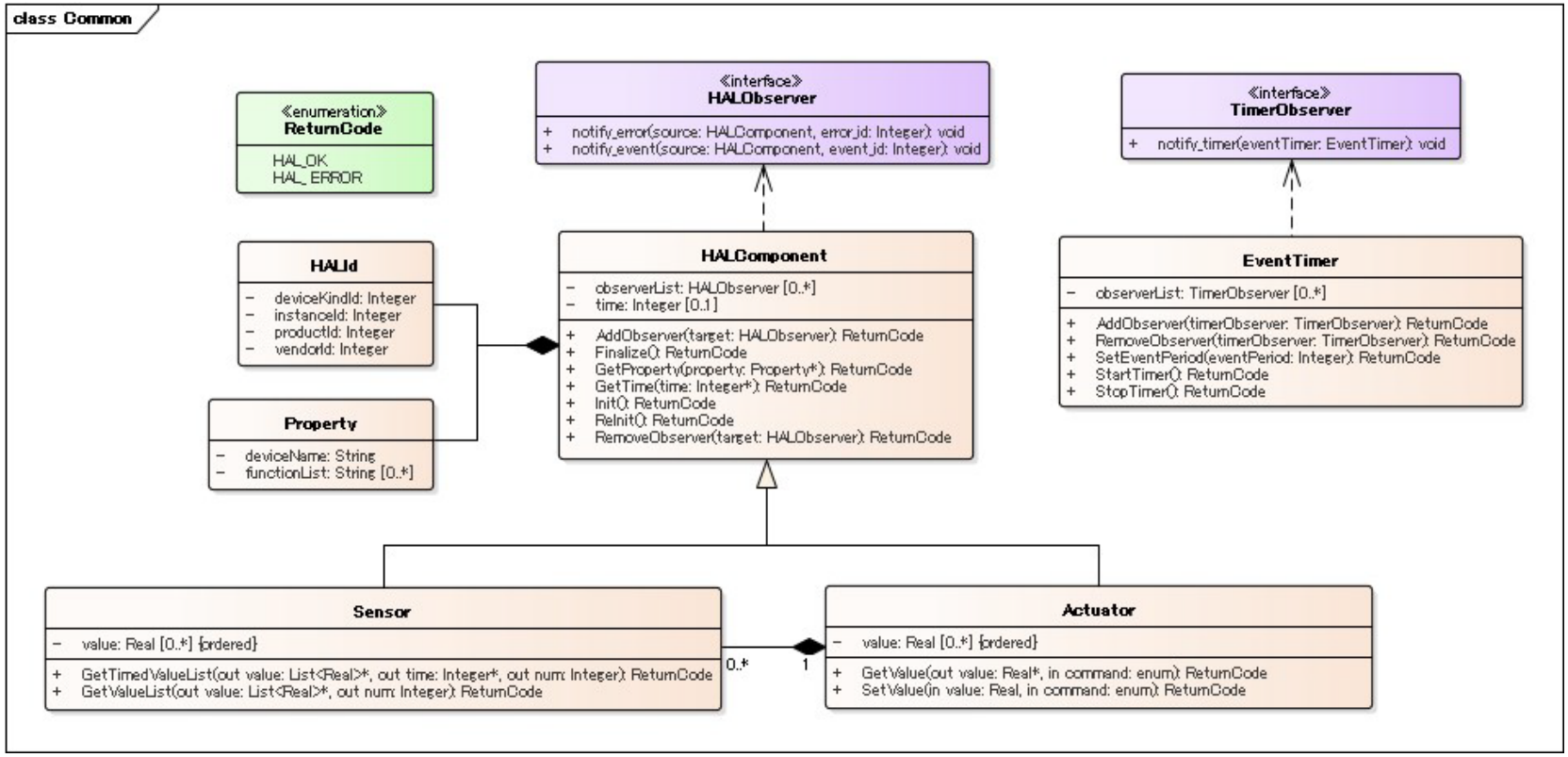
OpenEL® 3.2 仕様



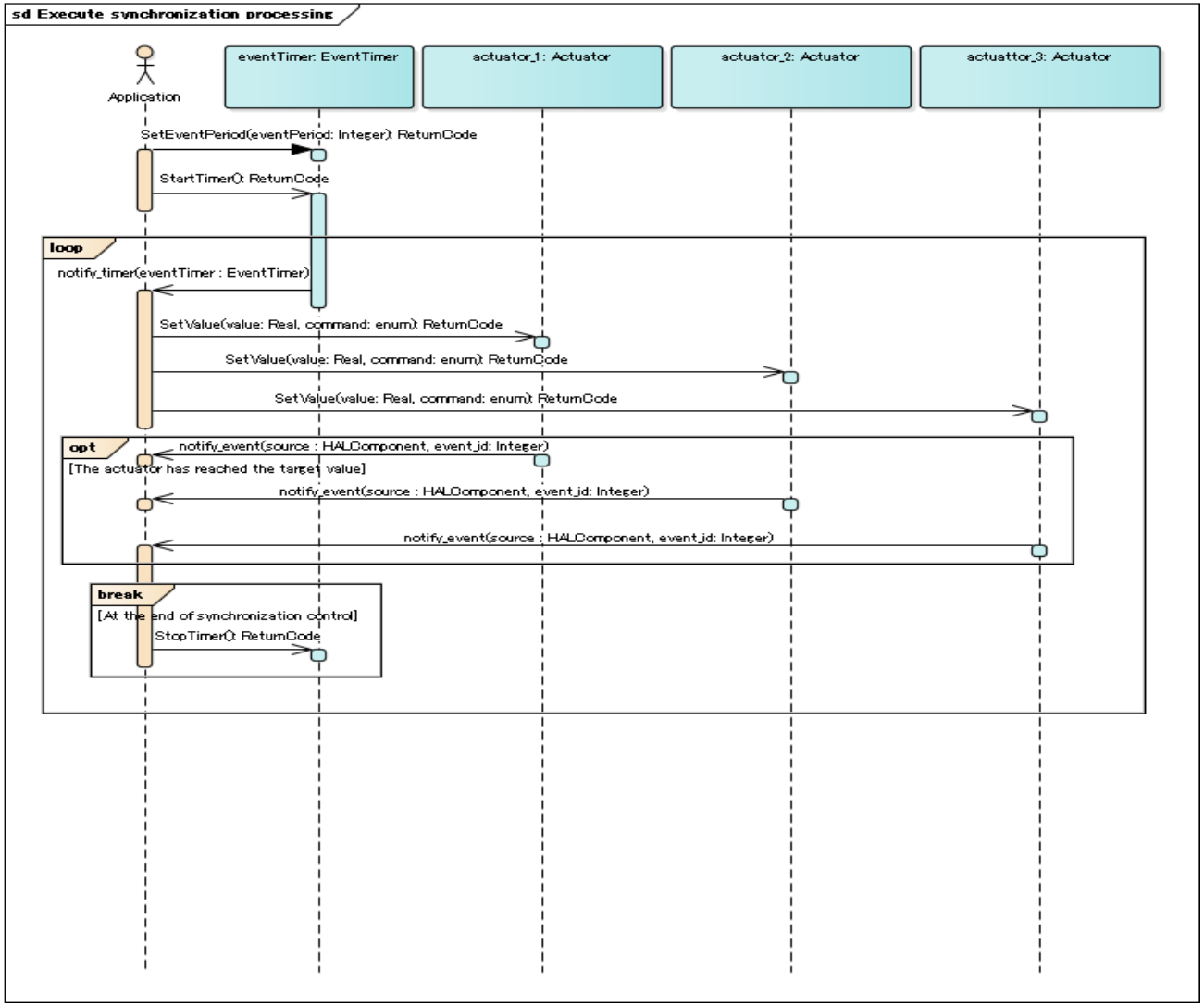
Surface層

Device層

OpenEL® 3.2 仕様


class Common
«enumeration»
ReturnCode
HAL_OK
HAL_ERROR
«interface»
HALObserver
+ notify_error(source: HALComponent, error_id: Integer): void
+ notify_event(source: HALComponent, event_id: Integer): void
«interface»
TimerObserver
+ notify_timer(eventTimer: EventTimer): void
HALId
- deviceKindId: Integer
- instanceId: Integer
- productId: Integer
- vendorId: Integer
HALComponent
- observerList: HALObserver [0..*]
- time: Integer [0..1]
+ AddObserver(target: HALObserver): ReturnCode
+ Finalize(): ReturnCode
+ GetProperty(property: Property*): ReturnCode
+ GetTime(time: Integer*): ReturnCode
+ Init(): ReturnCode
+ ReInit(): ReturnCode
+ RemoveObserver(target: HALObserver): ReturnCode
EventTimer
- observerList: TimerObserver [0..*]
+ AddObserver(timerObserver: TimerObserver): ReturnCode
+ RemoveObserver(timerObserver: TimerObserver): ReturnCode
+ SetEventPeriod(eventPeriod: Integer): ReturnCode
+ StartTimer(): ReturnCode
+ StopTimer(): ReturnCode
Property
- deviceName: String
- functionList: String [0..*]
Sensor
- value: Real [0..*] {ordered}
+ GetTimedValueList(out value: List<Real>*, out time: Integer*, out num: Integer): ReturnCode
+ GetValueList(out value: List<Real>*, out num: Integer): ReturnCode
0..*
1
Actuator
- value: Real [0..*] {ordered}
+ GetValue(out value: Real*, in command: enum): ReturnCode
+ SetValue(in value: Real, in command: enum): ReturnCode

OpenEL® 3.2 仕様


sd Execute synchronization processing
Application
eventTimer: EventTimer
actuator_1: Actuator
actuator_2: Actuator
actuattor_3: Actuator
SetEventPeriod(eventPeriod: Integer): ReturnCode
StartTimer(): ReturnCode
loop
notify_timer(eventTimer : EventTimer)
SetValue(value: Real, command: enum): ReturnCode
SetValue(value: Real, command: enum): ReturnCode
SetValue(value: Real, command: enum): ReturnCode
opt
[The actuator has reached the target value]
notify_event(source : HALComponent, event_id: Integer)
notify_event(source : HALComponent, event_id: Integer)
notify_event(source : HALComponent, event_id: Integer)
break
[At the end of synchronization control]
StopTimer(): ReturnCode

OpenEL® 3.2 仕様（C言語）

■　コンポーネント共通

```
HALRETURNCODE_T HalInit(HALCOMPONENT_T *halComponent);
HALRETURNCODE_T HalReInit(HALCOMPONENT_T *halComponent);
HALRETURNCODE_T HalFinalize(HALCOMPONENT_T *halComponent);
HALRETURNCODE_T HalAddObserver(HALCOMPONENT_T *halComponent, HALOBSERVER_T *halObserver);
HALRETURNCODE_T HalRemoveObserver(HALCOMPONENT_T *halComponent, HALOBSERVER_T *halObserver);
HALRETURNCODE_T HalGetProperty(HALCOMPONENT_T *halComponent, HALPROPERTY_T *property);
HALRETURNCODE_T HalGetTime(HALCOMPONENT_T *halComponent, int32_t *timeValue);
```

■　イベントタイマー

```
HALRETURNCODE_T HalEventTimerStartTimer(HALEVENTTIMER_T *eventTimer);
HALRETURNCODE_T HalEventTimerStopTimer(HALEVENTTIMER_T *eventTimer);
HALRETURNCODE_T HalEventTimerSetEventPeriod(HALEVENTTIMER_T *eventTimer, int32_t eventPeriod);
HALRETURNCODE_T HalEventTimerAddObserver(HALEVENTTIMER_T *eventTimer, HALTIMEROBSERVER_T *timerObserver);
HALRETURNCODE_T HalEventTimerRemoveObserver(HALEVENTTIMER_T *eventTimer, HALTIMEROBSERVER_T *timerObserver);
```

OpenEL® 3.2 仕様（C言語）

■　　モーター制御

```
#define HAL_REQUEST_NO_EXCITE                    (0)
#define HAL_REQUEST_POSITION_CONTROL     (1)
#define HAL_REQUEST_VELOCITY_CONTROL     (2)
#define HAL_REQUEST_TORQUE_CONTROL       (3)
#define HAL_REQUEST_POSITION_ACUTUAL     (5)
#define HAL_REQUEST_VELOCITY_ACUTUAL     (6)
#define HAL_REQUEST_TORQUE_ACUTUAL       (7)
```

```
HALRETURNCODE_T HalActuatorSetValue(HALCOMPONENT_T *halComponent, int32_t request, HALFLOAT_T value);
```

目標角度/位置、目標角速度/速度、目標トルクまでモータを動作させる. 本メソッドは非同期であり, モータが目標値に到達するまでは待たない. モータが目標値に到達する前に, 再度本メソッドが呼ばれた場合には, 目標値が更新される. モータが目標値に到達したことは, HALObserver を使用してアプリケーション側に通知される.ただし, この通知は, 最終的な 目標値に到達した場合のみ発行される.このため, 本メソッドを複数回呼び出し, 目標値が更新された場合には, 最終的な目 標値を設定したメソッドに対応した通知のみ行われる.

```
HALRETURNCODE_T HalActuatorGetValue(HALCOMPONENT_T *halComponent, int32_t request, HALFLOAT_T *value);
```

対象モータの現在角度/位置、現在角速度/速度、現在トルク/力を取得する.
現在値が測定できない要素の場合には, 推定値もしくは指令値を返す.

位置 単位[rad] または [m](製品仕様)
速度 単位[rad/s] または [m/s] (製品仕様)
トルク 単位[N・m] または [N] (製品仕様)

OpenEL® 3.2 仕様（C言語）

■　センサー入力

HALRETURNCODE_T HalSensorGetValueList(HALCOMPONENT_T *halComponent,int32_t *size, HALFLOAT_T *valueList);

センサーから値を取得する

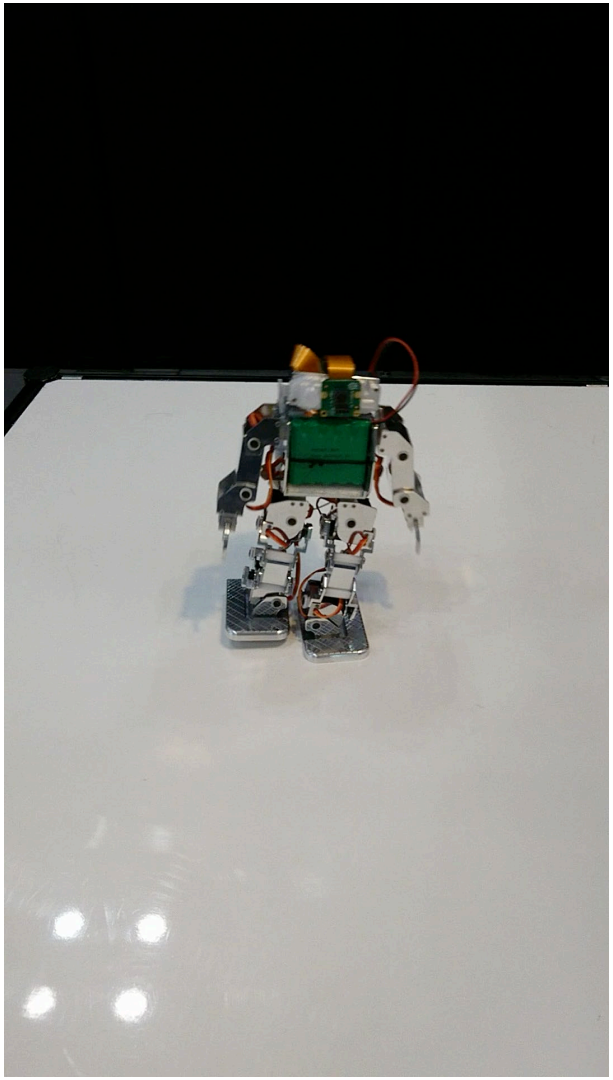
HALRETURNCODE_T HalSensorGetTimedValueList(HALCOMPONENT_T *halComponent,int32_t *size, HALFLOAT_T *valueList, int32_t *timeValue);

センサーから値と時間情報を取得する

OPENELの実装例

OpenELの実装例（C言語）

- 二足歩行ロボット
  UTRX-17
- Raspberry Pi Zero W
  + PCA9685
- OpenELでサーボモーターを制御

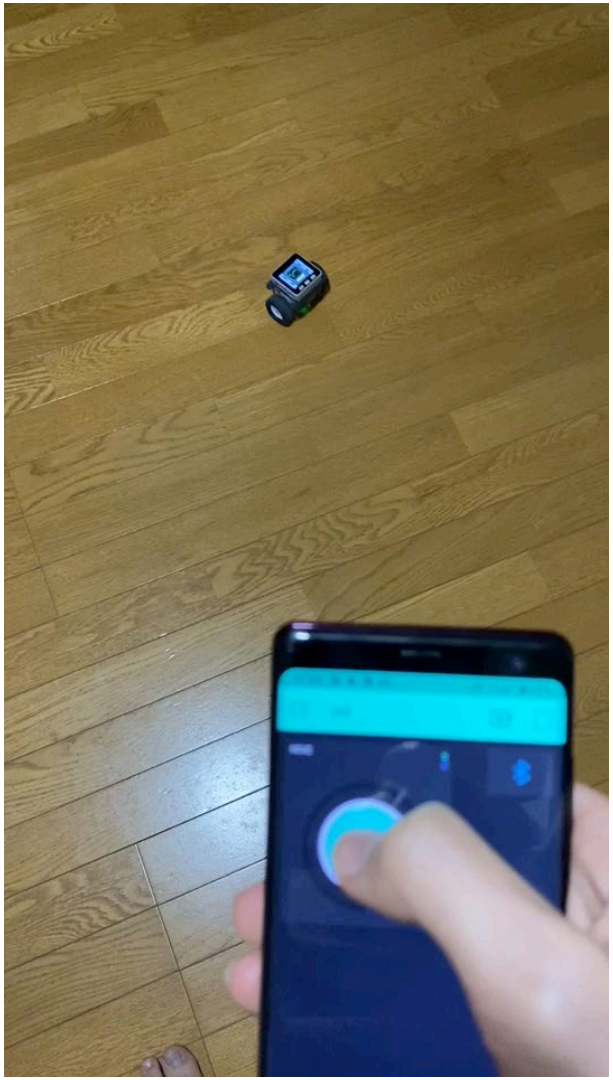


OpenELの実装例（C++言語）

- M5Stack FIRE + M5BALA
- M5Stack FIREにSH200Q（3軸加速度センサー、3軸ジャイロ）が搭載
- OpenELで倒立2輪ロボットを制御
- Arduino IDEで開発

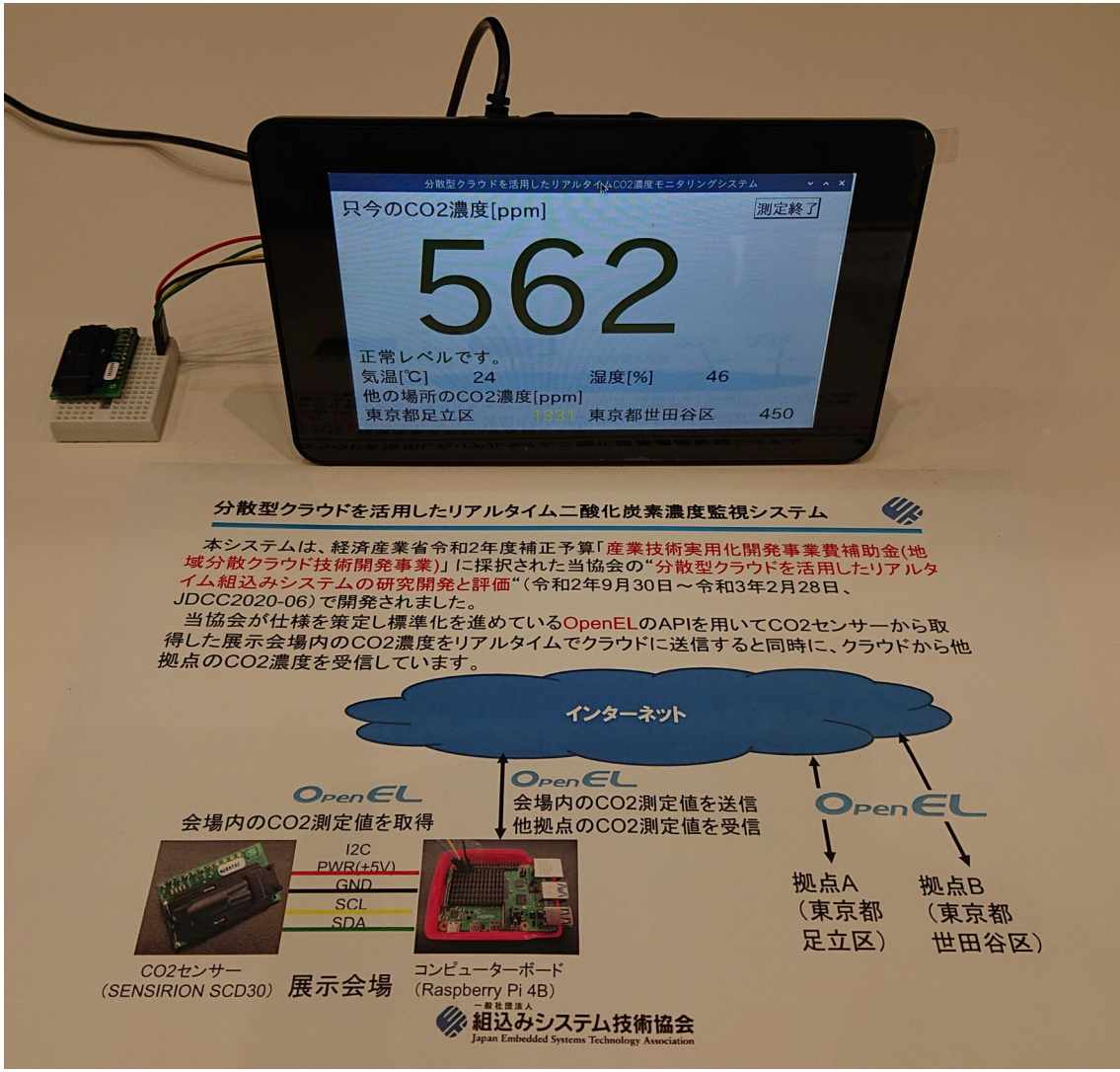
M5Stack BALA2

- M5Stack Gray + BALA2
- M5Stack GrayにMPU6886（3軸加速度センサー、3軸ジャイロ）が搭載
- OpenEL(C++)で倒立2輪ロボットを制御
- Arduino IDEで開発



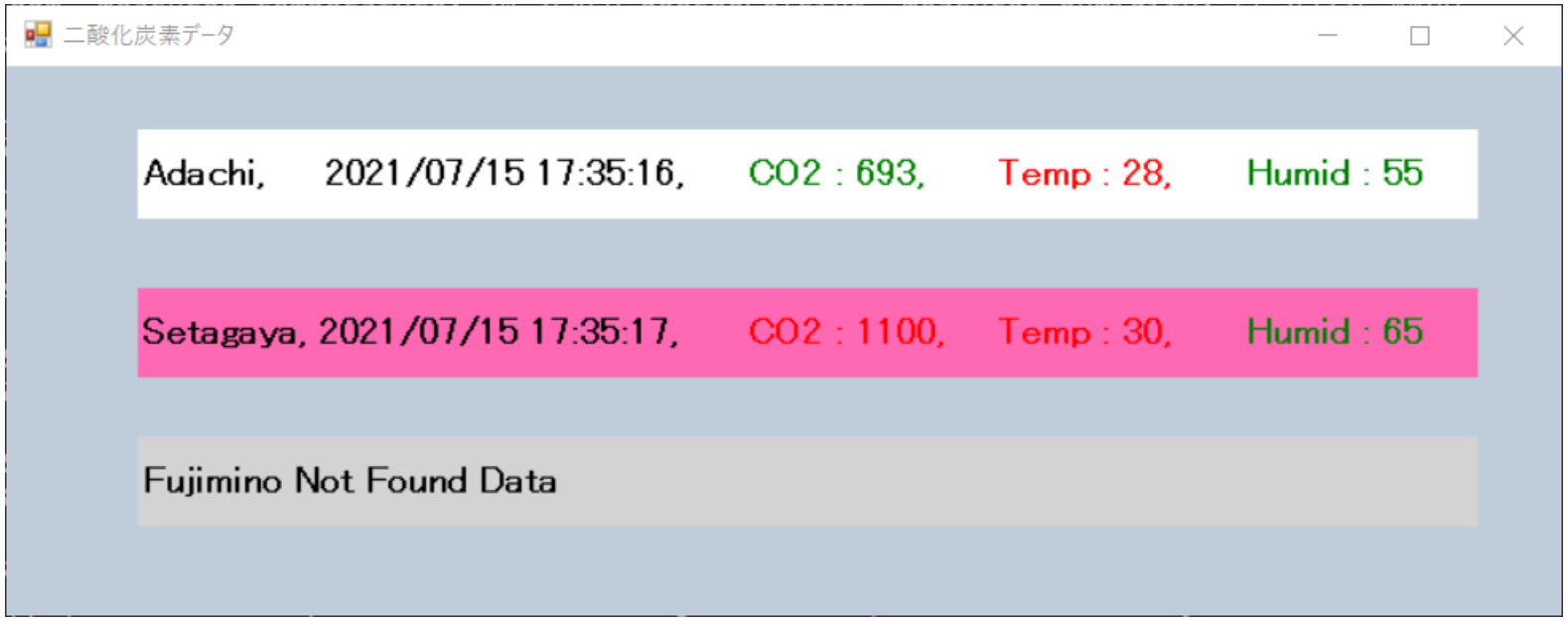
C#への対応

- 二酸化炭素センサー（Sensirion SCD30）に対応したことにより、換気具合の可視化「密の見える化」を実現
- C#に対応したことにより、OpenELコンポーネント内部でネットワーク通信の利用を実現
- 大規模リアルタイム通信エンジン「Diarkis」に対応したことにより、遠隔地のCO2濃度の測定を実現
- 拠点間で相互にCO2濃度のリアルタイム監視を実現



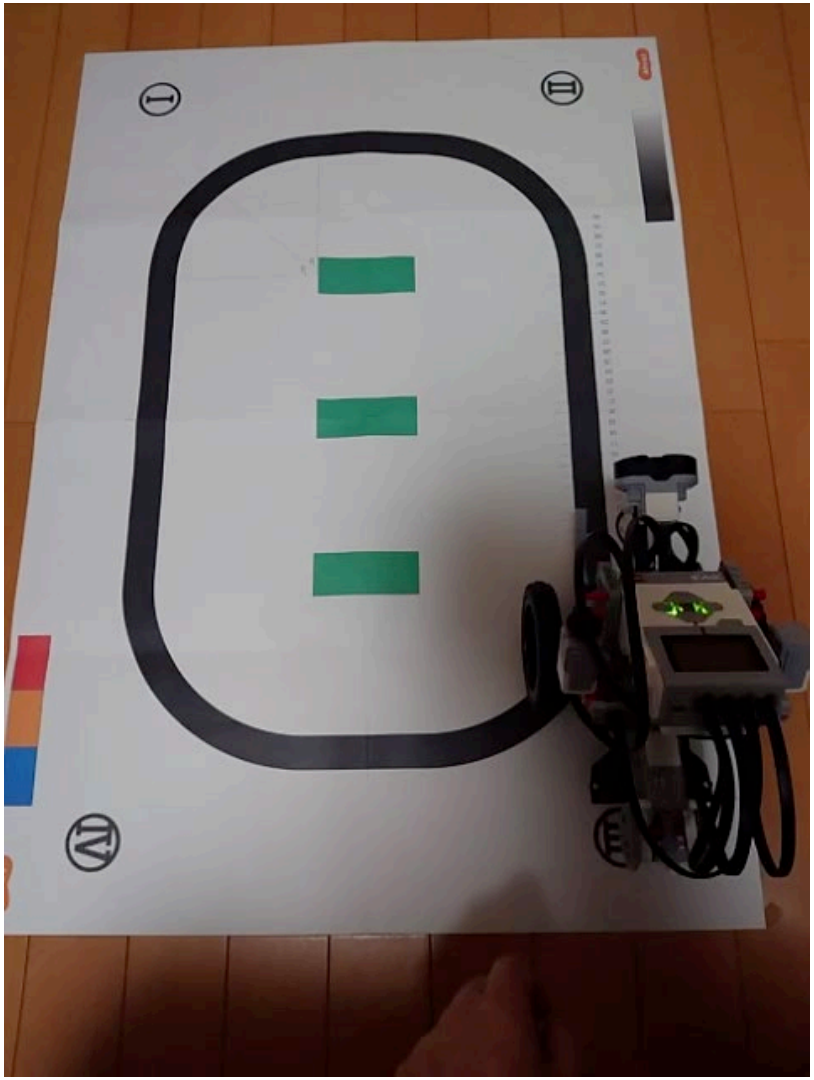
C#への対応：Windows GUIアプリの例

C#への対応によりWindowsのGUIアプリでもOpenELの利用が可能に！



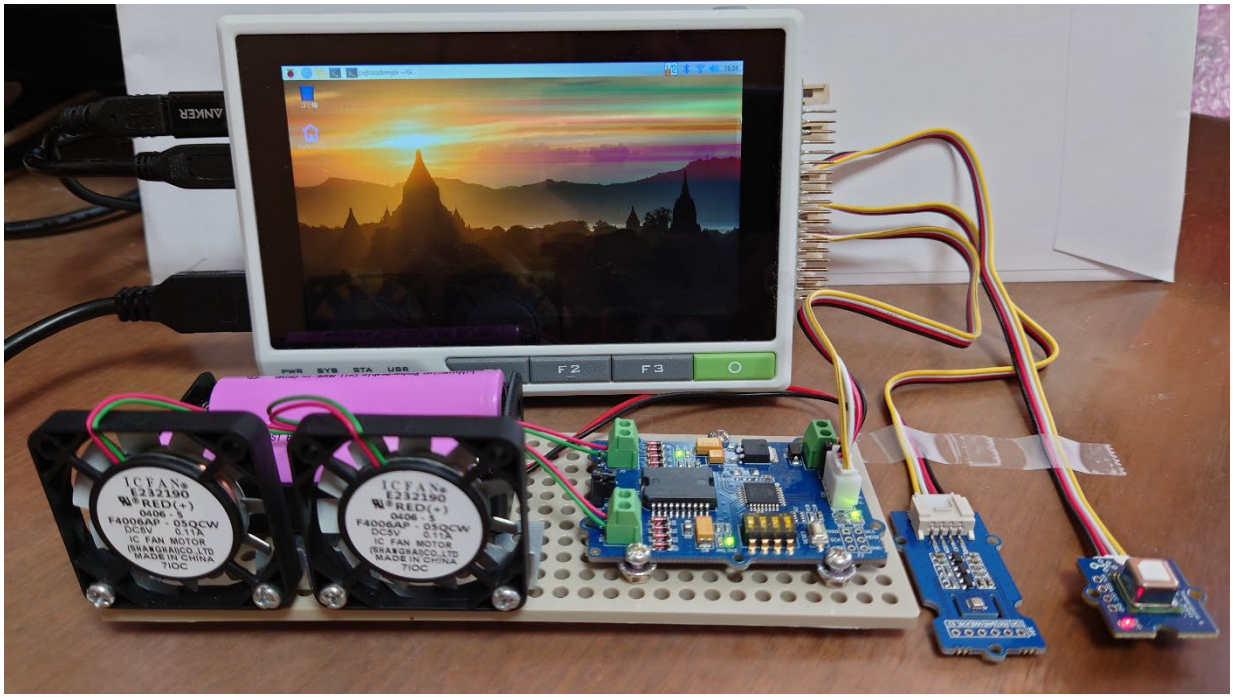
LEGO EV3

- ev3dev環境(Debian Linuxベース)
- モーター3個、光センサー、タッチセンサー、距離センサー、ジャイロセンサーに対応
- ETロボコン走行体でライントレースを実現

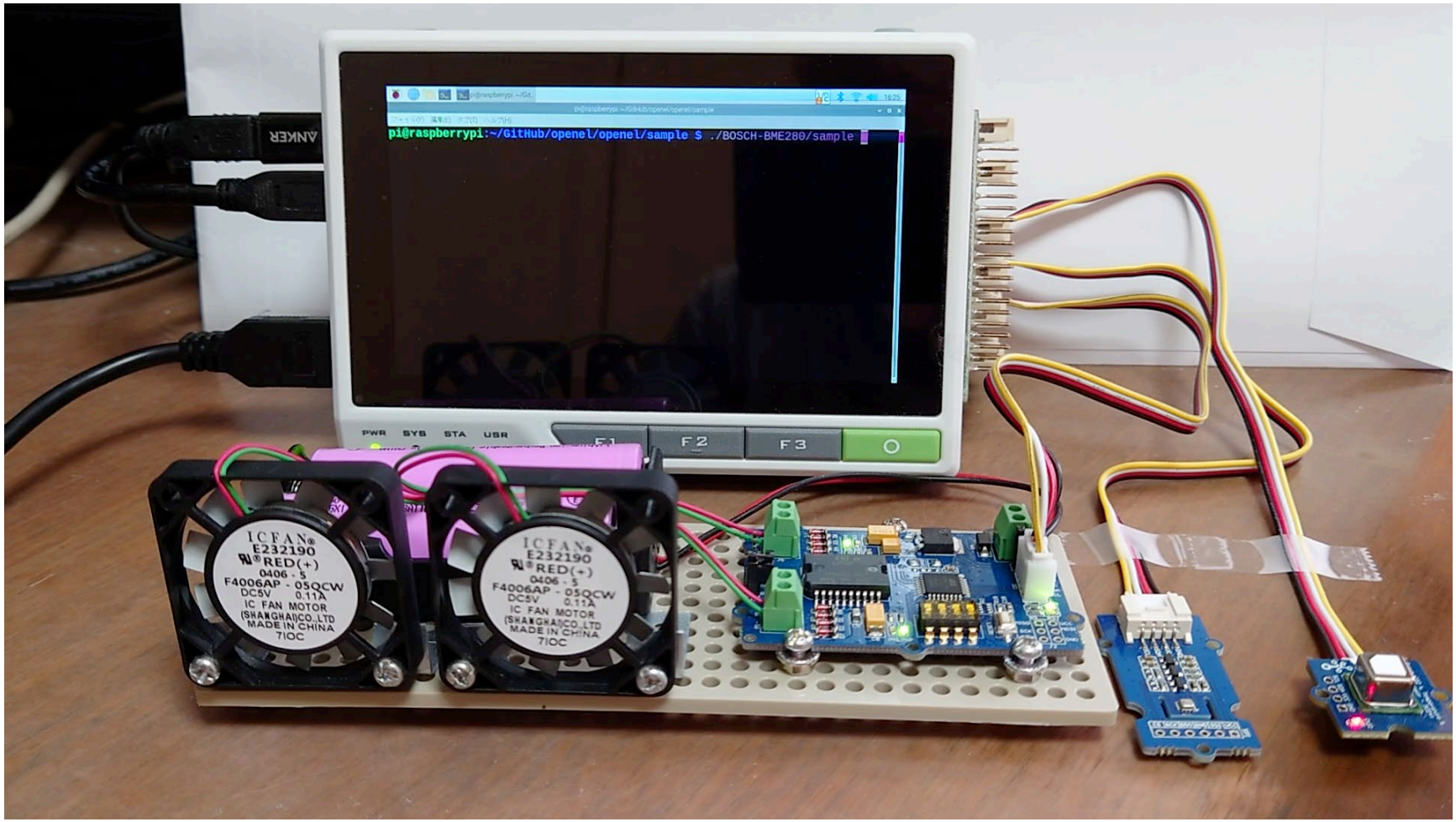


reTerminal

- Raspberry Pi CM4搭載
- 温度/湿度/気圧センサー　BOSCH BME280
- CO2/温度/湿度センサー　Sensirion SCD41
- I2Cモータードライバ＋DCファンモーター
- 二酸化炭素濃度を検知して自動的に換気を行うシステムを構築

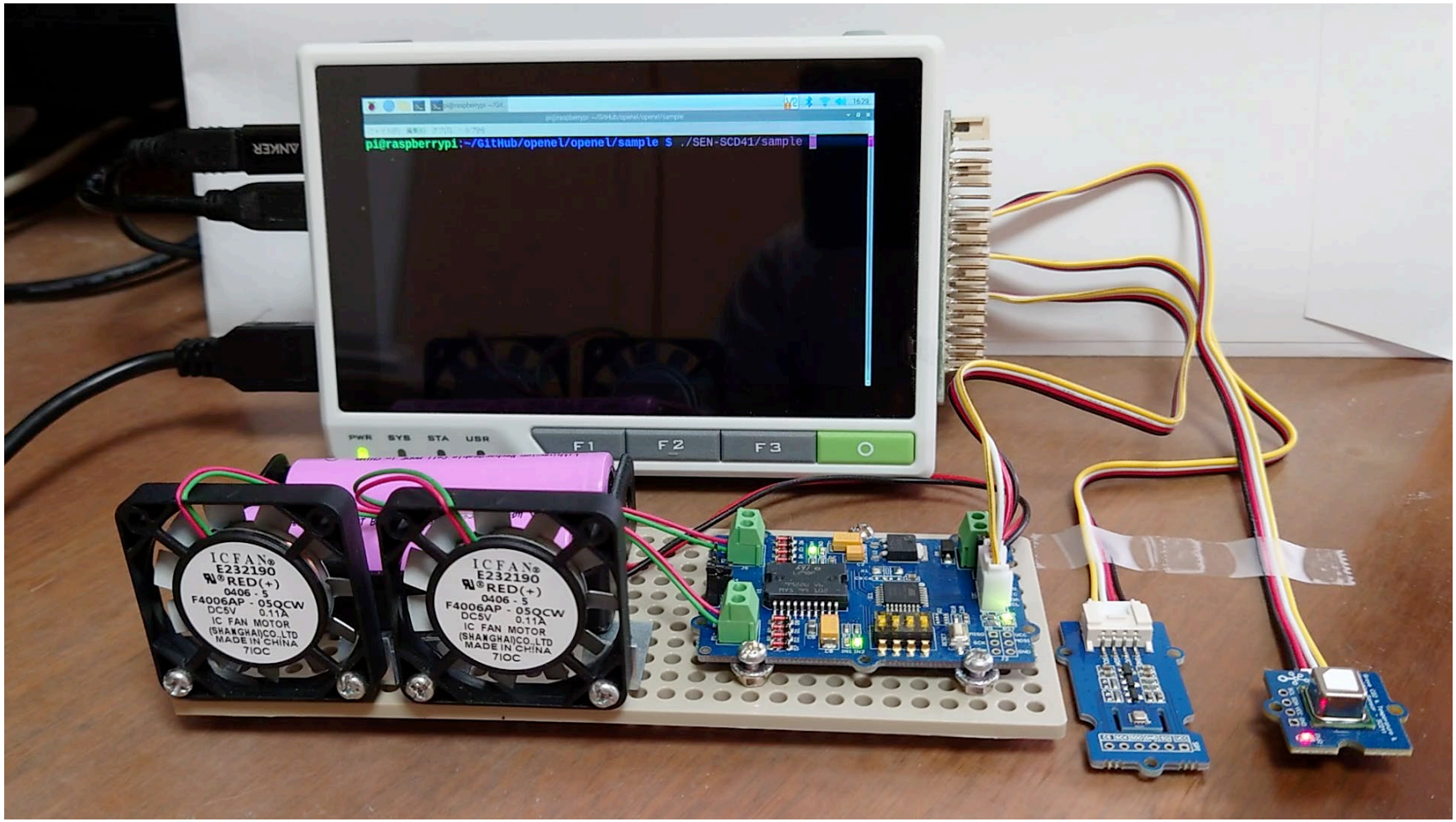


温度/湿度/気圧センサーのデモ

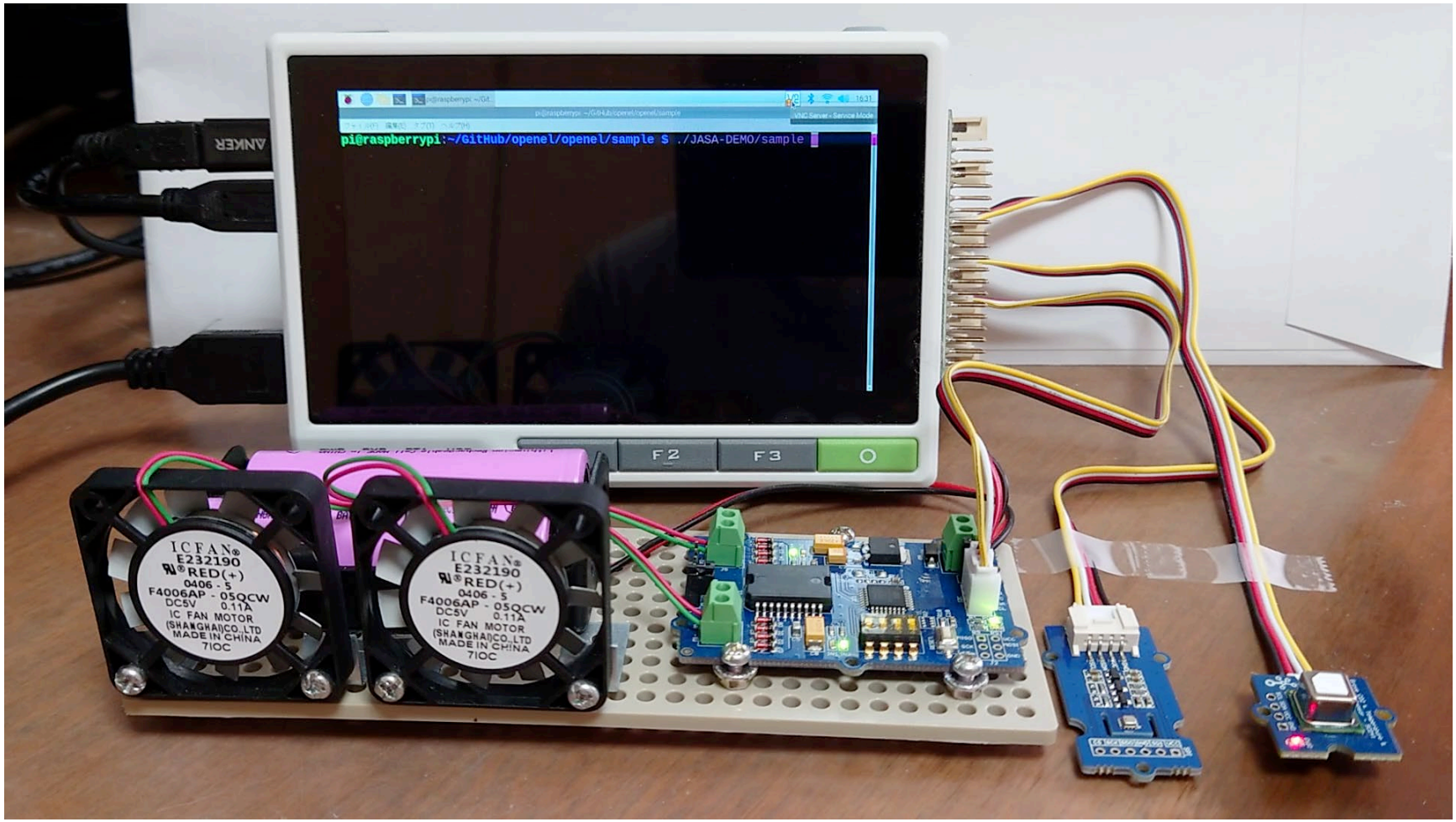


BME280で気圧の変化を測定

CO2/温度/湿度センサーのデモ

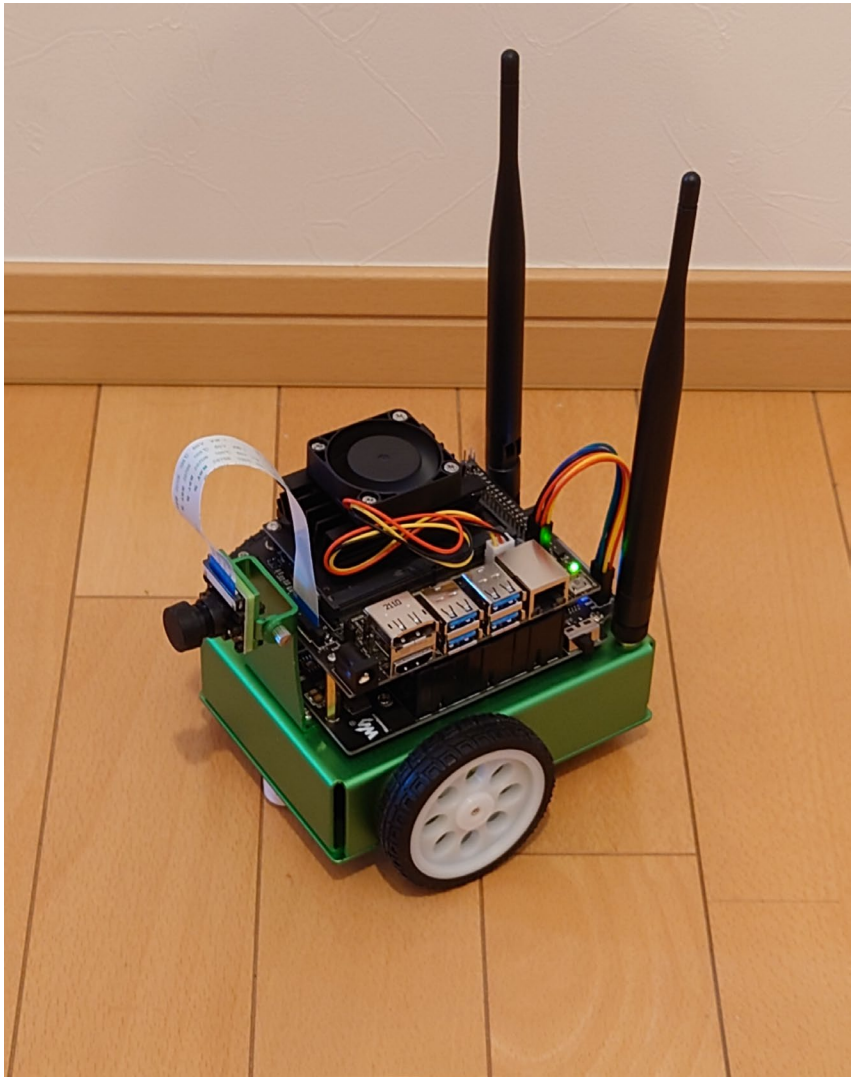


CO2濃度を検知する自動換気システムのデモ

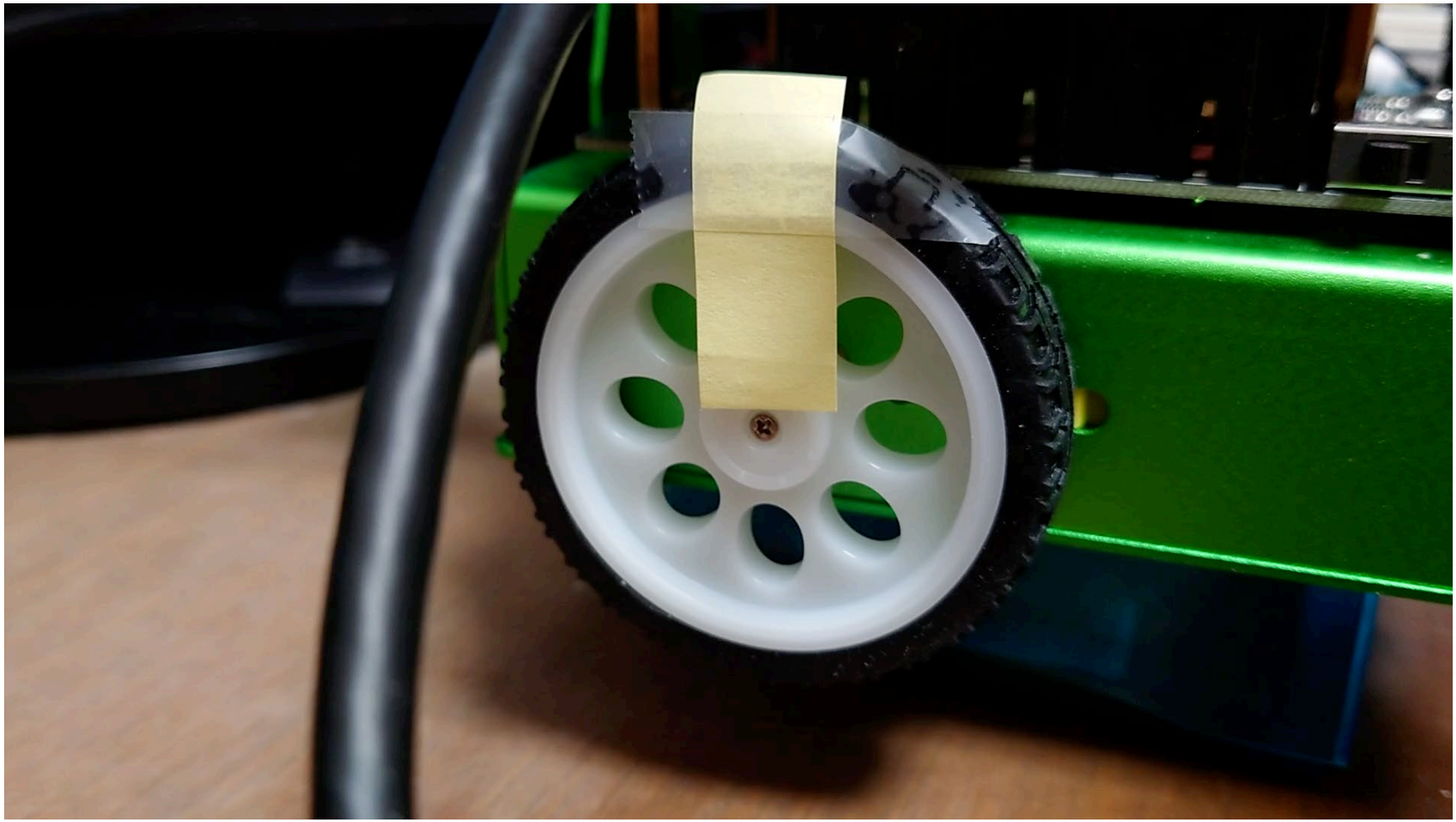


経済産業省「次世代ソフトウェアプラットフォーム実証事業」

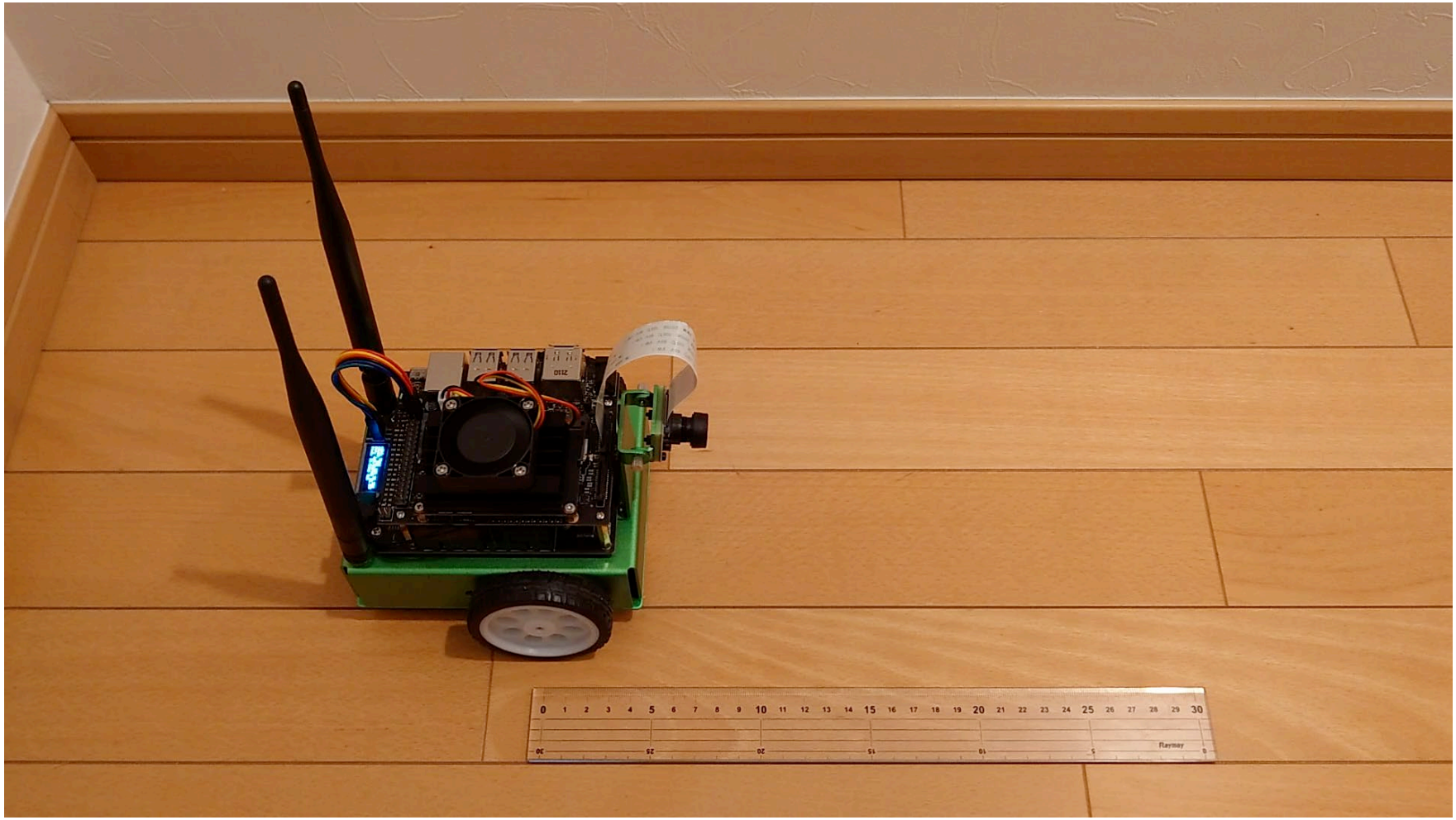
- NVIDIA JetBotの追加
  - Vendor IDの追加（0xB：NVIDIA）
  - Product IDの追加（0x1：JetBot）
  - Device Kind IDの追加（0xD：電流センサー）
- OpenEL®コンポーネントの追加
  - LEDディスプレイドライバ NXP PCA9685（モーター制御）
  - 電流センサー TI INA219（バッテリー監視）
- ROS2に対応
  - OpenELノード（Publisher/Subscriber）の追加
  - 機体番号、前後進距離[cm]、旋回角度[deg]
- ROS2に対応したdockerイメージの作成



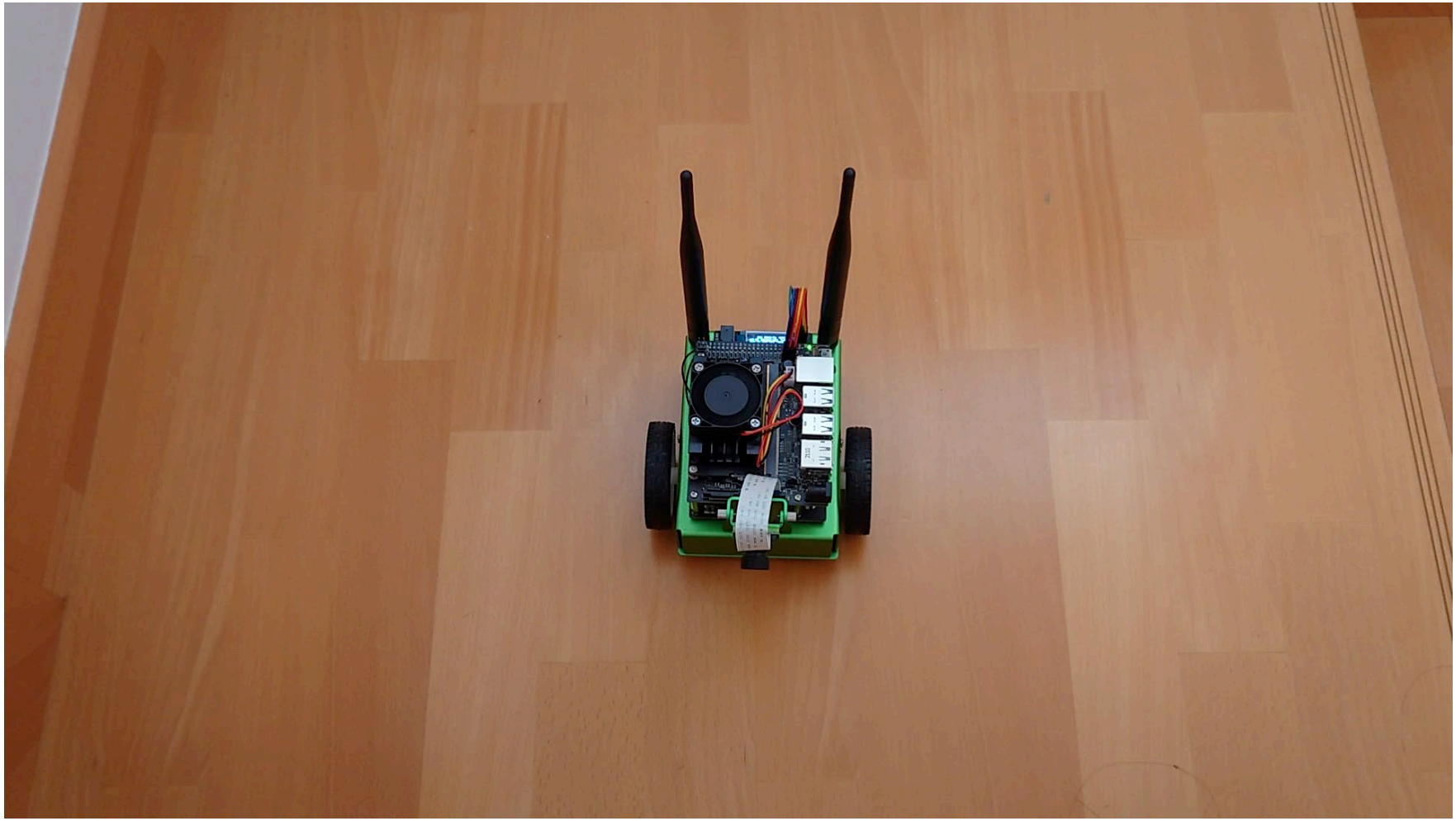
速度制御試験(−15π～15π[rad/s])



前後進試験



旋回試験



自己位置推定(Odometry)による移動

電力管理機能

応用例：バッテリーの電圧降下を検知してモーターの回転速度を制御することも可能

車輪回転速度
-15π～15π[rad/s]　負荷電圧[V]

nakamura — jetbot@nano-4gb-jp45: ~/upwind-technology/openel-dev/sa...

```
timer    50 ,    50 ,     6:  35.565  35.565  0.000 11.916  0.000  0.000
timer    51 ,    51 ,     6:  36.099  36.099  0.000 11.916  0.000  0.000
timer    52 ,    52 ,     6:  36.622  36.622  0.000 11.916  0.000  0.000
timer    53 ,    53 ,     6:  37.134  37.134  0.000 11.916  0.000  0.000
timer    54 ,    54 ,     6:  37.635  37.635  0.000 11.916  0.000  0.000
timer    55 ,    55 ,     7:  38.124  38.124  0.000 11.900  0.000  0.000
timer    56 ,    56 ,     7:  38.602  38.602  0.000 11.900  0.000  0.000
timer    57 ,    57 ,     7:  39.067  39.067  0.000 11.900  0.000  0.000
timer    58 ,    58 ,     7:  39.521  39.521  0.000 11.900  0.000  0.000
timer    59 ,    59 ,     7:  39.963  39.963  0.000 11.900  0.000  0.000
timer    60 ,    60 ,     7:  40.393  40.393  0.000 11.900  0.000  0.000
timer    61 ,    61 ,     7:  40.810  40.810  0.000 11.900  0.000  0.000
timer    62 ,    62 ,     7:  41.215  41.215  0.000 11.900  0.000  0.000
timer    63 ,    63 ,     7:  41.608  41.608  0.000 11.900  0.000  0.000
notify_event201a : 1
Charging.
notify_event201a : 2
Charging completed.
timer    64 ,    64 ,     8:  41.988  41.988  0.174 12.356 21.546  1.756
timer    65 ,    65 ,     8:  42.355  42.355  0.174 12.356 21.546  1.756
timer    66 ,    66 ,     8:  42.709  42.709  0.174 12.356 21.546  1.756
timer    67 ,    67 ,     8:  43.050  43.050  0.174 12.356 21.546  1.756
timer    68 ,    68 ,     8:  43.378  43.378  0.174 12.356 21.546  1.756
```

充電検知

充電完了（12V以上）検知

シャント電圧[V]

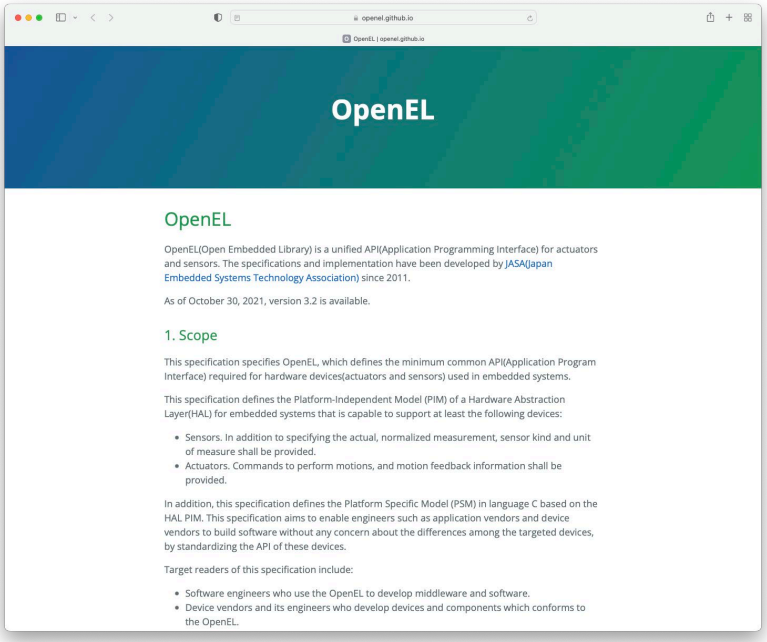
電力[W]　電流[A]

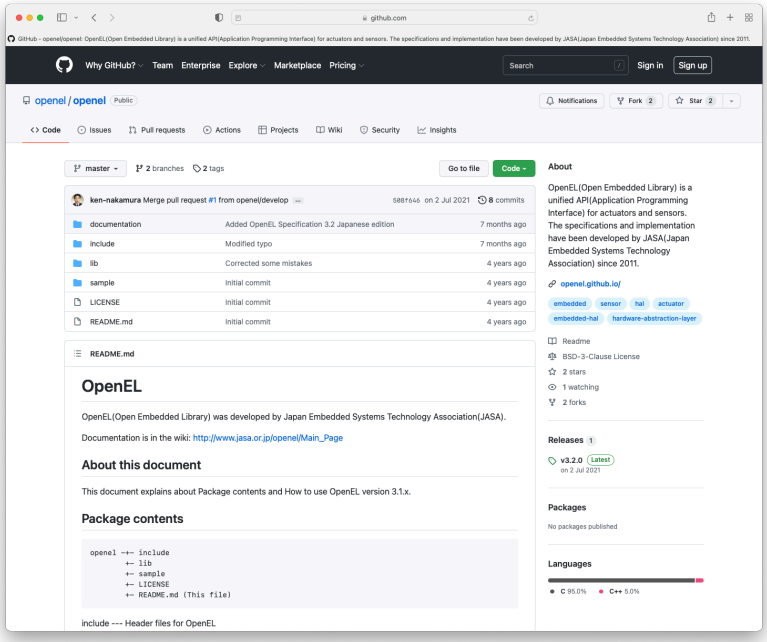
nakamura — jetbot@nano-4gb-jp45: ~/upwind-technology/openel-dev/sa...

```
jetbot@nano-4gb-jp45:~/upwind-technology/openel-dev/sample/NVIDIA-JETBOT$ ./sample
```

開発成果をGitHubで広く一般に公開

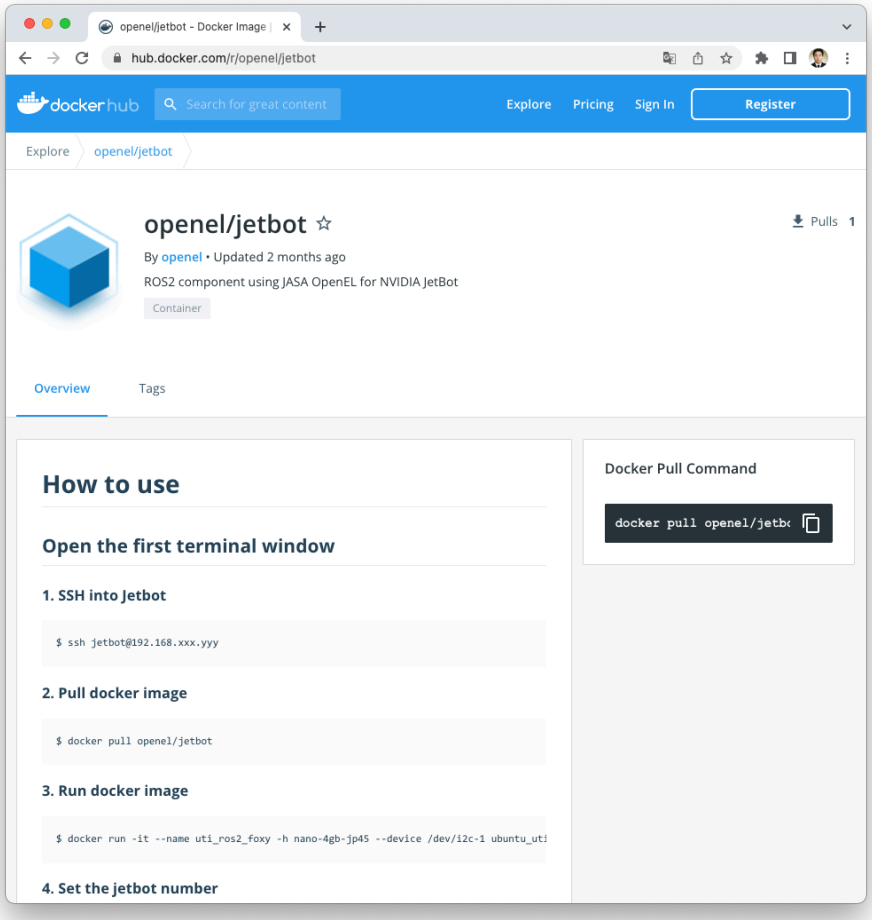
- M5Stack BALA2、EV3、JetBot向けの実装を公開！
- 日本語仕様書以外に、英語仕様書やチュートリアルも追加
- https://github.com/openel/openel





開発成果をdockerhubで広く一般に公開

- ROS2をセットアップ済みのJetBot向けDockerイメージを公開！
- JetBotにダウンロードするだけで、すぐにOpenELが利用可能！
- コンパイル不要！
- ROS2を起動してメッセージをPublishするだけでJetBotが動作！
- https://hub.docker.com/r/openel/jetbot



Dockerイメージの使い方(Subscriber)

Open the first terminal window

1. SSH into Jetbot

```
$ ssh jetbot@192.168.xxx.yyy
```

2. Pull docker image

```
$ docker pull openel/jetbot
```

3. Run docker image

```
$ docker run -it --name uti_ros2_foxy -h nano-4gb-jp45 --device /dev/i2c-1
ubuntu_uti:20220119 /bin/bash
```

4. Set the jetbot number

```
root@nano-4gb-jp45:/# cat /home/jetbot/dev_ws/src/openel_pubsub/openel_subscriber.yaml
/openel_subscriber:
 ros_parameters: jetbot_number: 1 <--- Change this number to the one which you want to use.
 use_sim_time: false
```

5. Set up ROS2 environment

```
root@nano-4gb-jp45:/# source /opt/ros/foxy/setup.bash
root@nano-4gb-jp45:/# source /home/jetbot/dev_ws/install/setup.bash
```

6. Run OpenEL package

```
root@nano-4gb-jp45:/# ros2 run openel_pubsub openel_listener --ros-args --params-file
/home/jetbot/dev_ws/src/openel_pubsub/openel_subscriber.yaml
[INFO] [1642562106.391263209] [openel_subscriber]: My jetbot_number: 1
```

Dockerイメージの使い方(Publisher)

1. SSH into Jetbot

$ ssh jetbot@192.168.xxx.yyy

2. Connect to the running container

jetbot@nano-4gb-jp45:~$ docker exec -it uti_ros2_foxy /bin/bash

3. Set up ROS2 environment

root@nano-4gb-jp45:/# source /opt/ros/foxy/setup.bash

root@nano-4gb-jp45:/# source /home/jetbot/dev_ws/install/setup.bash

4. Send the command to control Jetbot

By publishing the array [Jetbot number, forward / backward distance (grain size: 3 cm), turning angle (grain size: 3 degrees)] in / openel_topic, the specified Jetbot will operate.

root@nano-4gb-jp45:/# ros2 topic pub --once /openel_topic std_msgs/msg/Int16MultiArray "{data: [1,3,3]}"

publisher: beginning loop

publishing #1: std_msgs.msg.Int16MultiArray(layout=std_msgs.msg.MultiArrayLayout(dim=[], data_offset=0), data=[1, 3, 3])

The log is displayed in the first Terminal, and the JetBot is working.

[INFO] [1642562229.934486853] [openel_subscriber]: I heard: 1,3,3

Move:3[cm]

Turn:3[degree]

令和3年度（2021年度）の活動成果

- 2021年7月2日に、OpenEL 3.2（C#版）をGitHubで公開した。
  https://github.com/openel/openel-cs
- 2021年10月30日にGitHubでOpenELの英語版ページを公開した。
  https://openel.github.io/
- 2021年11月7日にGitHubでArduino(M5Stack BALA2)用OpenEL 3.2（C++版）と英語版入門文書を公開した。
  https://github.com/openel/openel-arduino
  https://openel.github.io/openel-arduino/
- 2022年3月12日にGitHubでLEGO EV3およびNVIDIA JetBot用のOpenELコンポーネントを公開した。
  https://github.com/openel/openel
- 2022年3月12日にDockerHubでNVIDIA JetBot用のOpenELコンポーネントを使用するROS2イメージを公開した。
  https://hub.docker.com/r/openel/jetbot

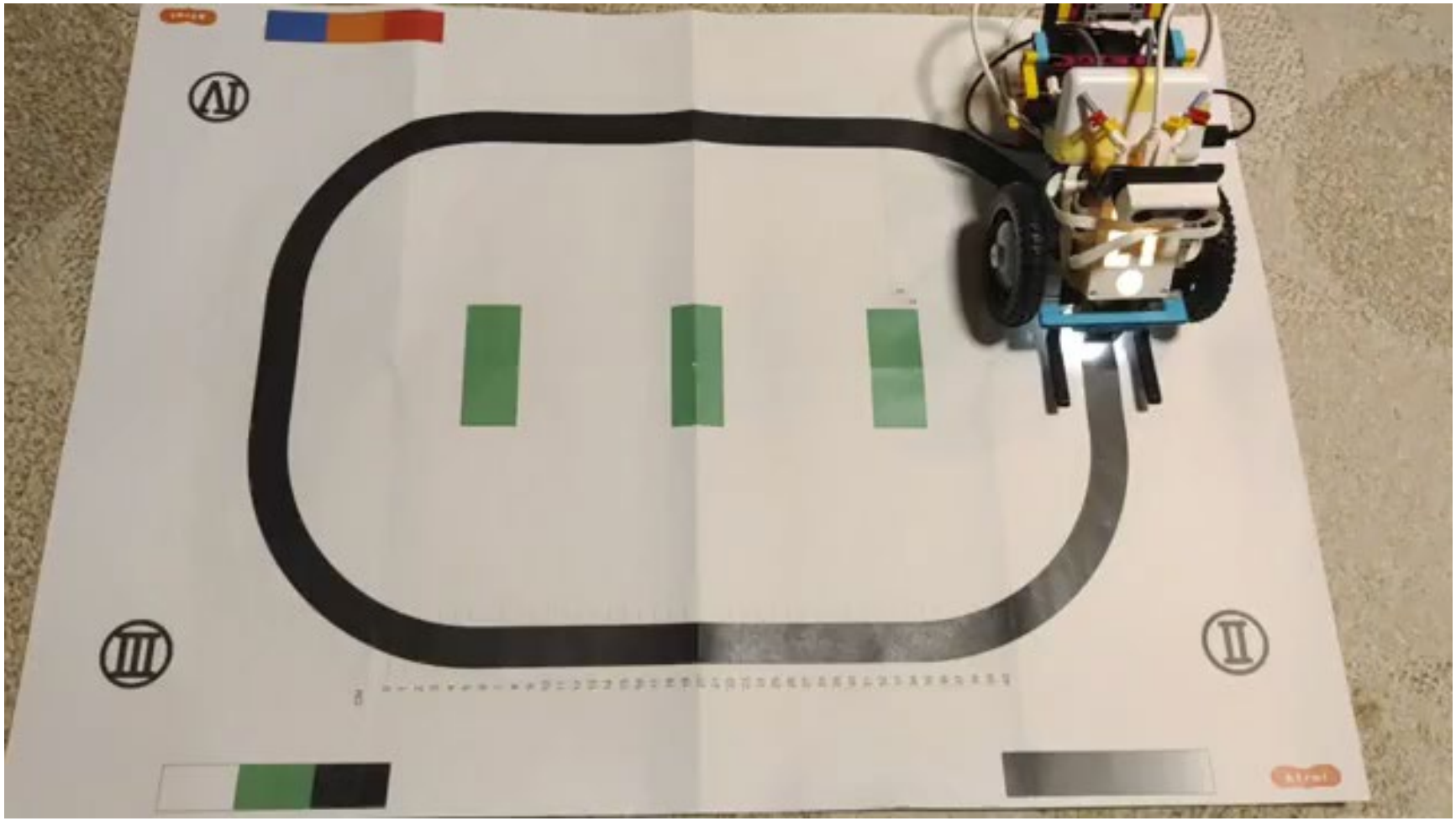
令和4年度の活動

ユーザーの増加、対応デバイスの増加、他システムとの連携、OpenELエコシステムの構築を目指して活動した。

- 対応デバイスの増加
  - ETロボコンSPIKEキットへの対応
  - シマフジ電機（株）SEMB1401への対応
- 他システムとの連携
  - W3C WoTへの対応（WoT-JP CGとの連携）
  - クラウドへの対応（Microsoft Azure、AWS、GCP）
  - 仮想シミュレーション環境への対応
- OpenELエコシステムの構築
- OpenEL活用WG主催セミナーの開催

ETロボコンSPIKEキットへの対応

ETロボコンSPIKEキットへの対応

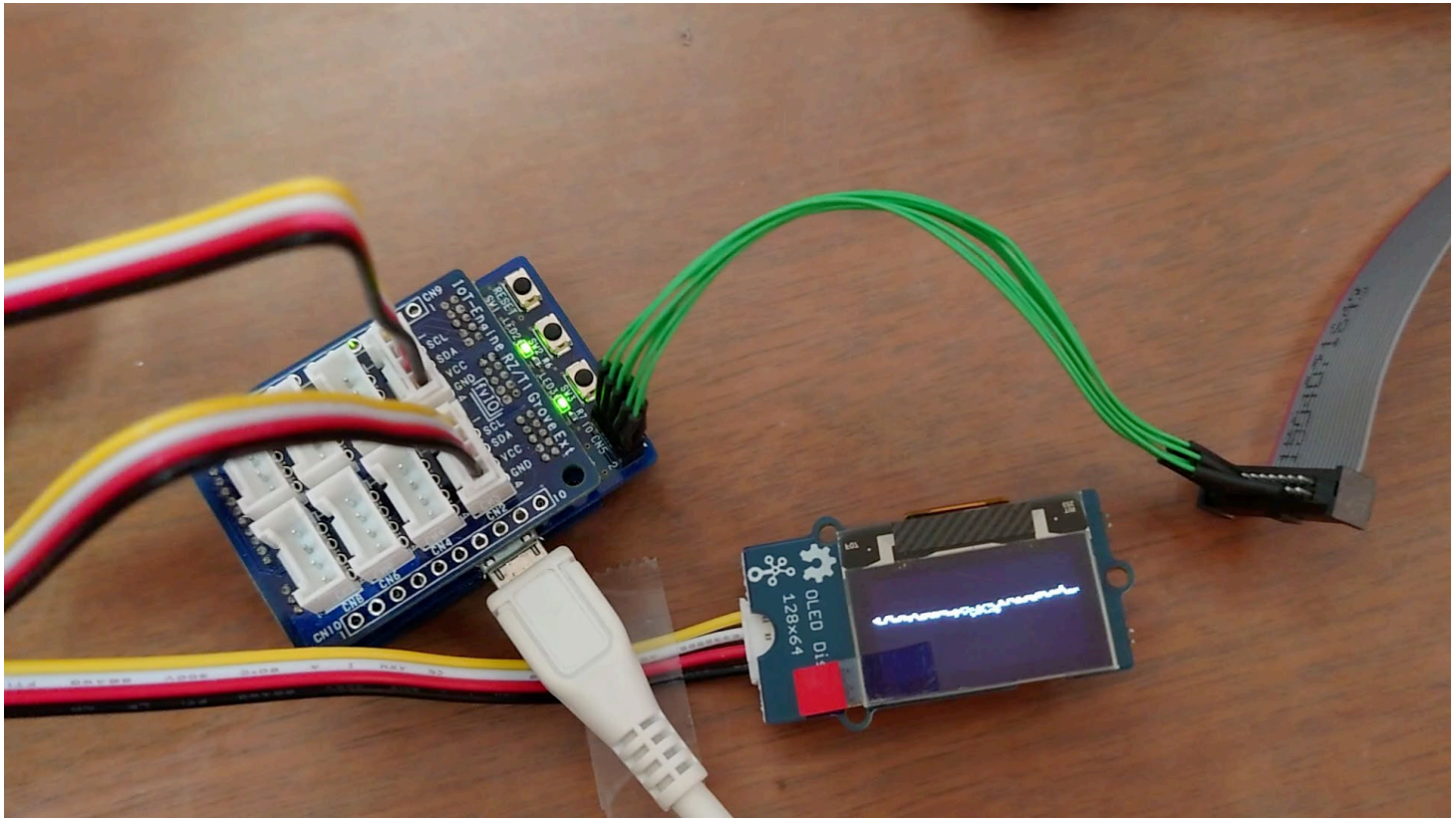


シマフジ電機（株）SEMB1401への対応

一般社団法人
組込みシステム技術協会
Japan Embedded Systems Technology Association

シマフジ電機（株）SEMB1401への対応

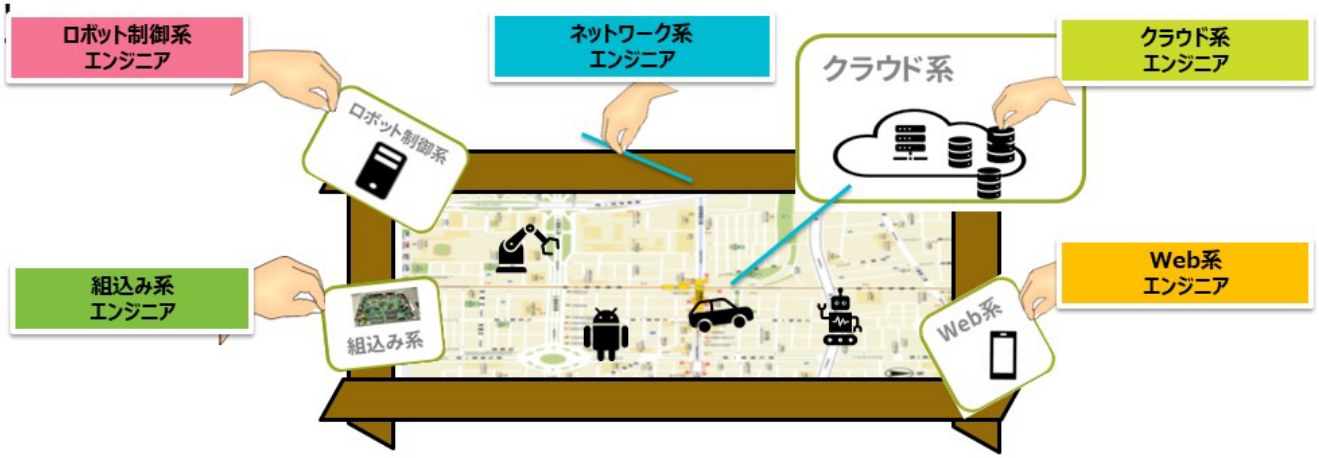
- I2C接続の加速度センサーADXL345の出力をOpenELで取得
- 今後、データをWi-Fiネットワークでゲートウェイに転送したり、クラウドに直接アップロードなどの拡張予定。



仮想シミュレーション環境「箱庭」への対応

『箱庭』の狙いとコンセプト

- 箱の中に，様々なモノをみんなの好みで配置して，いろいろ試せる！
  - ・仮想環境上(箱庭)でIoT/ロボット・システムを開発する
- ⇒ 各分野のソフトウェアを持ち寄って，机上で全体結合＆実証実験！



仮想シミュレーション環境「箱庭」への対応

- OpenELで箱庭が利用可能に！

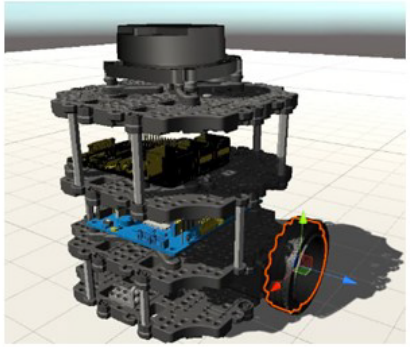
箱庭を実現する技術

TOPPERS
Toyohashi OPen Platform for Embedded Real-time Systems

仮想シミュレーション環境「箱庭」への対応

1個のロボットを動かす

- Unity上の1台のTurtlebot3をROS2で動かします



OpenELコンポーネントからROSを呼び出し

ロボット制御プログラム(C++)

OpenEL

ROS

Robot Operating System

unity

UI/ビジュアライズ/物理演算エンジン

箱庭

ROSトピック
(センサデータ等)

docker

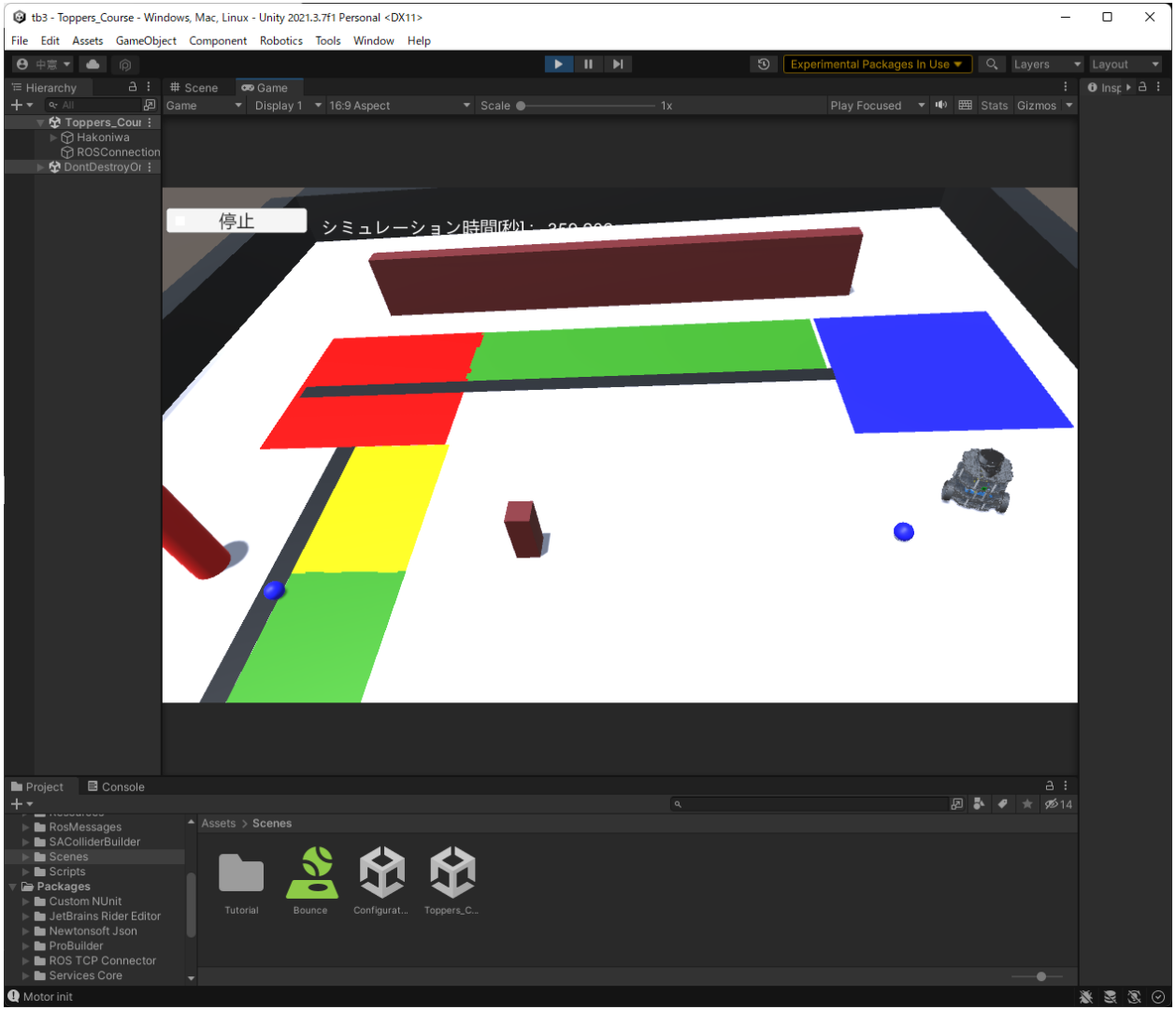
ubuntu®

TOPPERS

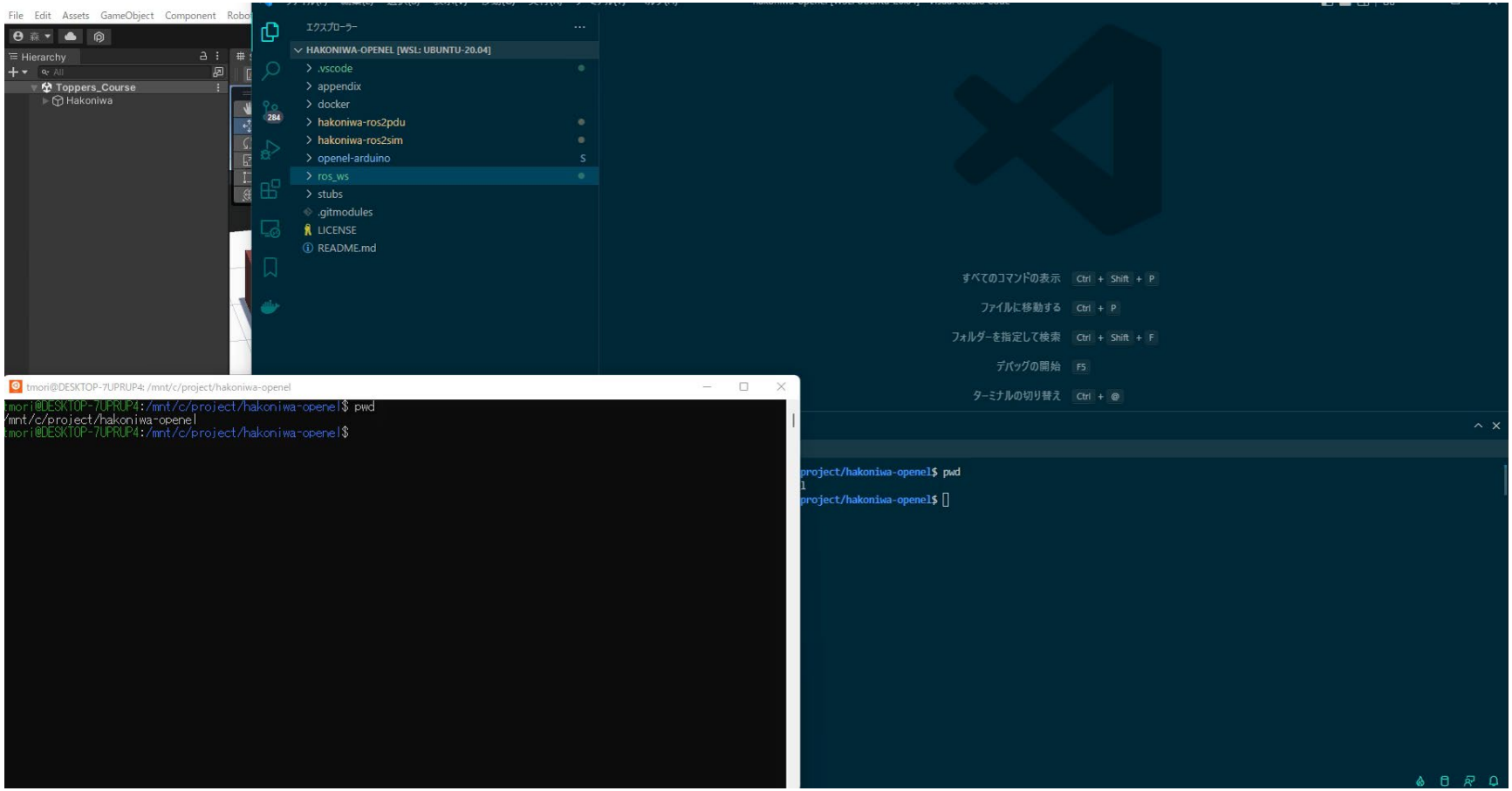
Toyohashi OPen Platform for Embedded Real-time Systems

シミュレーション環境「箱庭」でアプリケーションの動作を確認後、すぐに実機でも動作可能！

仮想シミュレーション環境「箱庭」への対応



仮想シミュレーション環境「箱庭」への対応



令和4年度（2022年度）の活動成果

■ OpenEL対応プラットフォームとデバイスの強化

- ETロボコンSPIKEキットにサンプル実装を行った。
- IoT-Engine RZ/T1（シマフジ電機（株） SEMB1401）に対応した。
- 仮想シミュレーション環境「箱庭」との連携を実現した。

■ OpenELの国内外における普及

- 2023年5月16日、OpenEL 3.2（C++版）をGitHubで公開した。
  https://github.com/openel/openel-cpp
- 2023年5月19日、OpenEL 3.2(C++版)に対応した箱庭を公開した。
  https://github.com/toppers/hakoniwa-openel-cpp

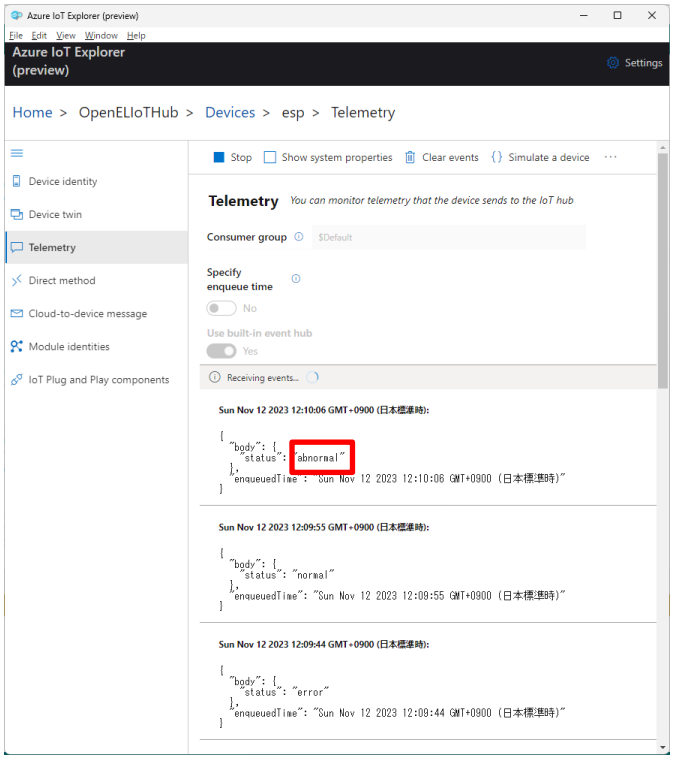
令和5年度の活動

ユーザーの増加、対応デバイスの増加、他システムとの連携、OpenELエコシステムの構築を目指して活動した。

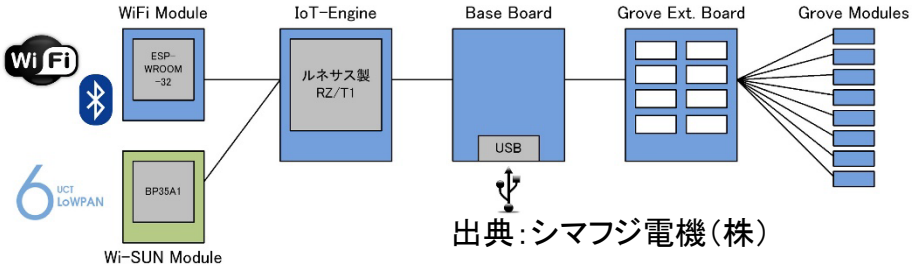
- ■ 対応デバイスの増加
  - IoT-Engine RZ/T1（シマフジ電機（株） SEMB1401）のWi-Fi対応
- ■ 他システムとの連携
  - クラウドへの対応（Microsoft Azure）
- ■ OpenELエコシステムの構築
- ■ OpenEL活用WG主催セミナーの開催

IoT-Engine + OpenEL

- IoT-Engine RZ/T1(シマフジ電機((株))SEMB1401)
- OpenELのAPIを使用してI2C接続された加速度センサーの値を取得し、リアルタイムでLCDに表示
- SEMB1401で異常振動を検知した場合、UARTでWiFi Moduleに通知
- WiFi Moduleがクラウド(Azure IoT Hub)へ通知



異常振動を検知したら、クラウドに通知

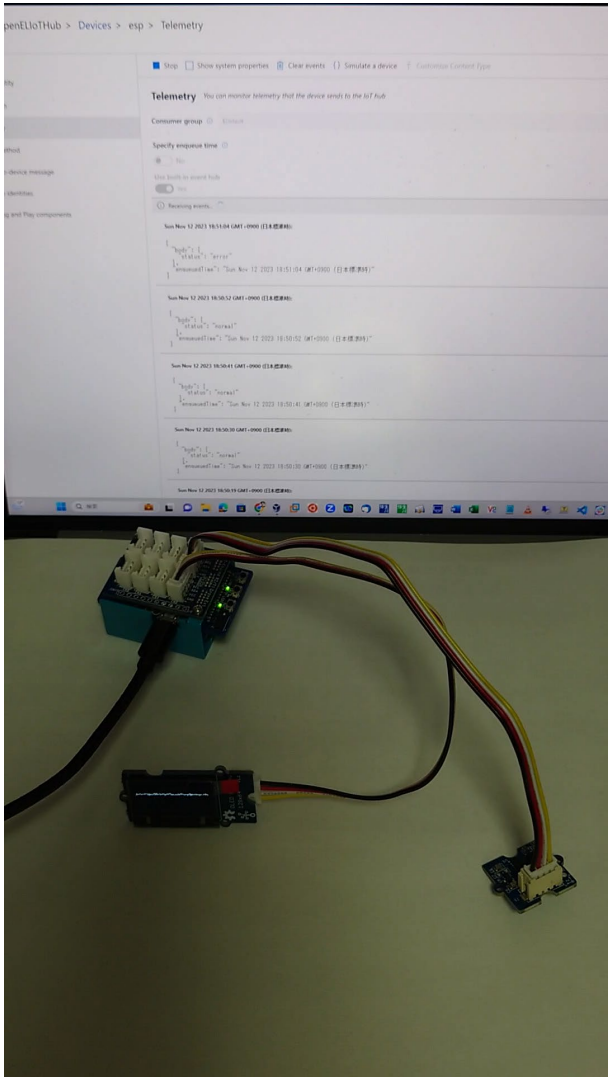


出典:シマフジ電機(株)

一般社団法人 組込みシステム技術協会 Japan Embedded Systems Technology Association

© Japan Embedded Systems Technology Association 2023

デモ動画



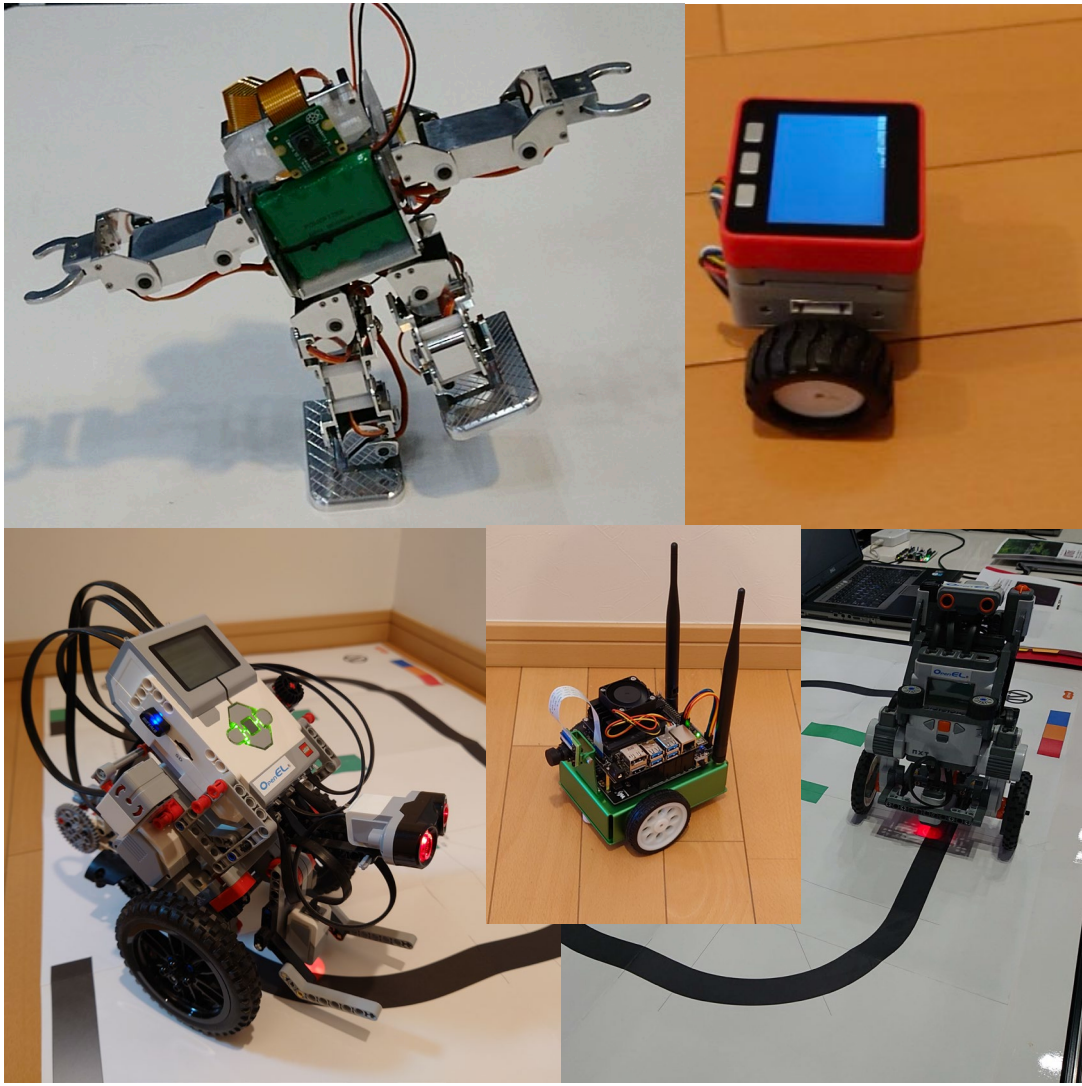
令和5年度（2023年度）の活動成果

- WG開催：11回
- セミナー開催：2回
- EdgeTech+ West出展、講演
- EdgeTech+ 出展、講演
- OpenEL対応プラットフォームとデバイスの強化
  - IoT-Engine RZ/T1（シマフジ電機（株） SEMB1401）のWi-Fi対応
    - クラウドへの対応（Microsoft Azure IoT Hub）
- 実製品でのOpenELの活用
- 国際会議での論文の引用
  - 国際会議APLIS 2023に採択されたIoT DEPの論文内でOpenELの論文[1]が参照。
  - [1]Kenichi Nakamura, “OpenEL – A Next Generation Platform for Robot Development from Japan to International Standard –,” IEICE Technical Report, CNR2013-11. 2013
- 国際標準への提案準備
  - ISO/IEC 30161 Internet of Things (IoT)の調査

2024年11月現在の対応デバイスと実装例

■ 対応デバイス
- モーター（速度制御、位置制御、トルク制御）
- ジャイロセンサー
- トルクセンサー
- 加速度センサー
- 地磁気センサー
- 距離センサー
- 力センサー
- 温度センサー
- 湿度センサー
- 気圧センサー
- 二酸化炭素センサー
- カラーセンサー
- タッチセンサー
- 電流センサー

■ 実装例
- 二足歩行ロボット UTRX-17
- M5Stack Fire/BALA/BALA2
- LEGO Mindstorms NXT/EV3
- NVIDIA JetBot
- LEGO SPIKE
- IoT-Engine RZ/T1 SEMB1401
- 箱庭



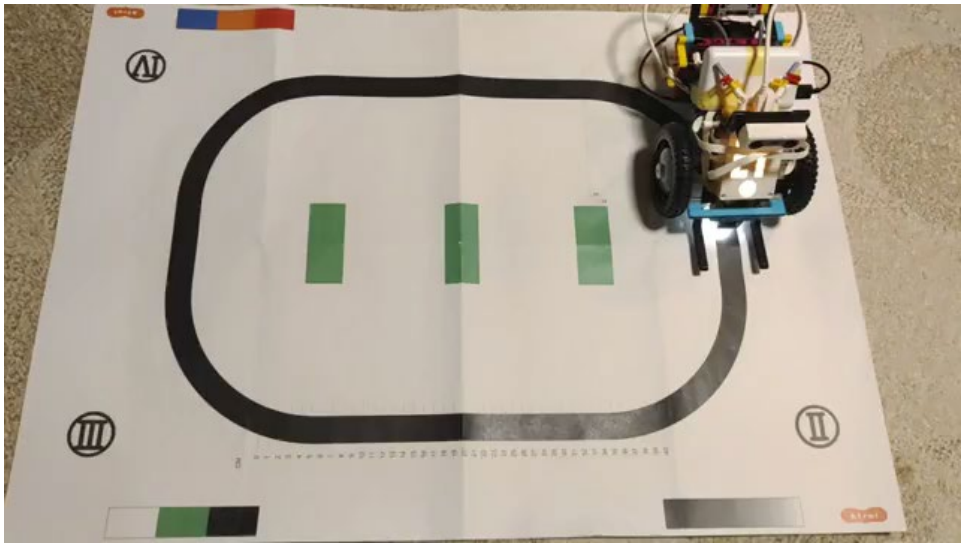
令和6年度の活動

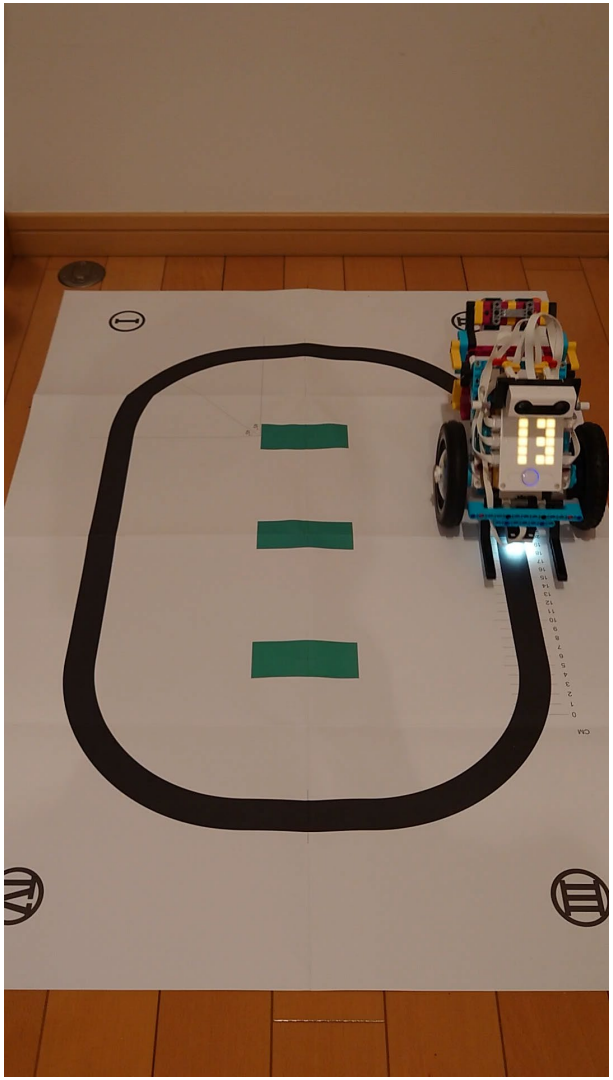
ユーザーの増加、対応デバイスの増加、他システムとの連携、OpenELエコシステムの構築を目指して活動している。

1. OpenELの国内外における普及
   - OpenELの国内外における普及のため、OpenEL対応プラットフォームとデバイスを強化する。
   - ETロボコンSPIKEキットへの対応
2. OpenELの仕様の強化
   - 国際標準への提案にあわせて仕様を強化する。
3. OpenELの国際標準への提案
   - ISO/IEC JTC1/SC41への提案
   - 経済産業省が国際標準化を進めているISO/IEC 30161 Internet of Things(IoT) - Data Exchange Platform for IoT servicesのPart 3: Architecture and interface for remote control systemsのInterface仕様としてOpenELを提案する。

LEGO Education SPIKE Primeへの対応

- 令和4年度（左の動画）は、ETロボコン走行体（HackSPi）のRaspberry Pi4（OS:Linux）に対応
- 令和6年度（右の動画）は、ETロボコン走行体（HackSPi）のPrime Hub（OS:spike-rt）とPUP(Power Up)デバイス（モータ、センサ）に対応
- OSが異なっても、OpenEL APIを使用するとほぼ同じライントレースプログラムが動作





ライントレースプログラムの例

```
#include "openEL.h"
/* HALコンポーネント */
HALCOMPONENT_T *halMotor01;
HALCOMPONENT_T *halMotor02;
HALCOMPONENT_T *halMotor03;
HALCOMPONENT_T *halSensor01;
/* グローバル変数 */
HALFLOAT_T velocityLeft=0, velocityRight=0;
HALFLOAT_T target_value=30;
HALFLOAT_T target_velocity=25;
HALFLOAT_T colorsensor_value=0;
int32_t size=0;
HALFLOAT_T Kp=0.25, Kd=0.1;
HALFLOAT_T diff[2]={0};
/* ライントレースタスク(周期タスク) */
void LineTrace(intptr_t exinf)
{
        HalSensorGetValueList(halSensor01, &size, &colorsensor_value); /* 光センサの値を取得 */
        diff[0]=diff[1];
        diff[1]=target_value - colorsensor_value; /* 目標値との差分 */
        /* PD制御 */
        velocityLeft   = target_velocity - Kp * (target_value - colorsensor_value) + Kd * (diff[1] - diff[0]) / 0.1;
        velocityRight  = target_velocity + Kp * (target_value - colorsensor_value) + Kd * (diff[1] - diff[0]) / 0.1;
        HalActuatorSetValue(halMotor01, HAL_REQUEST_VELOCITY_CONTROL, velocityLeft);  /* 左モータに指令 */
        HalActuatorSetValue(halMotor02, HAL_REQUEST_VELOCITY_CONTROL, velocityRight); /* 右モータに指令 */
}
```

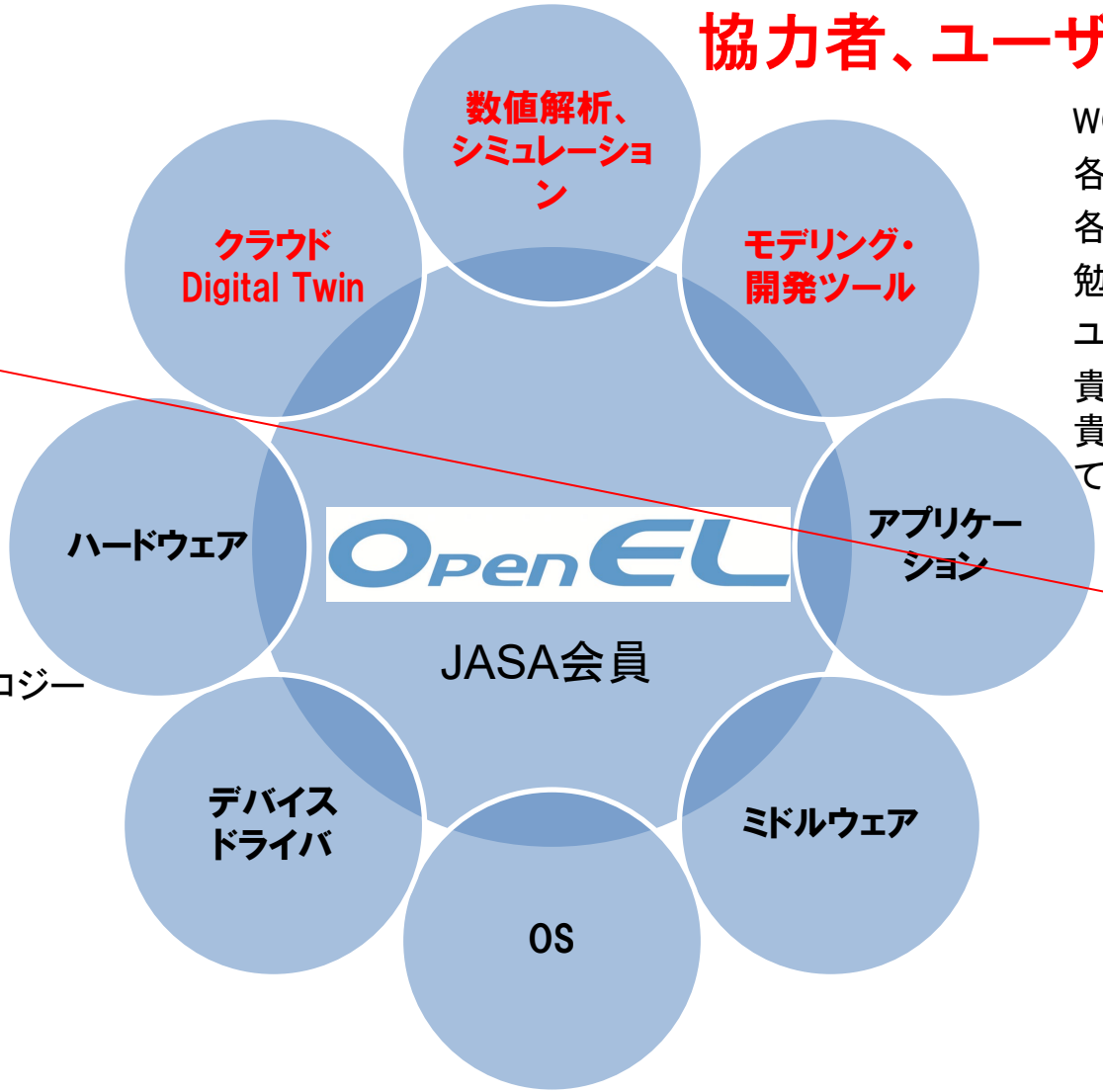
ライントレースプログラムの例

```
void Main(intptr_t exinf)
{
    /* HALコンポーネントの作成 */
    halMotor01  = HalCreate(0x00000001, 0x00000009, 0x00000002, 1); // LEGO-SPIKE-L-Motor Left Wheel
    halMotor02  = HalCreate(0x00000001, 0x00000009, 0x00000002, 2); // LEGO-SPIKE-L-Motor Right Wheel
    halMotor03  = HalCreate(0x00000001, 0x00000009, 0x00000002, 3); // LEGO-SPIKE-M-Motor
    halSensor01 = HalCreate(0x0000000B, 0x00000009, 0x00000002, 1); // LEGO-SPIKE-Color
    /* HALコンポーネントの初期化 */
    HalInit(halMotor01);
    HalInit(halMotor02);
    HalInit(halMotor03);
    HalInit(halSensor01);
    sta_cyc(TRACE_CYC); /* 周期タスクを起動 */

    while(1) {
        slp_tsk(); /* 自タスクをSUSPEND状態に遷移 */
    }
    /* HALコンポーネントの終了 */
    HalFinalize(halMotor01);
    HalFinalize(halMotor02);
    HalFinalize(halMotor03);
    HalFinalize(halSensor01);
    /* HALコンポーネントの削除 */
    HalDestroy(halMotor01);
    HalDestroy(halMotor02);
    HalDestroy(halMotor03);
    HalDestroy(halSensor01);
    exit(0);
}
```

OpenEL エコシステム

協力者、ユーザー募集！



WGメンバー
各種コンポーネントの実装
各種ツールの対応
勉強会講師
ユーザーサポート
貴社の顧客への提案等、貴社のビジネスツールとしてもご活用ください。

WGメンバー
アップウィンドテクノロジー
アフレル
エヌデーデー
京セラ
シマフジ電機
チェンジビジョン
金沢工業大学
静岡大学
東京大学
東洋大学
Knowledge & Experience
日立産機システム
UCサロン

教育
大学
高専
専門学校
工業高校

一般社団法人 組込みシステム技術協会
Japan Embedded Systems Technology Association

2024年度第1回OpenEL活用Webセミナー

https://www.jasa.or.jp/lists/event_seminar/openelweb11-29/

日時

2024年11月29日（金）15:00-16:00

講演タイトル

スマートホーム機器の新たな共通規格であるMatterとは

講演者

ディジインターナショナル株式会社　代表取締役社長　江川将峰様

テュフラインランドジャパン株式会社　シニアエキスパートエンジニア　齋藤修治様

講演概要

欧米を中心にスマートホームの普及が進んでいるもののメーカー毎にバラバラな通信規格は、普及の阻害要因となっていた。その問題を解決するために誕生したのが「Matter」である。

Matterとは、CSA（Connectivity Standard Alliance）が推進する業界統一のオープンソース接続規格で、メーカーを問わず、プラットフォームを超えてIoT機器間のシームレスな通信を可能にするものである。

このMatterのプロトコルによりプラットフォームの枠組みを超え、より便利にIoT機器の操作ができるようになる。

本講演では、スマートホーム、CSA（Connectivity Standard Alliance）、Matter、Matter対応製品、認証プロセスなどについて紹介する。

一般社団法人
組込みシステム技術協会
Japan Embedded Systems Technology Association

協力者、ユーザー募集！

■　ご静聴ありがとうございました。

「OpenELが変える組込みシステム開発」

2024/11/22 発行

発行者　一般社団法人 組込みシステム技術協会
東京都 中央区 入船 1-5-11 弘報ビル5階
TEL: 03(6372)0211 FAX: 03(6372)0212
URL: https://www.jasa.or.jp/

本書の著作権は一般社団法人組込みシステム技術協会(以下、JASA) が有します。
JASAの許可無く、本書の複製、再配布、譲渡、展示はできません。
また本書の改変、翻案、翻訳の権利はJASAが占有します。
その他、JASAが定めた著作権規程に準じます。

一般社団法人
組込みシステム技術協会
Japan Embedded Systems Technology Association

© Japan Embedded Systems Technology Association 2024