



OpenEL

OpenELが変える組込みシステム開発

対応デバイスのさらなる増加、他システムとの連携を実現！

2022年7月29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



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年度の事業計画
- 協力者募集



組込みシステム開発における課題

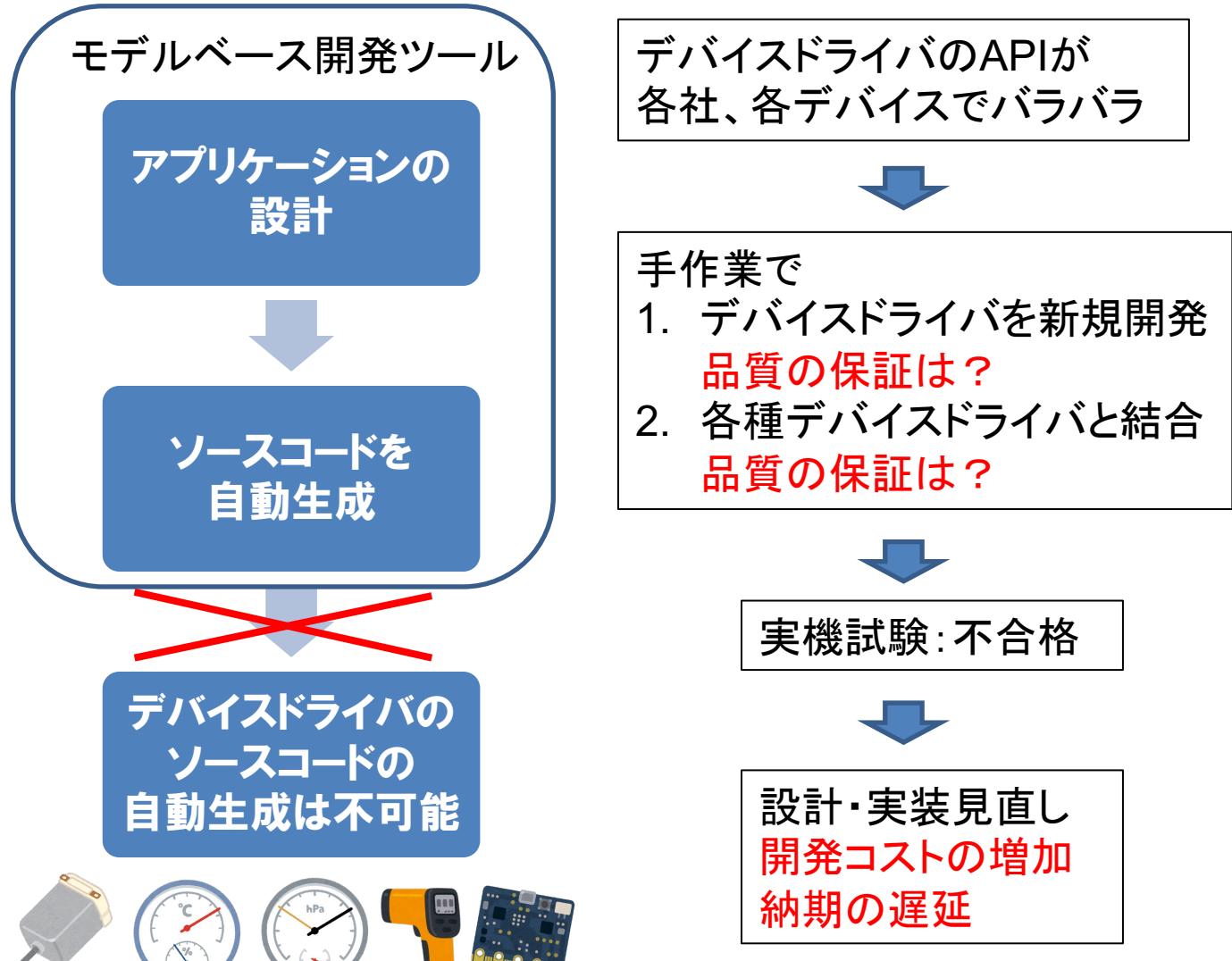
組込みシステム開発における課題



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



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(5): アップウインドテクノロジー、エヌデーター、チェンジビジョン、東洋大学、UCサロン
- 非会員(6): 京セラ、静岡大学、シマフジ電機、Knowledge & Experience、日立産機システム、ルネサスエレクトロニクス

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

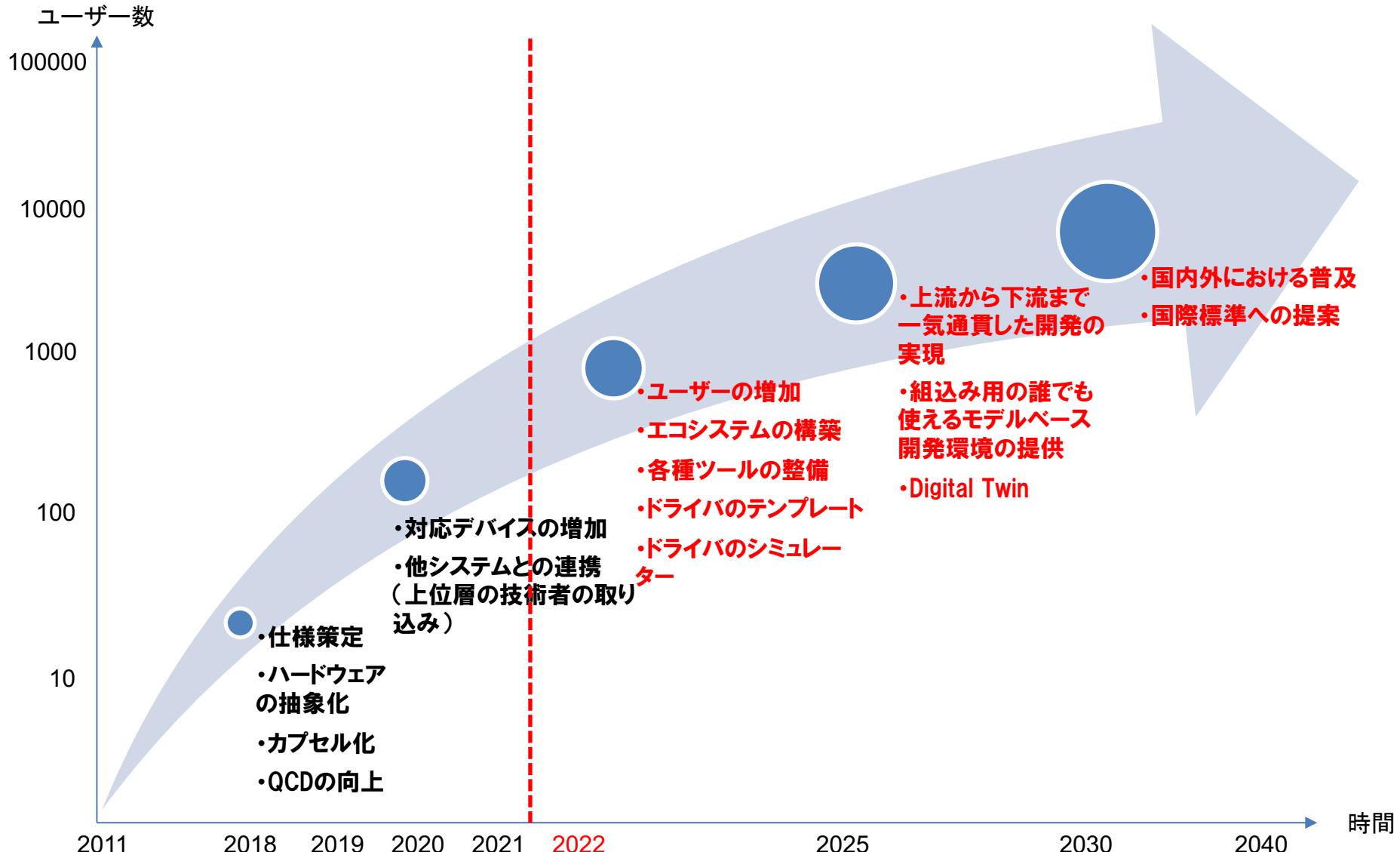
■ 背景

- 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



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？

分析

数値解析、
シミュレーション

シミュレーターで実機レス検証が可能に

設計

モデリング

完全なソースコードの自動生成が可能に

アプリ実装

アプリケーション、
ミドルウェア、OS

APIを統一

デバドラ実装

デバイスドライバ

OpenEL

OpenELで
上流から下流まで
一気通貫した開発が
可能に！

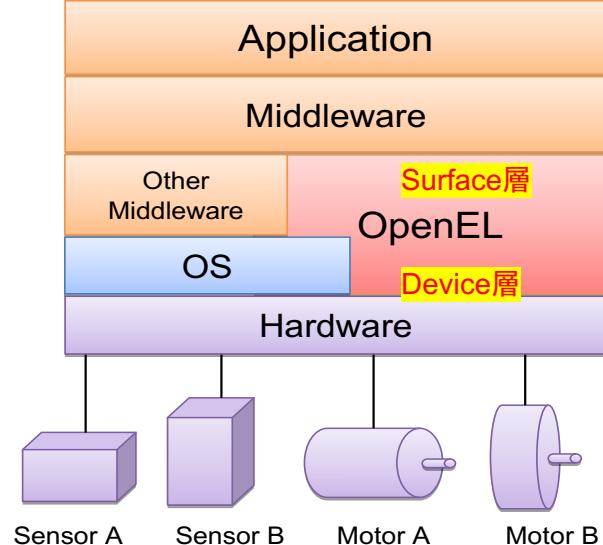


一般社団法人
組込みシステム技術協会
Japan Embedded Systems Technology Association

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®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>

The screenshot shows the GitHub repository page for 'openel/openel'. The repository has 8 commits, 2 branches, and 2 tags. The commit history includes:

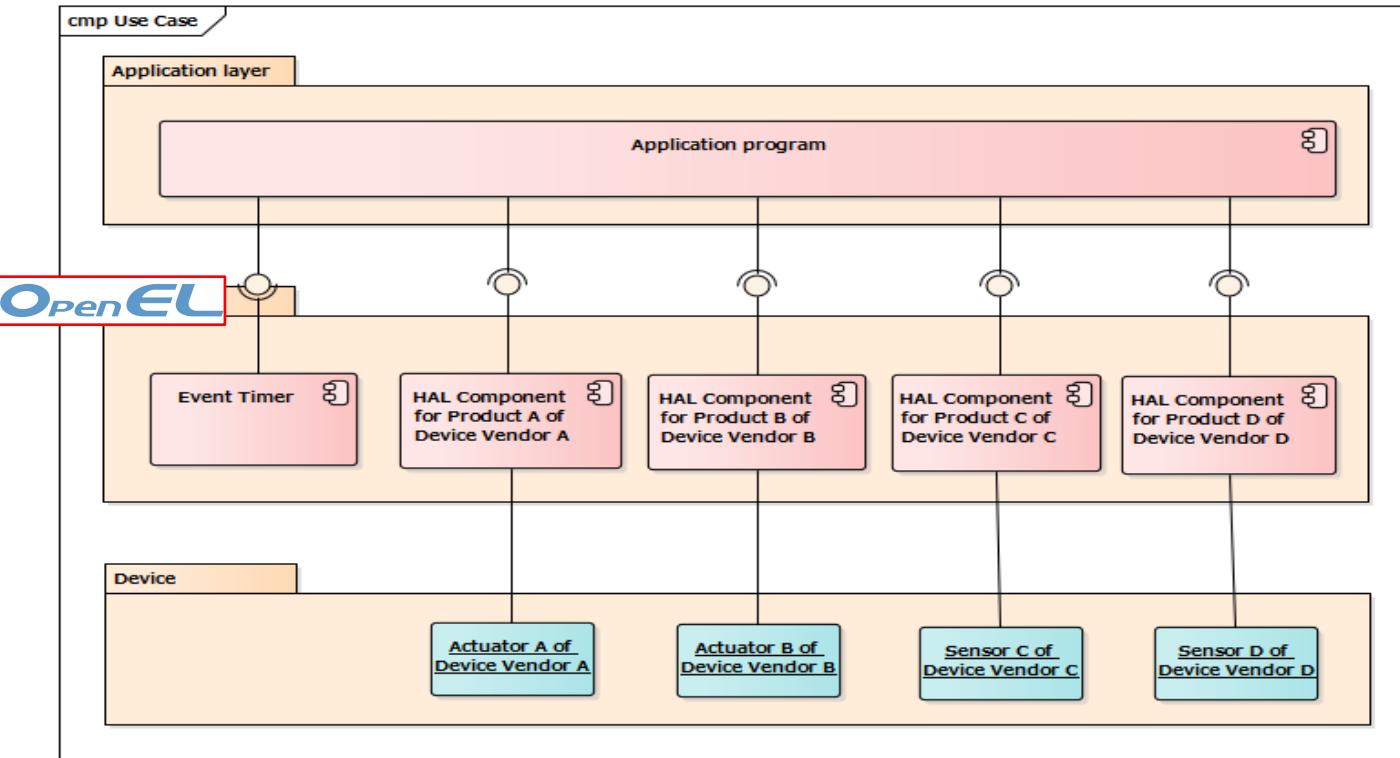
- ken-hakamura Merge pull request #1 from openel/develop ... (5881446, 2 Jul 2021)
- documentation Added OpenEL Specification 3.2 Japanese edition (7 months ago)
- include Modified type (7 months ago)
- lib Corrected some mistakes (4 years ago)
- sample Initial commit (4 years ago)
- LICENSE Initial commit (4 years ago)
- README.md Initial commit (4 years ago)

The repository has 2 stars, 1 watching, and 2 forks. It includes tags for 'openel.github.io', 'embedded', 'sensor', 'hal', 'actuator', 'embedded-hal', and 'hardware-abstraction-layer'. The 'About' section notes that OpenEL is a unified API(Application Programming Interface) for actuators and sensors, developed by JASA(Japan Embedded Systems Technology Association) since 2011.

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、
OpenEL 3.2を開発

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、
OpenELの拡張開発

OpenEL® 3.2 仕様



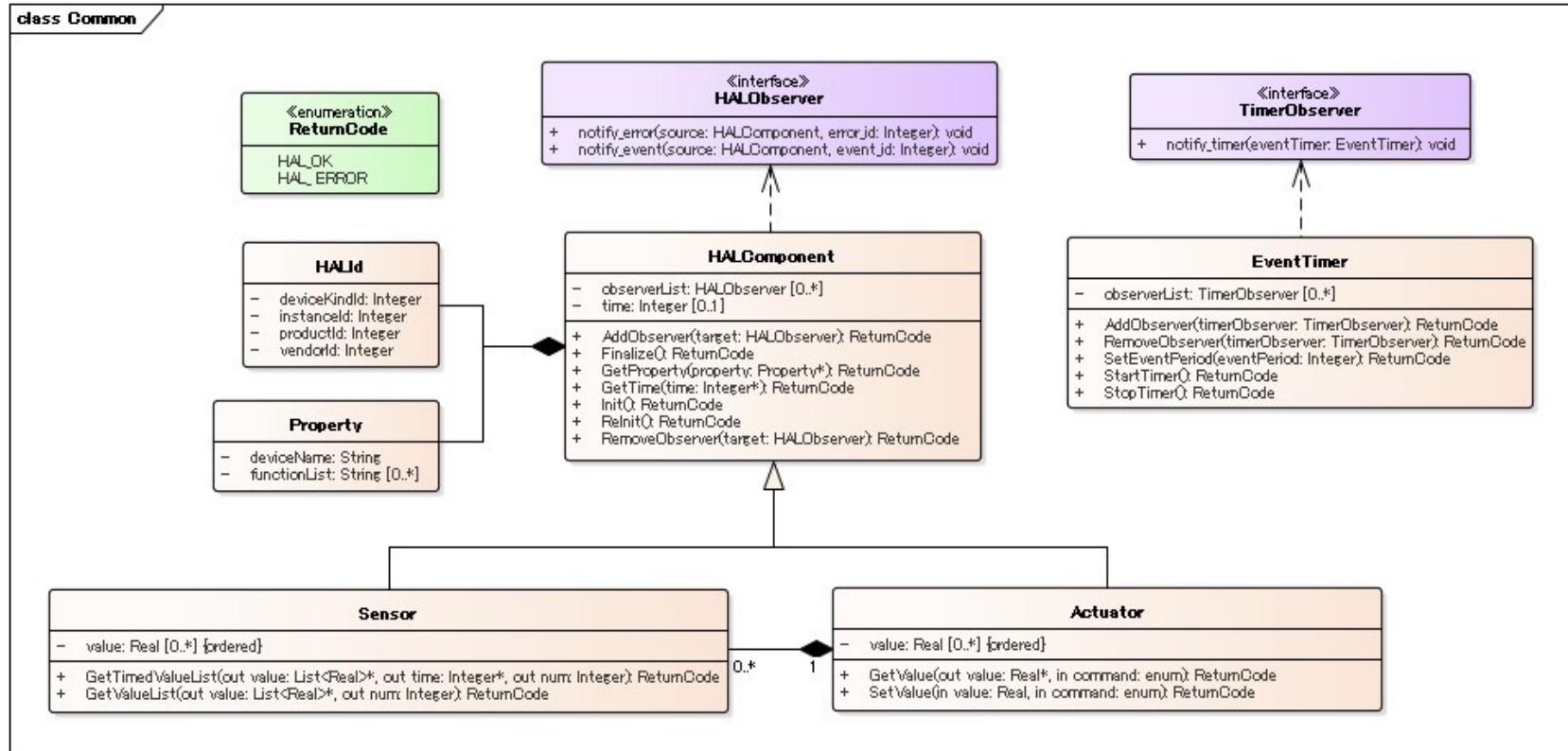
Surface層

Device層

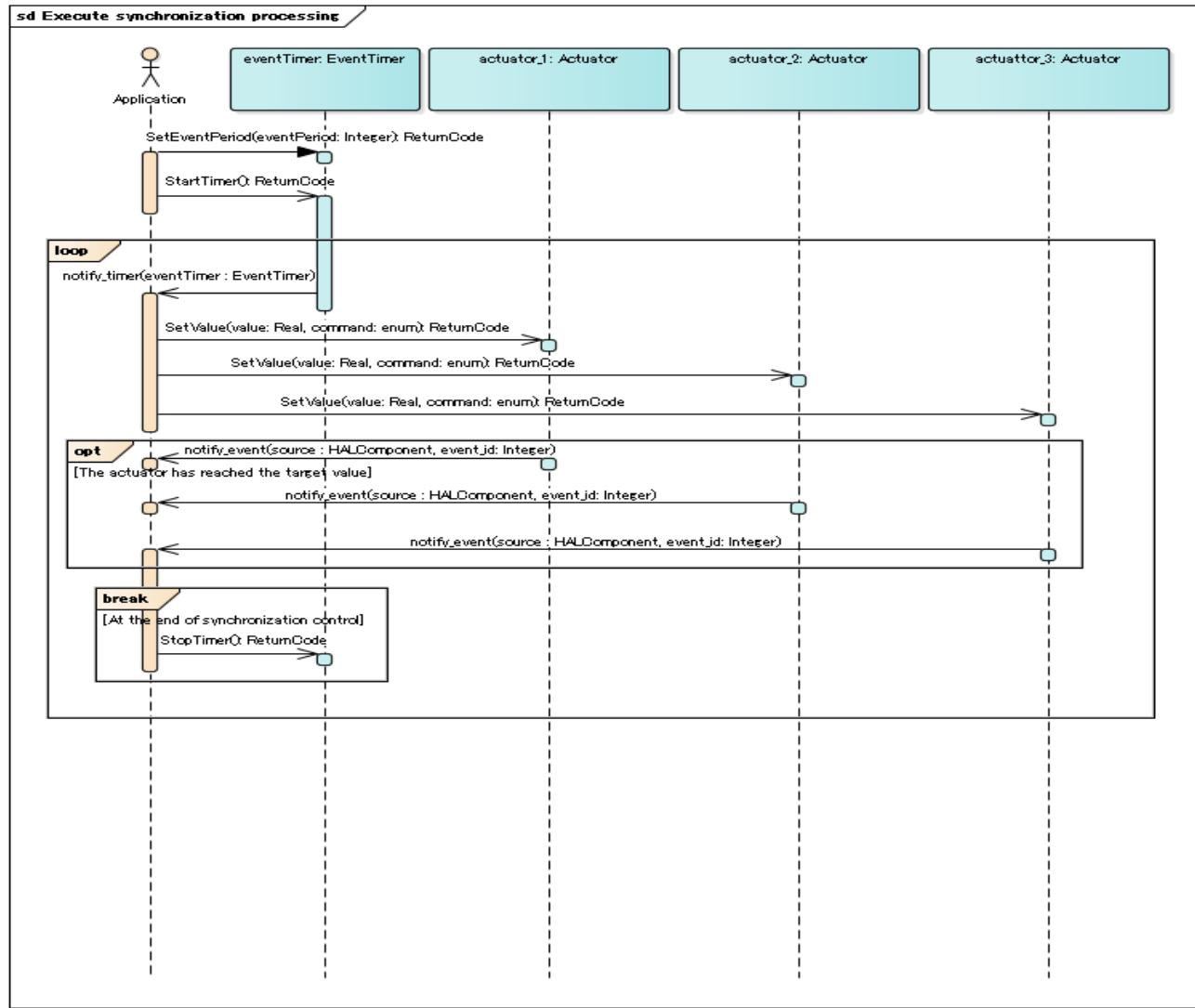


一般社団法人
組込みシステム技術協会
Japan Embedded Systems Technology Association

OpenEL® 3.2 仕様



OpenEL® 3.2 仕様





■ 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)



■ 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

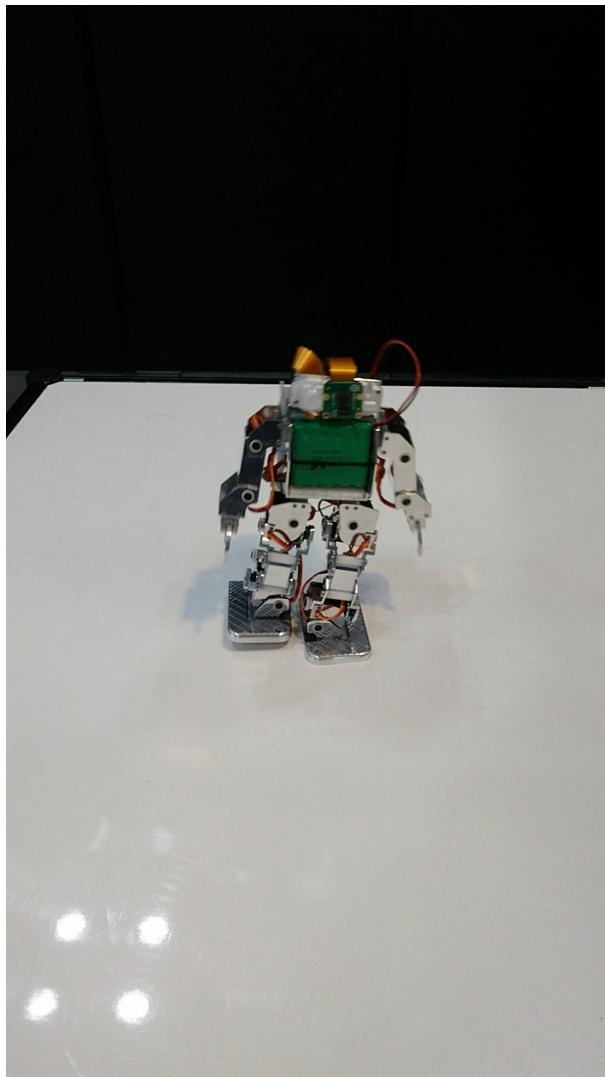


OPENELの実装例



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)

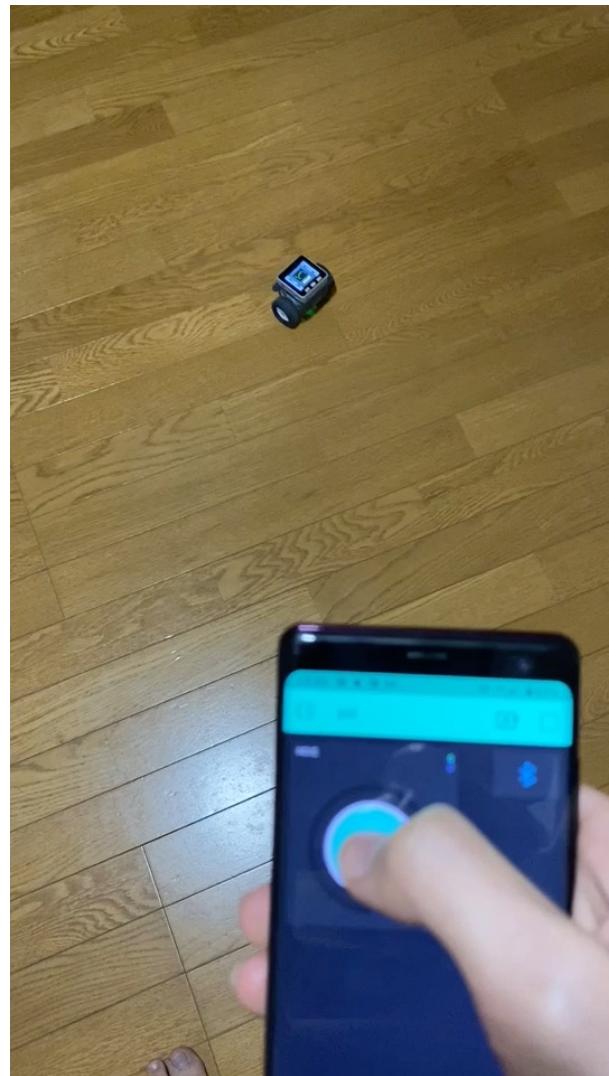
- M5Stack FIRE + M5BALA
- M5Stack FIREにSH200Q (3軸加速度センサー、3軸ジャイロ)が搭載
- OpenELで倒立2輪ロボットを制御
- Arduino IDEで開発





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

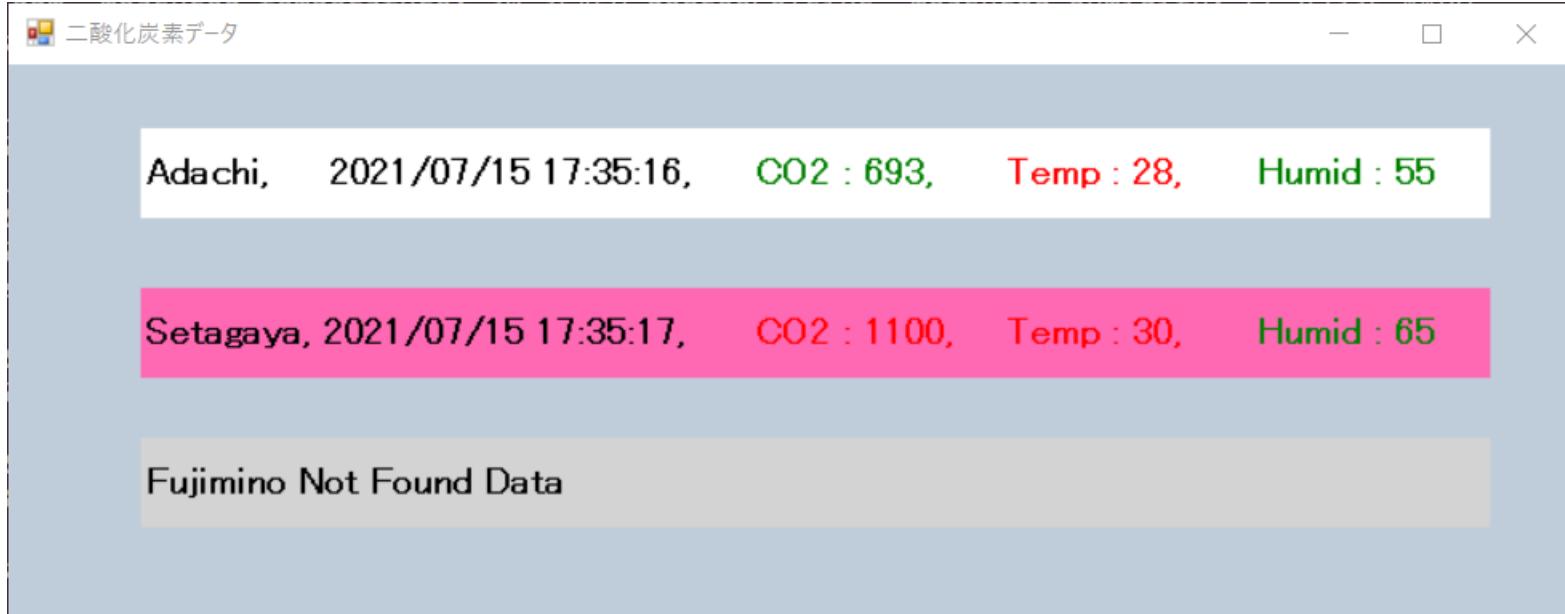
- 二酸化炭素センサー(
Sensirion SCD30)に対
応したことにより、換気
具合の可視化「密の見
える化」を実現
- C#に対応したことにより
、OpenELコンポーネント
内部でネットワーク通信
の利用を実現
- 大規模リアルタイム通信
エンジン「Diarkis」に対
応したことにより、遠隔
地のCO2濃度の測定を
実現
- 拠点間で相互にCO2濃
度のリアルタイム監視を
実現
- ET&IoT West 2021と
ET&IoT 2021のJASAパ
ビリオンで展示



C#への対応: Windows GUIアプリの例

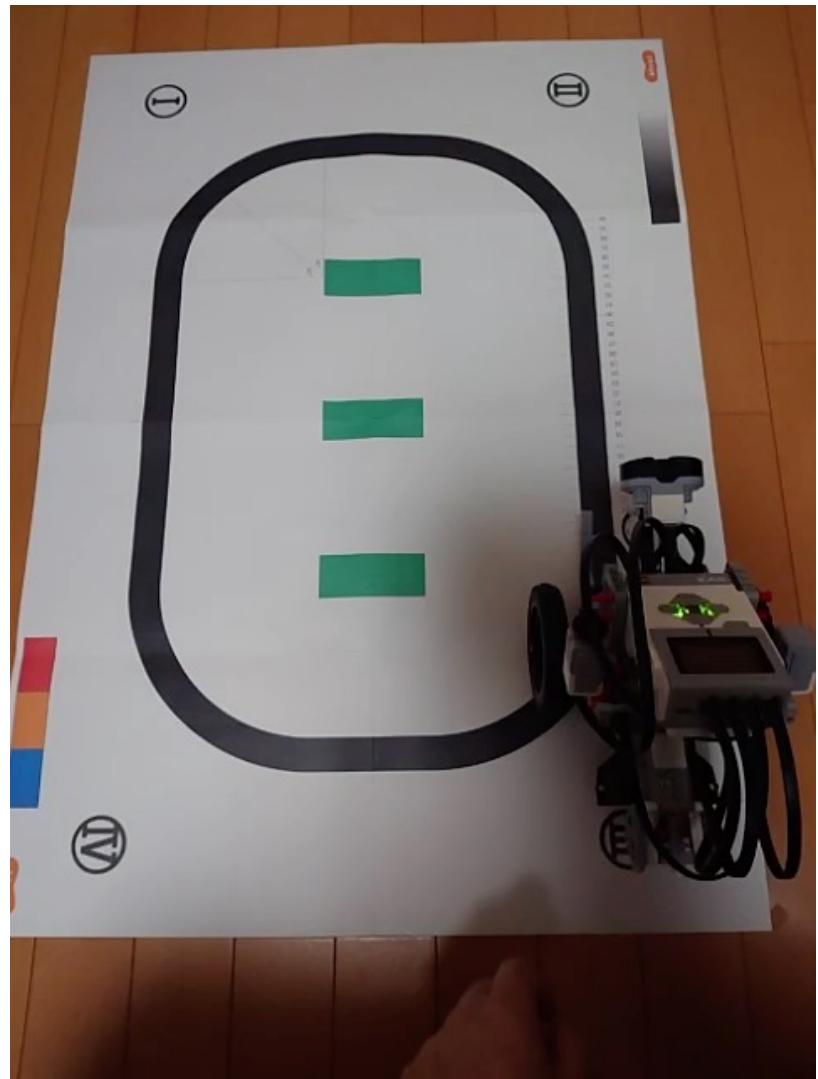


C#への対応によりWindowsのGUIアプリでもOpenELの利用が可能に！



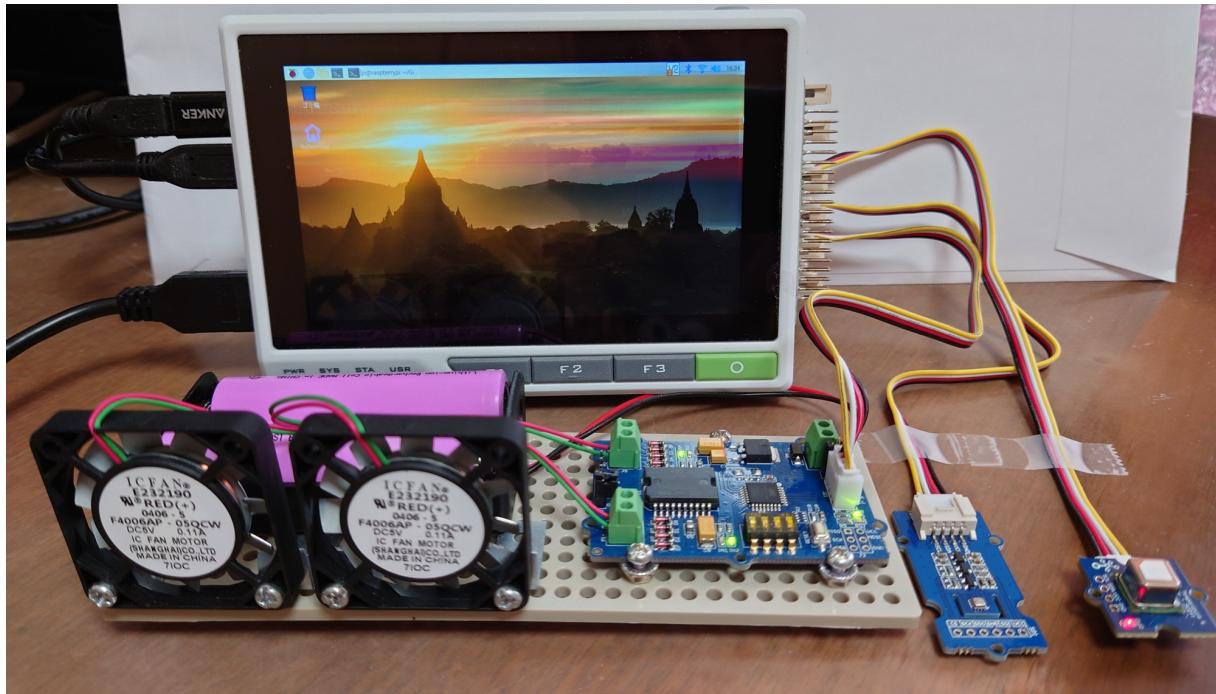


- ev3dev環境(Debian Linuxベース)
- モーター3個、光センサー、タッチセンサー、距離センサー、ジャイロセンサーに対応
- ETロボコン走行体で
ライントレースを実現

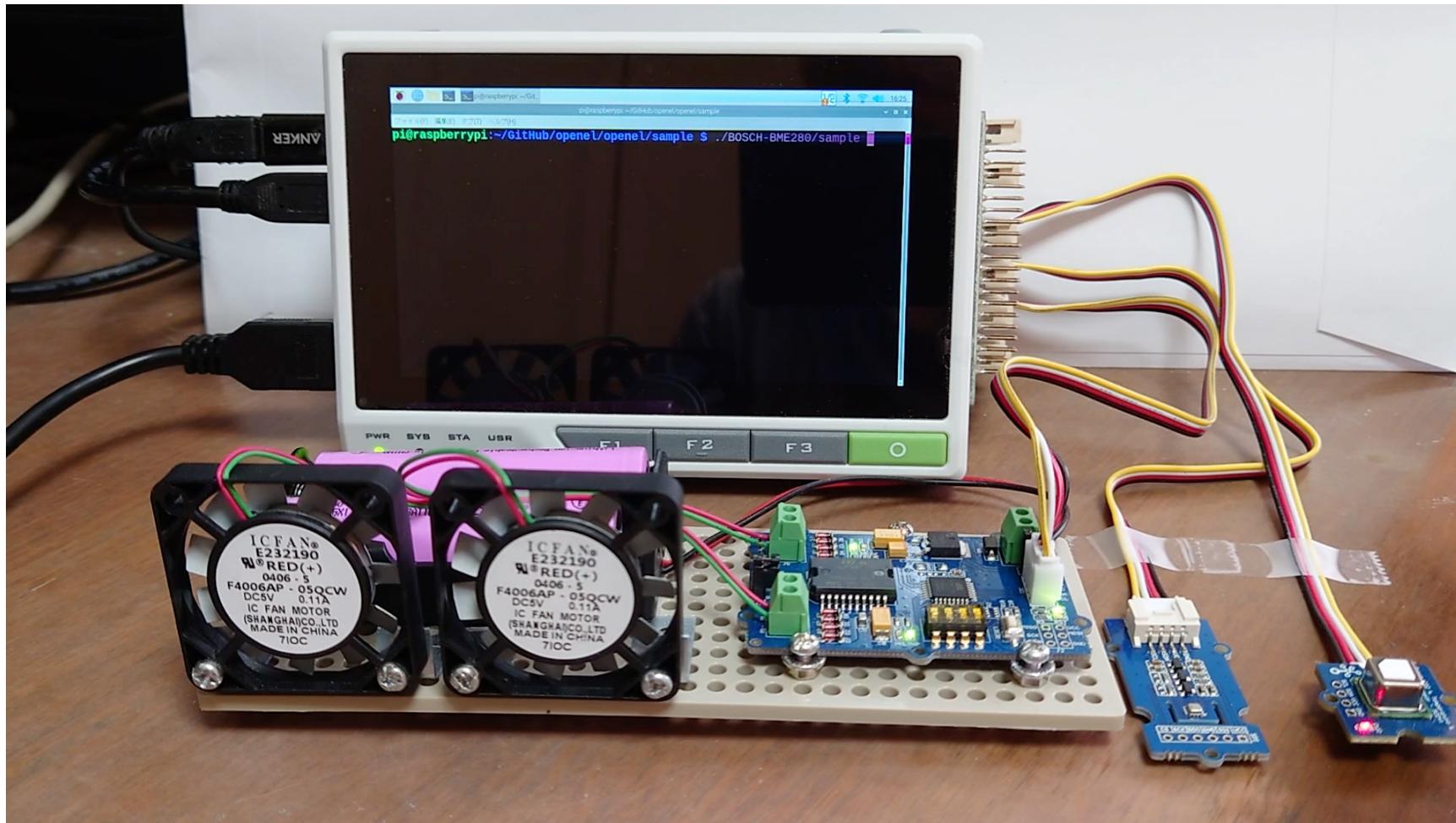




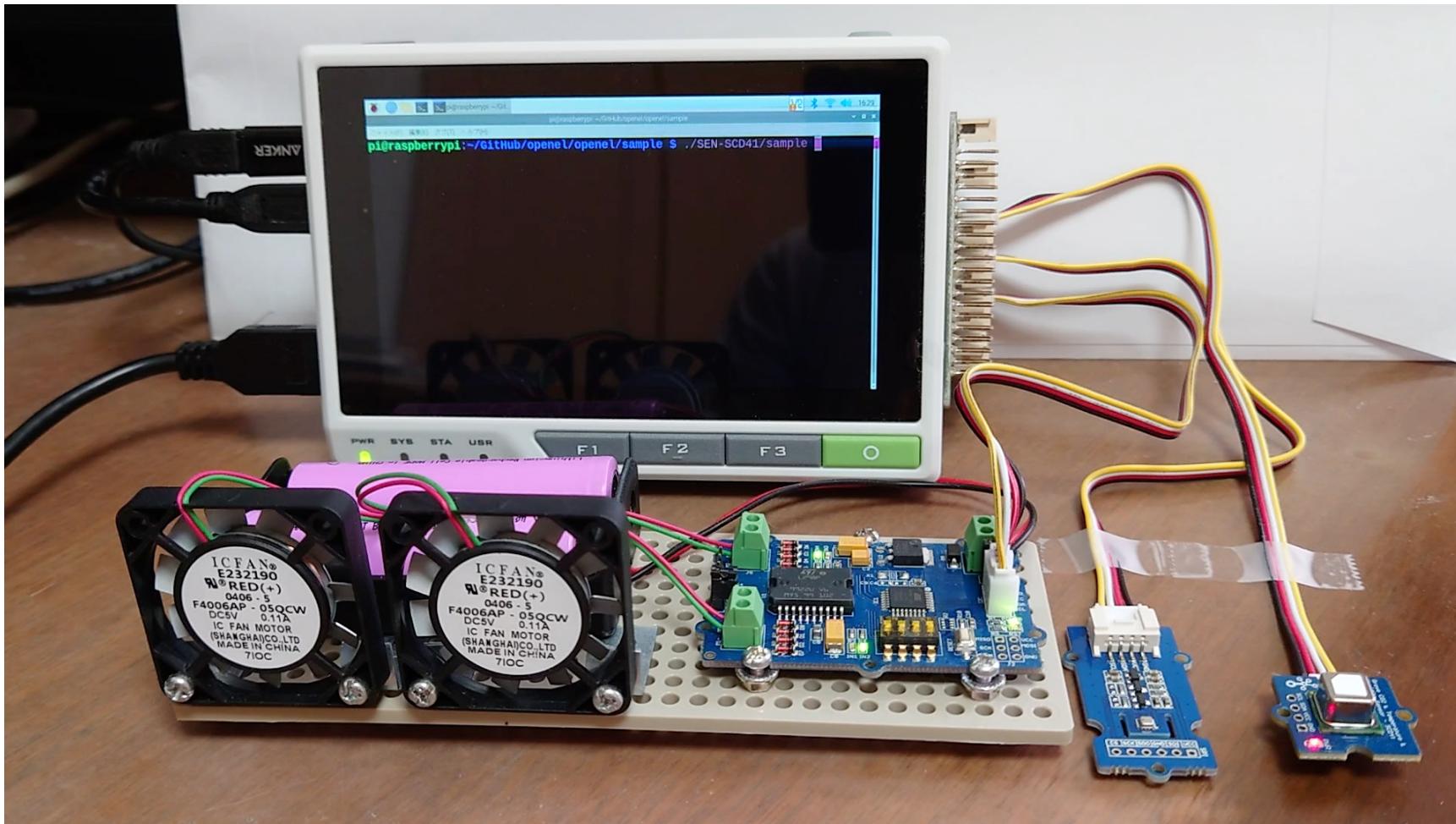
- Raspberry Pi CM4搭載
- 温度/湿度/気圧センサー BOSCH BME280
- CO2/温度/湿度センサー Sensirion SCD41
- I2Cモータードライバ+DCファンモーター
- 二酸化炭素濃度を検知して自動的に換気を行うシステムを構築



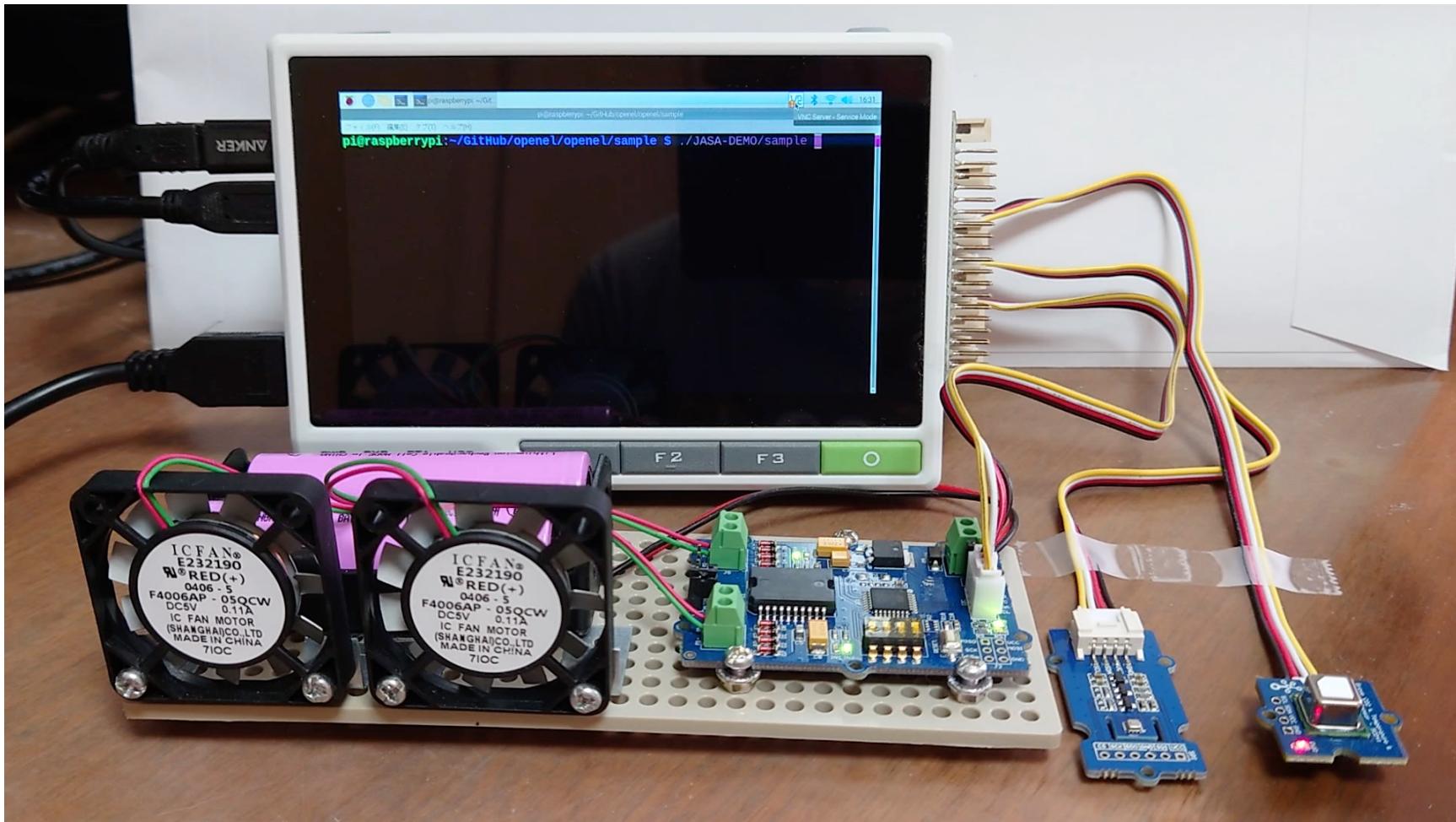
温度/湿度/気圧センサーのデモ



CO₂/温度/湿度センサーのデモ



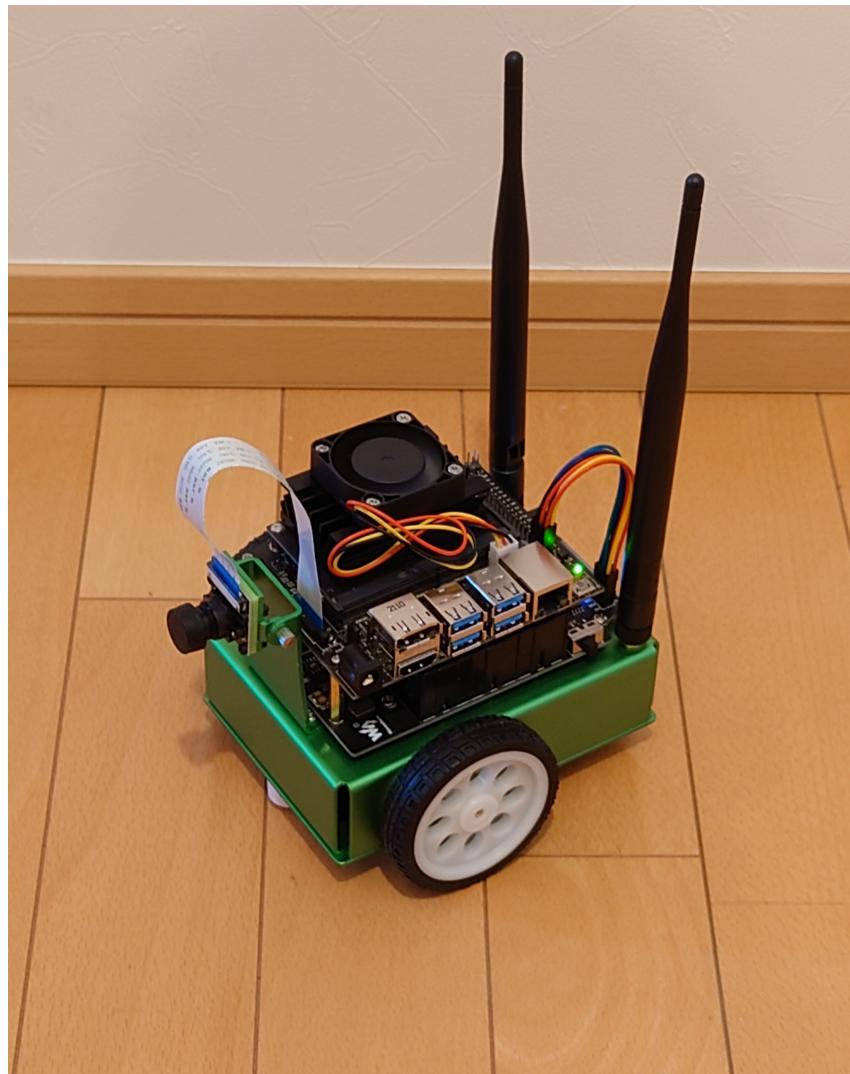
CO₂濃度を検知する自動換気システムのデモ



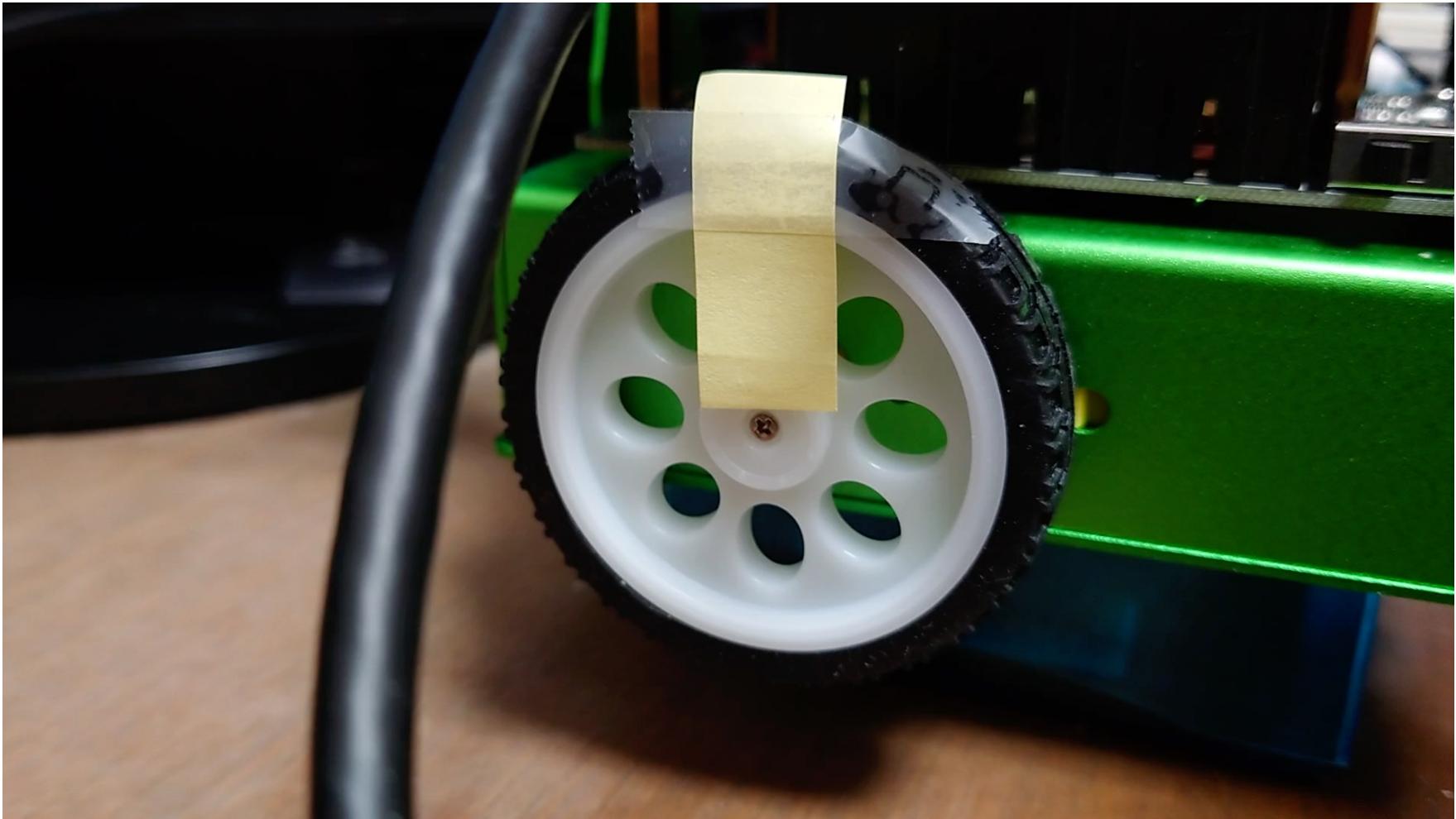
一般社団法人
組込みシステム技術協会
Japan Embedded Systems Technology Association



- 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



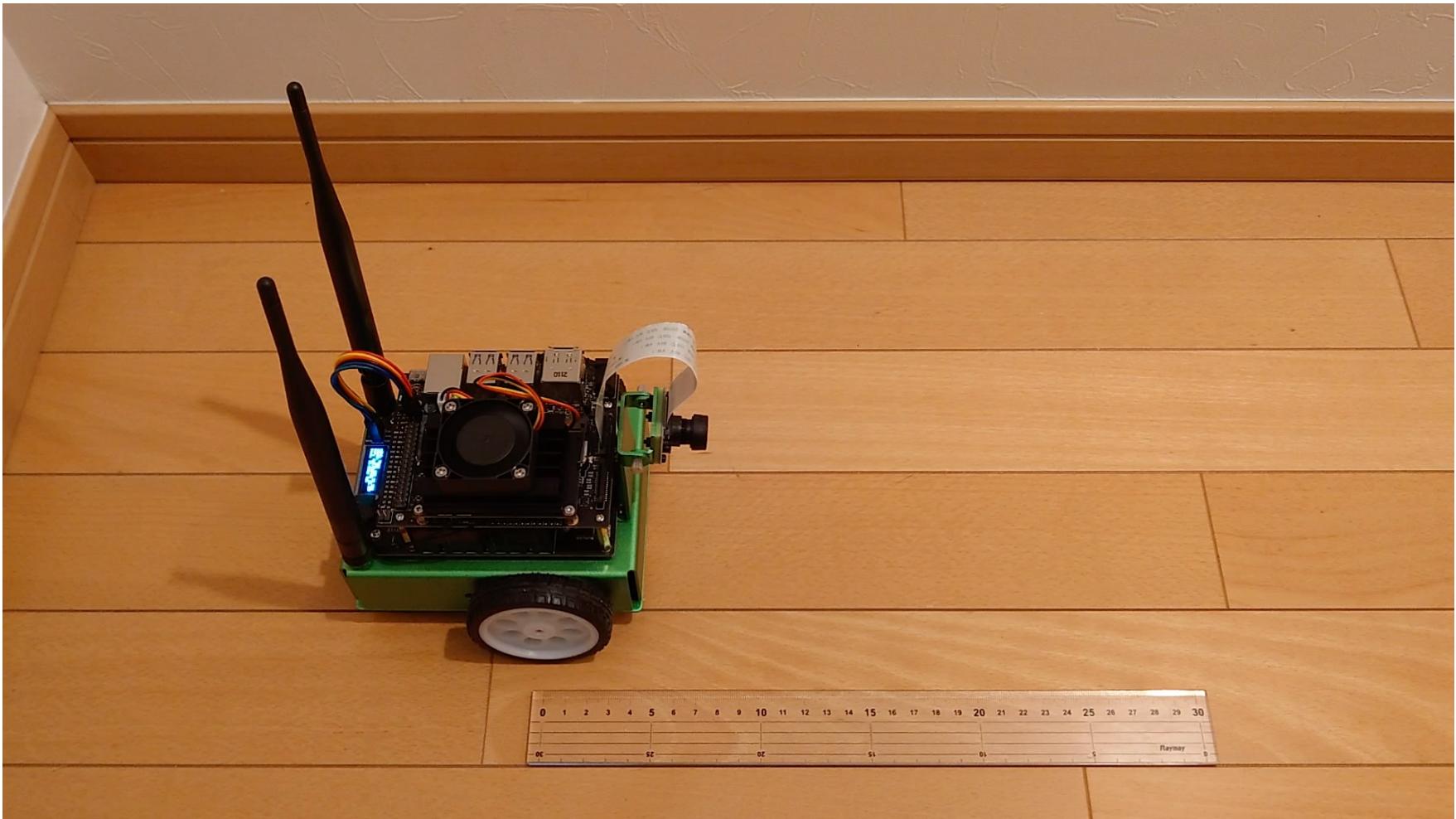
速度制御試験($-15\pi \sim 15\pi$ [rad/s])



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

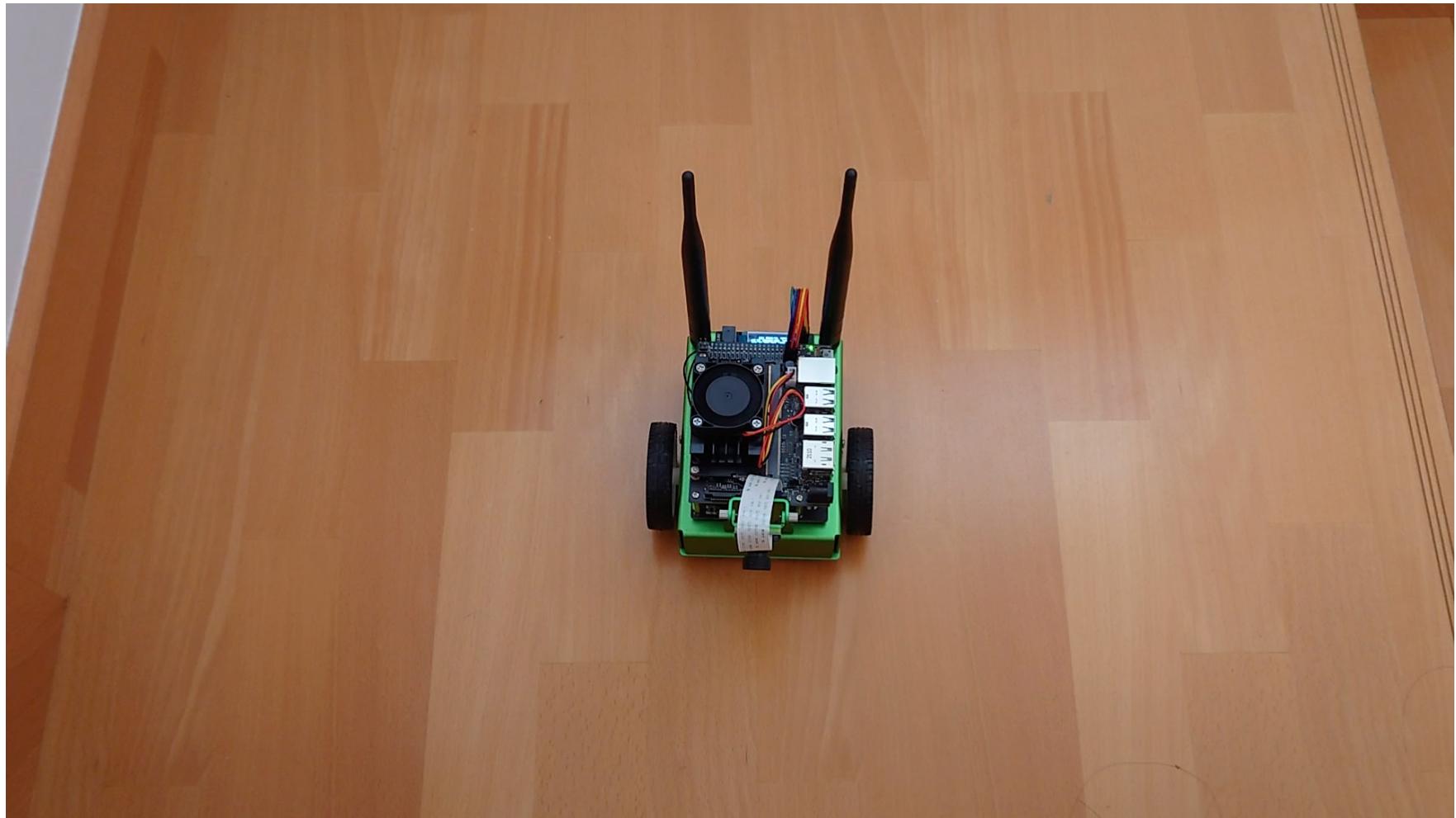


前後進試験





旋回試験



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

自己位置推定(Odometry)による移動



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



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 ← 充電検知
Charging.
notify_event201a : 2 ← 充電完了(12V以上)検知
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
```

↑ 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$ ./sampl
le ]
```



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>

The screenshot shows the homepage of the OpenEL website. The header features the text "OpenEL" in white on a dark blue gradient background. Below the header, there's a section titled "OpenEL" with a brief description: "OpenEL(Open Embedded Library) is a unified API(Application Programming Interface) for actuators and sensors. The specifications and implementation have been developed by JASA(japan Embedded Systems Technology Association) since 2011." It also mentions the current version is 3.2.0, released on October 30, 2021. A "Scope" section details the specification's purpose and scope, mentioning the Platform-Independent Model (PIM) and Hardware Abstraction Layer (HAL). A "Target readers" section lists software engineers and device vendors as intended users.

The screenshot shows the GitHub repository page for "openel/openel". The repository has 2 stars, 1 watch, and 2 forks. It contains 2 branches (master and develop), 2 tags, and 8 commits. The repository description states: "OpenEL(Open Embedded Library) is a unified API(Application Programming Interface) for actuators and sensors. The specifications and implementation have been developed by JASA(japan Embedded Systems Technology Association) since 2011." The repository includes documentation, include files, lib files, sample code, and a LICENSE file. A README.md file provides an overview of the library. The repository is associated with the URL "openel.github.io" and has labels for "embedded", "sensor", "hal", "actuator", "embedded-hal", and "hardware-abstraction-layer". It also shows releases (v3.2.0), packages (none), and languages (C 99.0% and C++ 5.0%).



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>

The screenshot shows the Docker Hub interface for the `openel/jetbot` Docker image. The page includes the image's name, a blue cube icon, the author (`openel`), the last update time (2 months ago), and a brief description (ROS2 component using JASA OpenEL for NVIDIA JetBot). Below the image, there are tabs for `Overview` and `Tags`. On the right, there is a section titled `How to use` with numbered steps: 1. SSH into Jetbot, 2. Pull docker image, 3. Run docker image, and 4. Set the jetbot number. Each step has a corresponding command line example. A `Docker Pull Command` box contains the command `docker pull 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の英語版ページを開いた。
<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開いた。
<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コンポーネントを開いた。
<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イメージを開いた。
<https://hub.docker.com/r/openel/jetbot>

2022年7月現在の対応デバイスと実装例

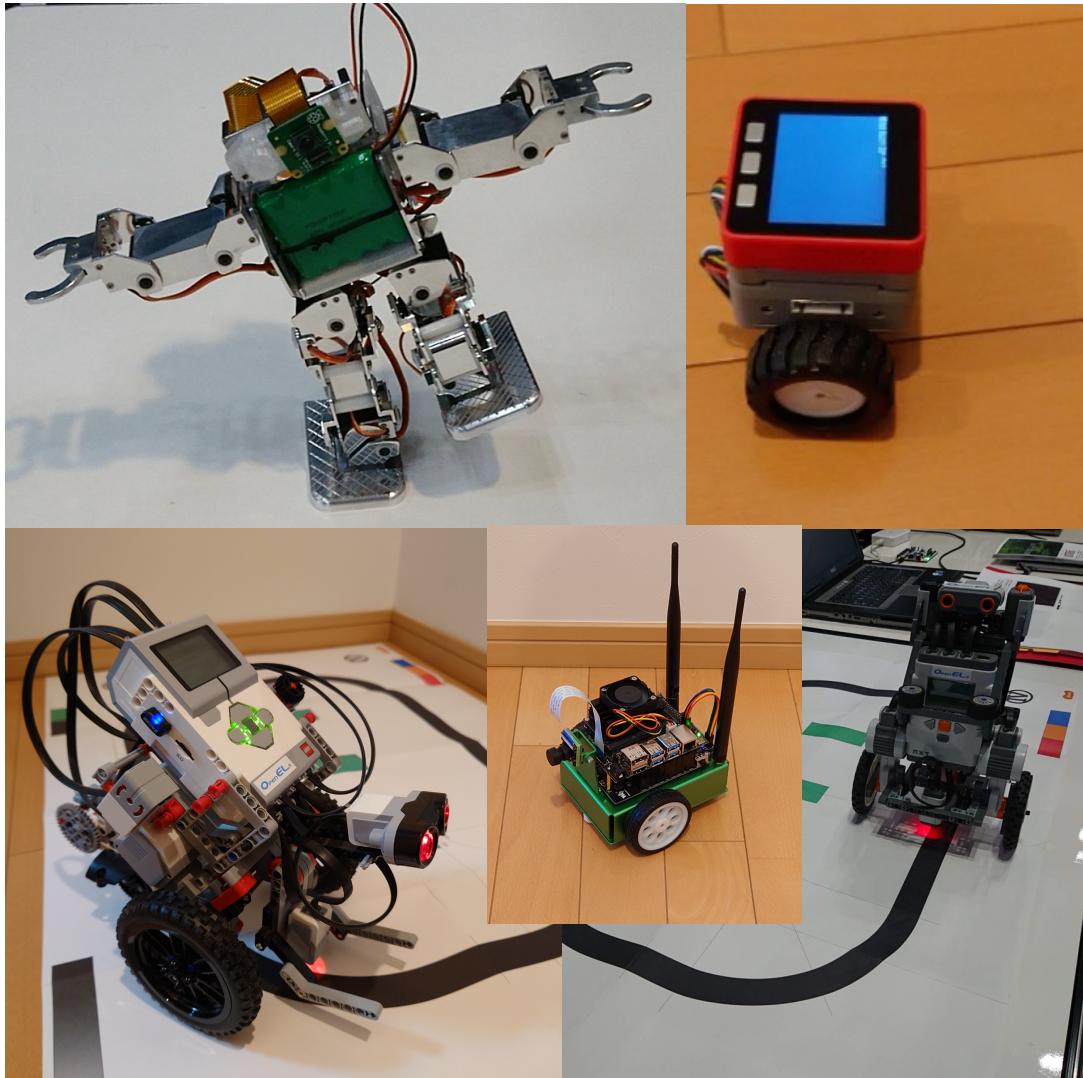


■ 対応デバイス

- 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





令和4年度の事業計画

2022年4月～、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中

■ 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)

■ OpenEL エコシステムの構築

■ OpenEL 勉強会の開催



■ ETロボコンSPIKEキットへの対応

走行体

モータ(尻尾用)(背面)

モータ(車輪用)

超音波センサ

ジャイロセンサ

モータ(アーム用)

カラーセンサ

RasPi用モバイルバッテリー

RaspberryPi(背面)

モータ(アーム用)

モータ(車輪用)

カラーセンサ





令和4年度の事業計画

■ シマフジ電機(株)SEMB1401への対応

The screenshot shows a web browser displaying the Shimafuji Electric website. The URL is shimafuji.co.jp/products/941. The page title is "SEMB1401 | シマフジ電機". The main content area is titled "SEM1401" and features a photograph of the IoT-Engine RZ/T1 module. To the left, there is a sidebar with a navigation menu:

- マイコン評価ボード
 - SBEV-RZ/V2L
 - R-IN32M4-CL3
(SBEV-RIN32M4CL3)
 - R-IN32M3 Module Evaluation Board
(SEM1320)
 - RZ/A2M Eva-Lite
 - SEM1401-3
 - SBEV-RZ/A2M

The main content area also includes descriptive text and a "CPU RZ/T1" section.

SEM1401

IoT-Engine RZ/T1

本製品はルネサスエレクトロニクス社製マイコン（RZ/T1）を搭載したIoT-Engine規格のCPUモジュールです。

fvlO技術を使用して、CPU負荷無しで、任意シーケンスのI2C通信を8チャネル同時に最大通信性能で実行することができます。（400Kbpsで隙間なく通信可能）

取得したデータをUSBやWiFi/Wi-SUNを介してサーバに送信することができます。

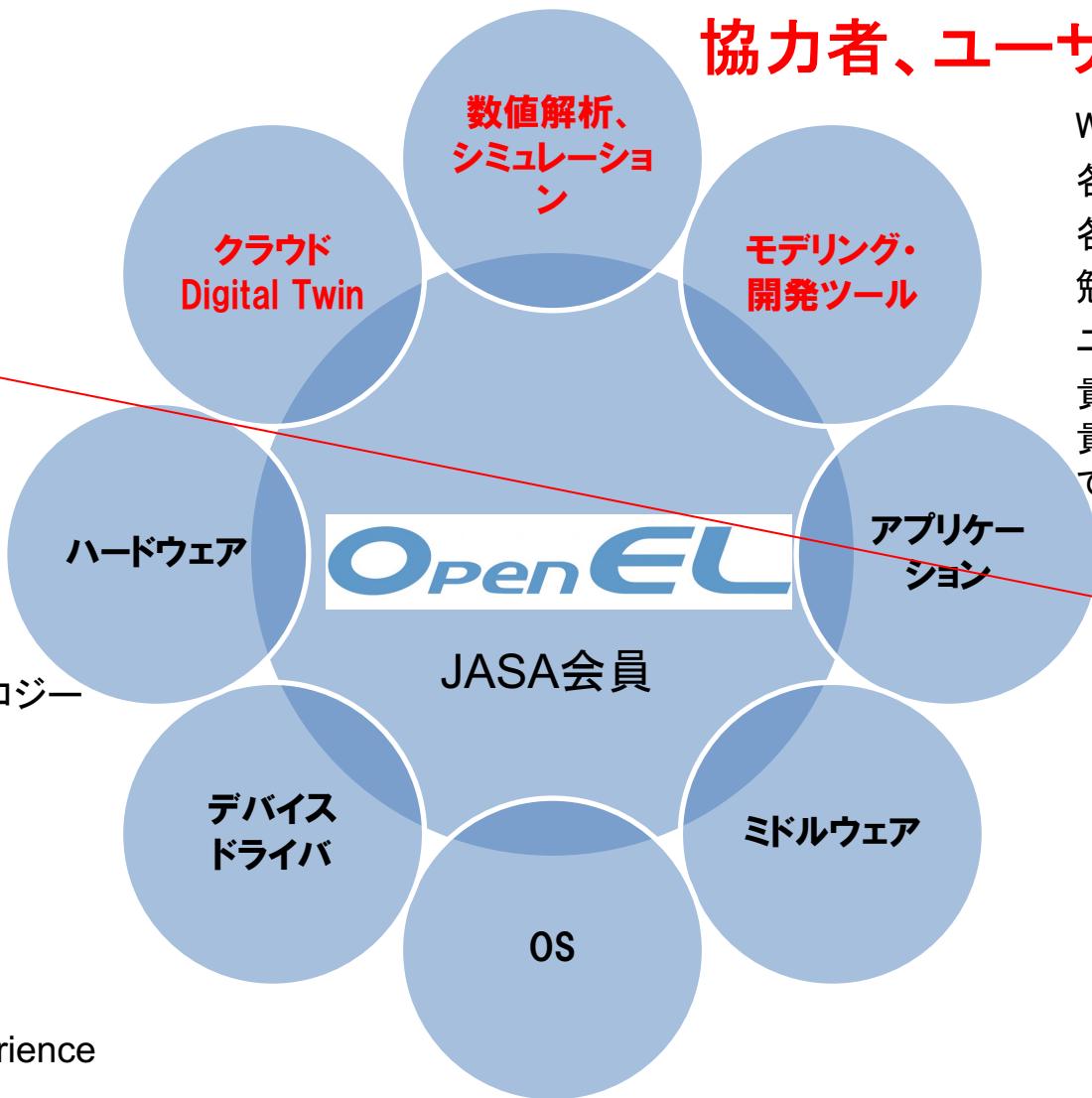
CPU
RZ/T1





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。



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」

2022/7/29 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。