Armadillo Base OS 開発ガイド

Version 1.1.0
2022/04/27

株式会社アットマークテクノ [https://www.atmark-techno.com]

Armadillo サイト [https://armadillo.atmark-techno.com]

Armadillo Base OS 開発ガイド

株式会社アットマークテクノ

Version 1.1.0
2022/04/27

目次

図目次

表目次

1. はじめに

本ドキュメントは、これから Armadillo Base OS を搭載した製品を用いたアプリケーションを設計・開発する方に向けて、Armadillo-IoT ゲートウェイ G4 を例として基本的な開発の流れについての情報を提供します。また、Armadillo-IoT ゲートウェイ G4 の特徴である NPU(Neural Processing Unit)を活かして、実際に機械学習を用いたサンプルアプリケーションを開発しつつ、Armadillo Base OS 搭載製品の開発方法について紹介します。その際に使用する物体検出モデル [1]も、既存のモデルをベースに学習を行う転移学習という手法で作成し、学習したモデルをアプリケーションに組み込みます。

なお、一般的なアプリケーション開発には企画、要件定義、設計、実装、検証、運用保守などといったステップがありますが、本ドキュメントはその全てを網羅するものではありません。本ドキュメントでは、Armadillo Base OS 搭載製品特有の手順や留意点に焦点を当てて説明していきます。説明のない箇所は、Armadillo Base OS であることによる特殊性が影響しない部分であるため、一般的なアプリケーション開発手法や、お客様の会社やグループ内のルールに則って開発を進めてください。

1.1. 本ドキュメントを読むことで習得できること

- Armadillo Base OS 上での開発手法
  - podman イメージ及びコンテナの作成と運用
  - ATDE を用いた各種開発手法
- 機械学習による物体検出アプリケーションを作る際の手法
  - TensorFlow を用いた既存の物体検出モデルの転移学習の方法
  - TensorFlow の SavedModel 形式 [2]のモデルを TFLite 形式 [3]に変換する方法
  - アプリケーションへの物体検出モデル組み込み

1.2. アイコンについて

具体的な機械学習を使用したサンプルアプリケーションの開発例を通して、Armadillo Base OS の開発手法を紹介します。本ドキュメント内で「図 1.1. 機械学習関連情報アイコン」のアイコンがある箇所は、機械学習について記述した内容であり、それ以外の箇所は機械学習に使用/不使用関係なく Armadillo Base OS に共通した内容になっています。機械学習を用いないアプリケーション開発についての情報だけを見たい方は、「図 1.1. 機械学習関連情報アイコン」のアイコンのない箇所を参照してください。

図 1.1 機械学習関連情報アイコン

また、「図 1.2. 補足情報アイコン」に示すアイコンが付いている箇所については、サンプルアプリケーションの開発には利用しませんが Armadillo Base OS を利用した開発においてよく行うことになる手順について紹介していますので、サンプルアプリケーション以外の開発時にお役立てください。

[1]「機械学習における、入力があったときにそれに何らかの評価を加えて出力値とするもの」を本ドキュメント内では「モデル」及び「推論モデル」と呼称します。
[2]TensorFlow Guide SavedModel 形式の使用: https://www.tensorflow.org/guide/saved_model?hl=ja
[3]TensorFlow Guide TensorFlow Lite: https://www.tensorflow.org/lite/guide?hl=ja

図 1.2 補足情報アイコン

1.3. 表記について

1.3.1. フォント

本書では以下のような意味でフォントを使いわけています。

表 1.1 使用しているフォント

フォント例	説明
本文中のフォント	本文
[PC ~]$ ls	プロンプトとユーザ入力文字列
text	編集する文字列や出力される文字列。またはコメント

1.3.2. コマンド入力例

本書に記載されているコマンドの入力例は、表示されているプロンプトによって、それぞれに対応した実行環境を想定して書かれています。「 / 」の部分はカレントディレクトリによって異なります。各ユーザのホームディレクトリは「 ~ 」で表します。

表 1.2 表示プロンプトと実行環境の関係

プロンプト	コマンドの実行環境
[ATDE ~/]$	ATDE 上の一般ユーザで実行
[armadillo /]#	Armadillo 上 Linux の root ユーザで実行
[container /]#	Podman コンテナ内で実行

1.3.3. アイコン

本ドキュメントでは以下のようにアイコンを使用しています。

注意事項を記載します。

1.4. サンプルソースコード

本ドキュメントで紹介するサンプルソースコードは、 https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/ からダウンロードできます。

1.5. ライセンス

本ドキュメントで紹介するサンプルソースコードは MIT ライセンス [4]の下に公開します。

ただし、 コンテナイメージ [https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/abos-dev-guide-v1.0.0.tar]及び コンテナ内のソフトウェア [https://

[4]http://opensource.org/licenses/mit-license.php

download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/read_meter.tar.gz]に含まれる、アナログメーターのサンプル動画 [5]及び、本ドキュメント内における当該の動画のスクリーンショットについては、クリエイティブコモンズの表示-継承 4.0 国際ライセンスの下に提供されています。

図 1.3 クリエイティブコモンズライセンス

[5]kaggle Pressure Gauge Reader Data: https://www.kaggle.com/juliusgrassme/pressure-gauge-reader-data

2. 本ドキュメントについて

2.1. 本ドキュメントで実施する作業

本ドキュメントでは、Armadillo Base OS 上で実際にサンプルアプリケーションの作成を行います。

「図 2.1. Armadillo Base OS 上でのアプリケーション開発の流れ」に、Armadillo Base OS 上でのアプリケーション開発の基本的な流れについて示します。本ドキュメントにおいても、この流れに沿って開発を行います。

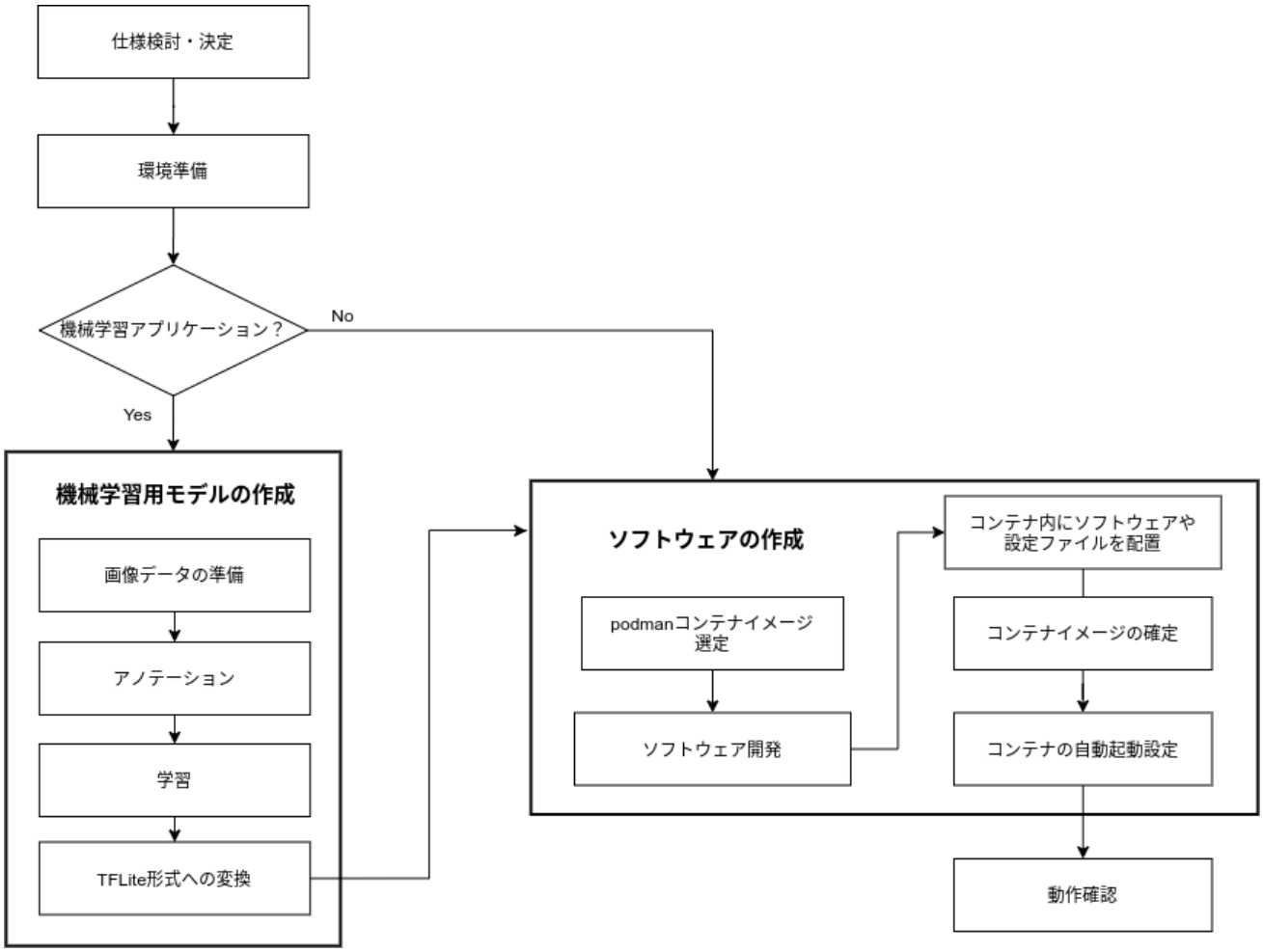


図 2.1 Armadillo Base OS 上でのアプリケーション開発の流れ

Armadillo Base OS における、アプリケーションの開発時の流れをより詳細にした図を「図 2.2. 詳細なアプリケーション開発の流れ」に示します。図中の破線内は、本サンプルアプリケーションのように機械学習をアプリケーションに組み込む場合に必要な手順であり、主に使用する推論モデルのチューニングを行っています。それ以外の機械学習を用いないアプリケーションの場合には、破線内の処理は必要ありません。

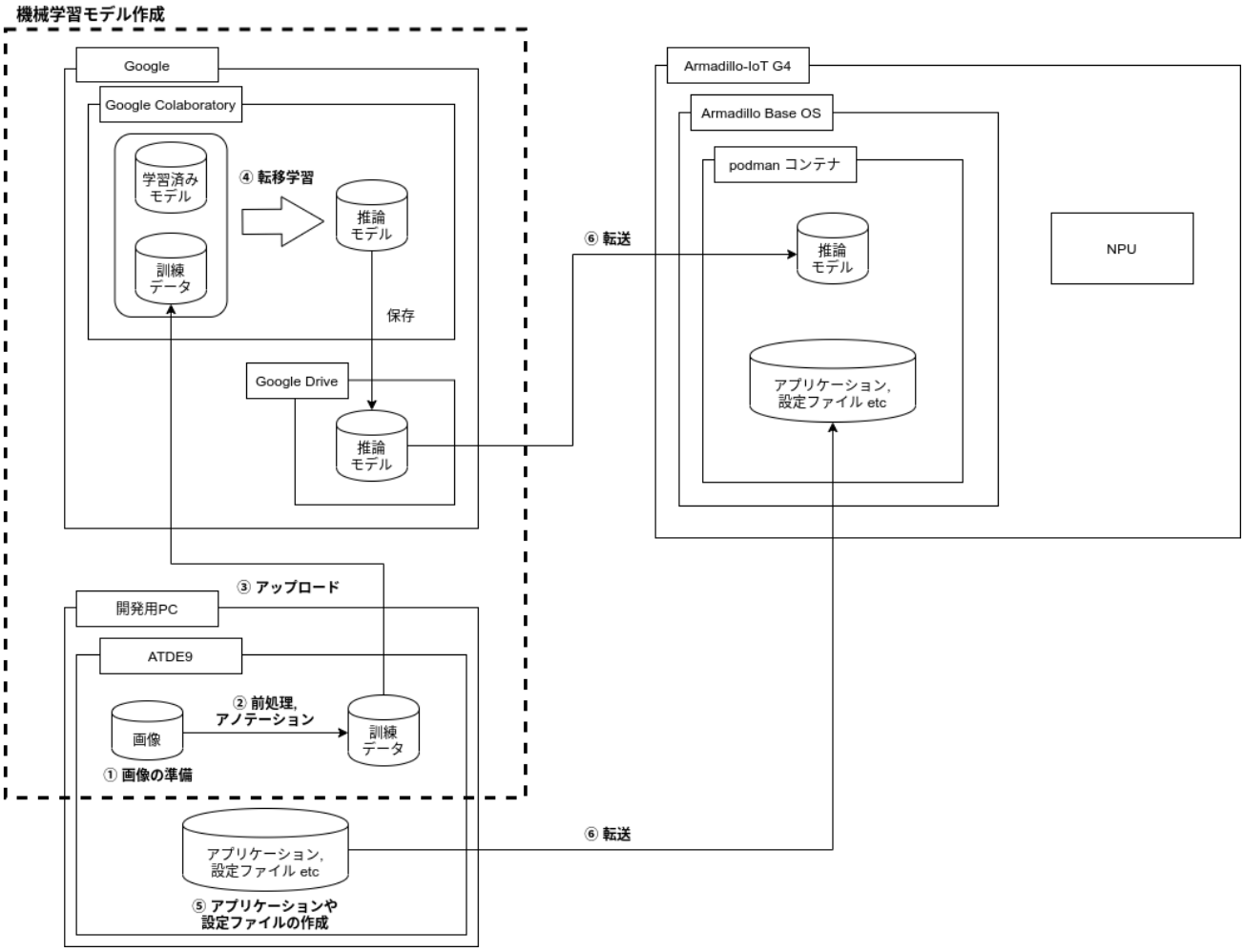


図 2.2 詳細なアプリケーション開発の流れ

本サンプルアプリケーション作成の大まかな流れは、機械学習部分を含むため、以下の通りです。

1. ATDE 上に転移学習用の画像データを用意する
2. 画像データに対してラベル名や、その位置などを含む情報を付与(アノテーション)して訓練データとする
3. 訓練データを Google Colaboratory にアップロードする
4. Google Colaboratory 上で学習済みモデルと組み合わせて転移学習を行う
5. 推論モデルを使用したアプリケーション本体や、付随する設定ファイル等を ATDE 上で作成する
6. 4 で作成した推論モデルと、5 で作成したアプリケーション等を Armadillo に転送する

各手順については本書内で詳しく紹介していきます。

2.2. 本ドキュメントの成果物

本ドキュメントの手順を踏むことで最終的に、丸型アナログメーター自動読み取りサンプルアプリケーションが作成できます。出来上がるサンプルアプリケーションの動作について「図 2.3. サンプルアプリケーションの動作」に示します。

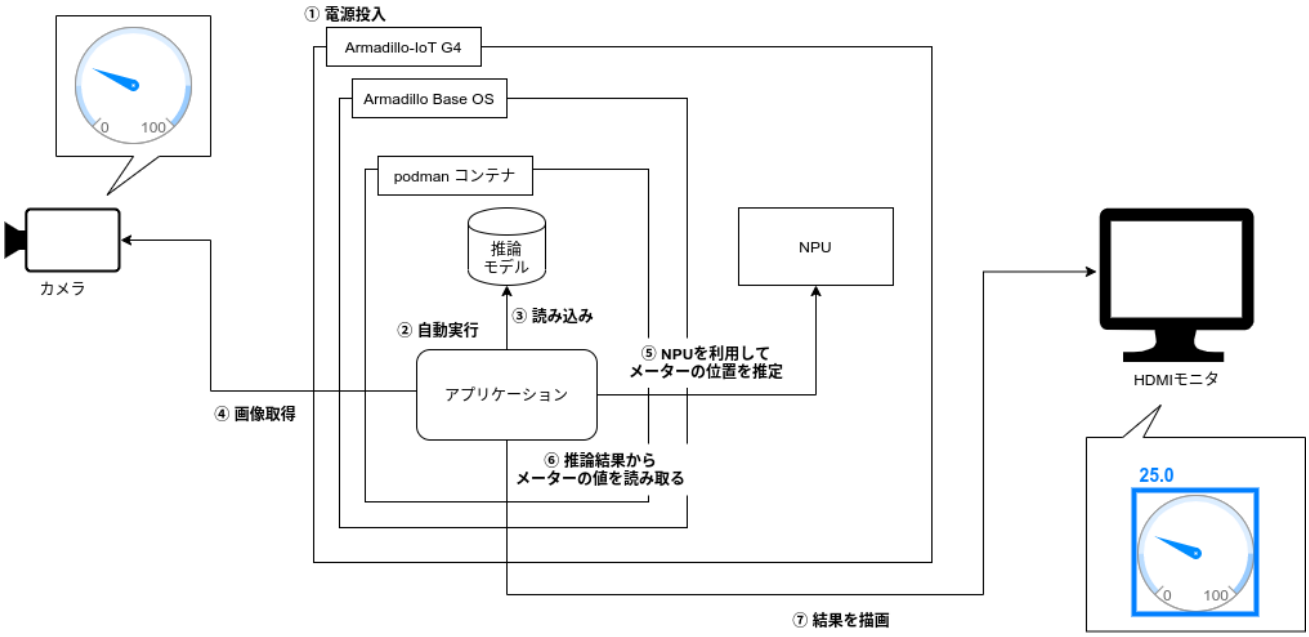


図 2.3 サンプルアプリケーションの動作

アプリケーションの挙動としましては以下の通りです。

1. Armadillo-IoT ゲートウェイ G4 に電源を投入する
2. podman コンテナとアプリケーションが自動的に起動する
3. 初期化処理として推論モデルをロード
4. USB カメラから画像を取得
5. NPU を利用して画像データからアナログメーターの位置推定を行う
6. 5 の結果をもとにアナログメーターが指す値を読み取る
7. 6 で得た結果を画像と共に HDMI モニタに表示
8. 4〜7 を繰り返す

USB カメラから取得した画像内のどこにアナログメーターが存在するかを検知するために、機械学習による物体検出を利用しています。また、アナログメーターの読み取りには OpenCV による画像解析を使用しています。

詳細な仕様については、「6.1. サンプルアプリケーションの詳細」で紹介します。

3. 準備

ここからは「2.2. 本ドキュメントの成果物」で紹介したサンプルアプリケーションの作成前に、必要な準備事項について紹介していきます。

3.1. 必要なもの

本ドキュメントでは以下のものを使用します。

- Armadillo-IoT ゲートウェイ G4 開発セット [https://armadillo.atmark-techno.com/armadillo-iot-g4/AGX4500-C00D0]
- 以下を満たす開発用 PC
  - インターネットに接続可能
  - (必要ならば)VMware が動作可能
- USB カメラ(なくても可)
- HDMI モニタ
- microHDMI-HDMI 変換ケーブル
- Google アカウント

サンプルアプリケーションの映像入力は Armadillo 内のサンプル動画でも代用できますので、USB カメラはなくても試すことができます。

3.2. 事前準備

3.2.1. 開発環境の準備

Armadillo-IoT ゲートウェイ G4 製品マニュアルの「 4. Armadillo の電源を入れる前に [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.4.0/ch04.html]」を参考に Armadillo 向け開発環境である ATDE を整備し、シリアルコンソール経由で Armadillo-IoT ゲートウェイ G4 を操作できるように準備してください。

3.2.2. 機械学習向け開発環境の準備 機械学習

本ドキュメントで作成するサンプルアプリケーションでは、既存の学習済みモデルを自分のやりたいこと向けに学習し直す「転移学習」を行います。転移学習を行うことで、学習に必要な教師データ [1]の数が減ったり、学習にかかる時間の短縮につながったりするなどのメリットがあります。転移学習の詳細については TensorFlow の転移学習紹介ページ [https://www.tensorflow.org/js/tutorials/transfer/what_is_transfer_learning?hl=ja]も参照してください。

注意点としまして、Armadillo-IoT ゲートウェイ G4 に搭載されている GPU/NPU は、学習済みモデルを用いて推論を行うことはできますが、学習することには向いていません。そのため、GPU が利用で

[1]教師データとは、機械学習の教師あり学習において、ニューラルネットワークに予め与えられる例題と答えについてのデータを指します。

きる別な環境で学習を行い、その結果得られた推論モデルを Armadillo に送って推論を行うことになります。

お使いの開発用 PC に機械学習開発に十分な GPU が搭載されているならば機械学習の開発環境とすることが可能ですが、搭載されていない場合は、Google が提供している Google Colaboratory [https://colab.research.google.com/notebooks/welcome.ipynb?hl=ja]というサービスで代用できます。

Google Colaboratory は、Google アカウントを持っていれば無料で利用できる、ブラウザ上で Jupyter Notebook [2]形式で Python や Shell コマンドなどを記述・実行できるサービスです。特徴として以下のような点が挙げられます。

- Web ブラウザアプリケーションとして動作し、ローカルに環境構築が不要
- 無料で GPU を利用することができる
- Google Drive を用いたファイルの共有や保存を行うことができる

本ドキュメントでは Google Colaboratory を使用して推論モデルの作成を行います。Google Colaboratory を使用するためには Google アカウントを作成しておく必要があります。Google Colaboratory には無料版と有料版があり、無料版には連続利用可能時間などのいくつかの制約がありますが、本ドキュメントのサンプル開発は無料版で十分です。

3.2.3. 接続物の準備

「図 3.1. Armadillo-IoT ゲートウェイ G4 への周辺機器の接続」、「図 3.2. 接続後のイメージ」に示すようにように、各種接続物を Armadillo-IoT ゲートウェイ G4 に接続してください。

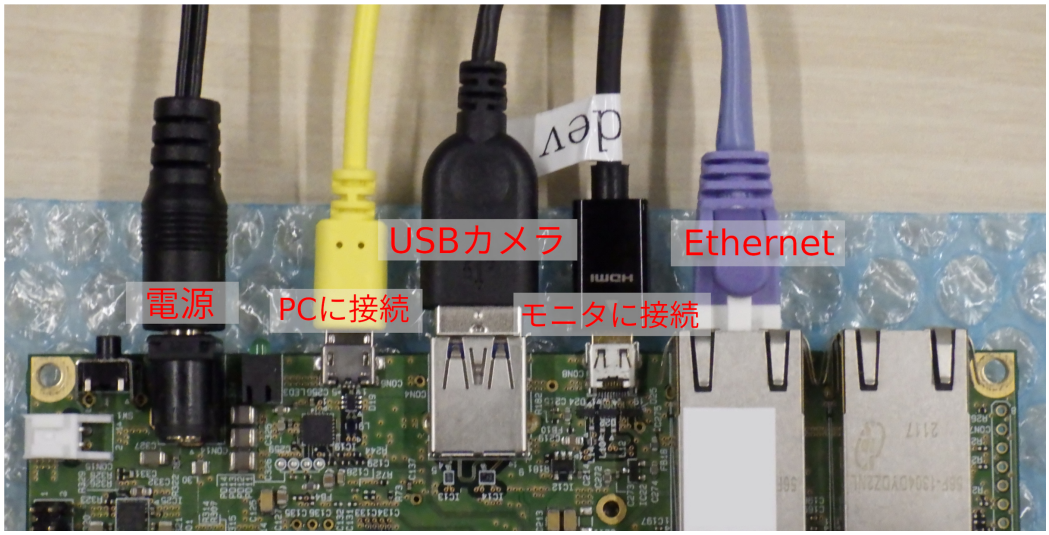


図 3.1 Armadillo-IoT ゲートウェイ G4 への周辺機器の接続

[2]Jupyter Notebook: https://jupyter.org/

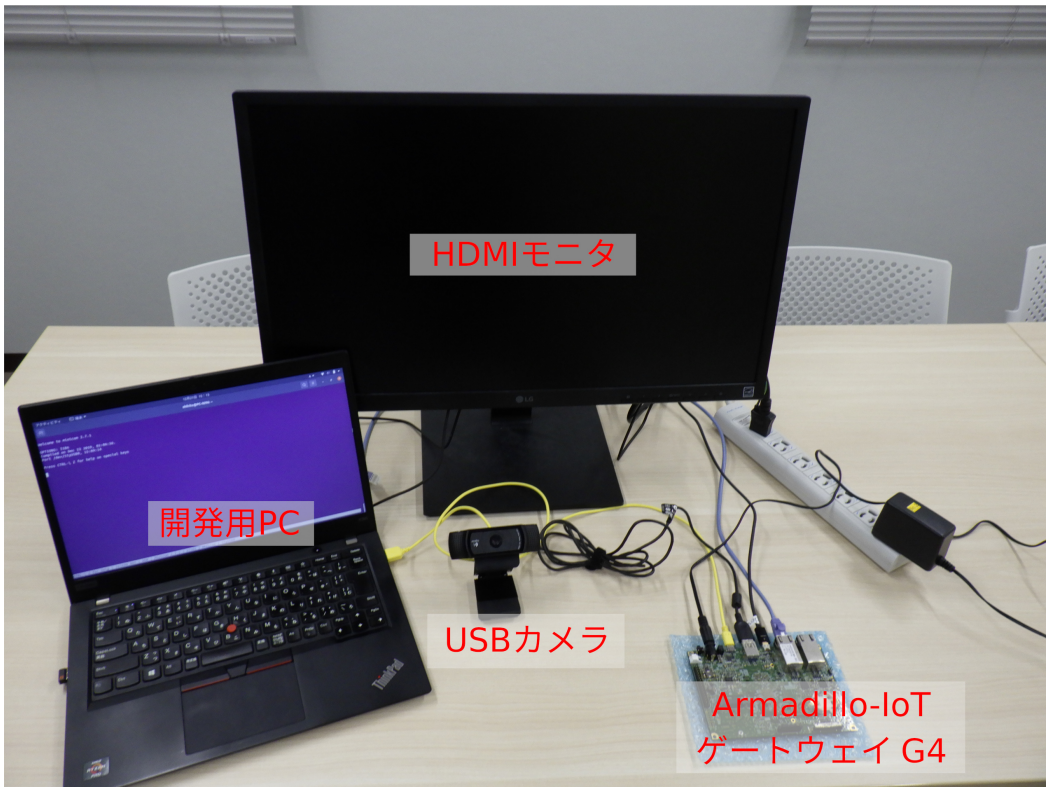


図 3.2 接続後のイメージ

4. 基本的な開発の流れ

Armadillo Base OS 上でのアプリケーション開発の大まかな流れについては、「図 2.1. Armadillo Base OS 上でのアプリケーション開発の流れ」に示した通りです。

Armadillo を用いた開発は、作業用 PC(または Armadillo 向けの開発環境が揃った仮想マシンである ATDE)内でアプリケーションの開発を行い、Armadillo へデプロイする方法と、Armadillo 内で開発を行う 2 パターンがあります。どちらの手法でも問題ありませんが、本ドキュメントでは基本的に作業用 PC 内に ATDE を立ち上げ、その中で開発を行っていきます。ATDE を用いた開発時のイメージを「図 4.1. 開発時のイメージ」に示します。

Armadillo Base OS において、ユーザーアプリケーションは podman コンテナ上で動作させることを前提としています。Armadillo 上で podman コンテナを構築して、そのコンテナ内に開発したアプリケーションが動作する環境を整えることでアプリケーションを実行することができるため、使用時に Armadillo Base OS をあまり意識する必要はありません。

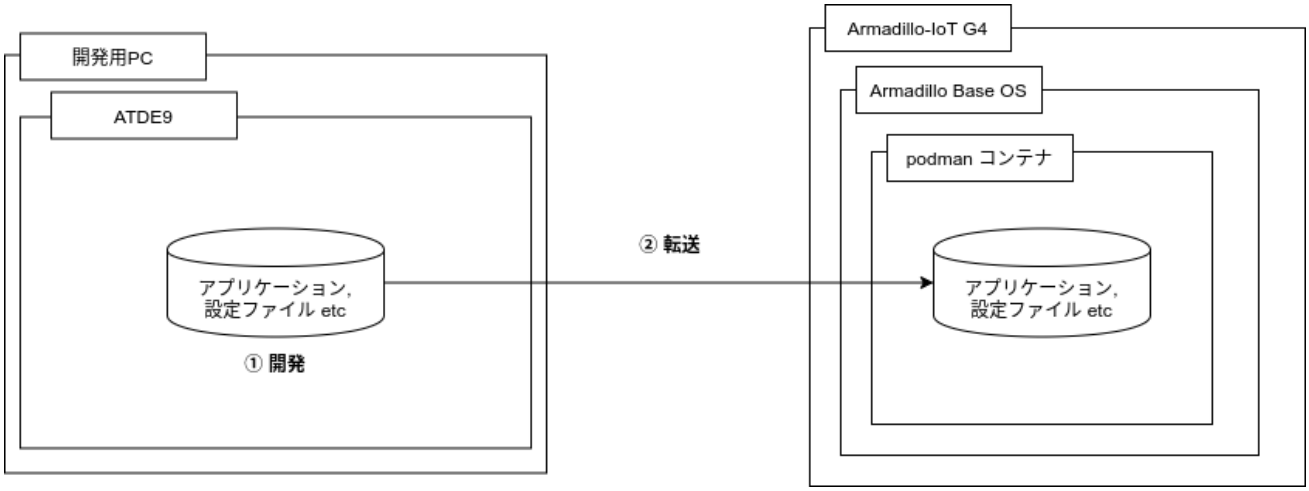


図 4.1 開発時のイメージ

各手順について詳細は「6. サンプルアプリケーションの作成」内で説明します。

4.1. 機械学習を用いたアプリケーションの開発の流れ 機械学習

「図 2.1. Armadillo Base OS 上でのアプリケーション開発の流れ」に示しているように、今回作成するような機械学習を用いたアプリケーション開発においては、仕様検討までは機械学習を用いないアプリケーションと同様の手順ですが、実現したいことに合わせた推論モデルを生成する手順が別途必要になります。

今回は既存の物体検出モデルをベースに転移学習するため、以下のような手順となっています。あくまでも一例ですので、実現したいことによって手順は異なることに注意してください。

1. 教師データの元となる画像を準備
2. アノテーションして教師データを作成
3. 学習

4. TFLite 形式への変換
5. ソフトウェア開発(以降は機械学習を用いないアプリケーションと同様)

各手順についての詳細は「6.2. 推論モデルの作成 機械学習」で説明します。

5. 仕様を検討・決定する

Armadillo Base OS 上での開発に限った話ではありませんが、作成するアプリケーションの仕様の検討を行います。本ドキュメントでは機械学習を用いたアプリケーションを開発するため、この章では主に機械学習に重きを置いて説明します。

5.1. Armadillo Base OS におけるアプリケーション仕様検討

「4. 基本的な開発の流れ」でも説明したとおり、Armadillo Base OS では基本的にユーザーアプリケーションを podman コンテナ上で実行します。そのため、実行環境として Armadillo Base OS を意識する必要はありません。従来の Armadillo 製品を使用した開発を行ったことがある方であれば、その際の製品開発の仕様を引き継ぐことも可能です。

この後の「5.2. podman コンテナイメージの選定」でも紹介していますが、実現したいことに合った実行環境をコンテナイメージとして自由に選択できるため、Armadillo Base OS におけるアプリケーション仕様の検討時の制限は少ないはずです。

5.2. podman コンテナイメージの選定

作成するアプリケーションの実行環境としてどのコンテナイメージをベースとするかは最終的なイメージサイズにも関わりますので、検討しておく必要があります。

podman は、同じくコンテナを扱えるソフトウェアである Docker [https://www.docker.com/]と基本的に互換性があります。そのため、podman で使用するコンテナイメージとして Docker Hub [https://hub.docker.com/]に存在するイメージも使用することができます。

コンテナイメージサイズをなるべく小さくしたいならば alpine、Python で書いたアプリケーションを手軽に実行したいならば Debian ベースの python:3.x-slim-bullseye など、作成するソフトウェア仕様に沿ってベースイメージを決定します。

また、Armadillo-IoT ゲートウェイ G4 において映像出力を行う、もしくは NPU を使用して推論を行う場合は、そのために必要なライブラリがありますので、 アットマークテクノが提供している Debian ベースの podman コンテナイメージ [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/container]を使用することをお勧めします。

5.2.1. コンテナを用いたシステムの設計 補足情報

使用するコンテナイメージの選定と共に、コンテナをシステムの中でどのように用いるかを検討しておく必要がある場合があります。本サンプルアプリケーションのように単一のコンテナのみで完結するシステムであれば難しくはないですが、管理・運用のために機能毎にコンテナを複数個に分けてコンテナ間で通信し合う場合であったり、コンテナの外側である Armadillo Base OS のコマンドやリソースへアクセスしながら使用したりする場合などには、どのようなシステム設計をすべきか難しい場合があります。

本ドキュメントでは、Armadillo Base OS 上でコンテナを使用したシステムを構築する際によくあるパターンや注意点について「7.4. コンテナデザインパターン」にまとめてありますので参考にしてください。

5.3. ハードウェアの拡張 補足情報

サンプルアプリケーションでは行いませんが、Armadillo の拡張インターフェースに I2C や SPI などのインターフェースを持つ外部デバイスを接続する場合、対象のピンの機能やパラメータを変更しなければならない場合があります。

ハードウェア拡張時のパラメータ変更の方法について、詳しくは「7.3. Device Tree を変更しハードウェアを拡張する」を参照してください。

5.4. 機械学習を用いたアプリケーションについて 機械学習

機械学習を用いたアプリケーションを開発する場合で、今までに機械学習を用いた開発を行ったことがない場合は、少し検討が必要かもしれません。機械学習という言葉が独り歩きして、「機械学習は簡単になんでもできる」など間違った認識で開発が行われる場合があるためです。

機械学習アプリケーションは、それ以外のソフトウェアでは実現できないようなことを実現できる強力なものであり、それだけに魅力的ですが、万能ではありません。当然デメリットも存在することを忘れてはいけません。機械学習アプリケーションのデメリットを知り、そのデメリットをシステムが許容できるかを検討する必要があります。場合によっては機械学習を使わないで、別な手段で問題を解決するという選択をすることも重要になります。

以下に機械学習を用いたアプリケーション開発のデメリットを挙げます。

5.4.1. 学習の手間

自分で一から推論モデルを作成しようとすると、膨大な量の教師データが必要になります。物体検出サンプルならば何千枚といった画像データを用意し、それに手作業でラベル付けと物の位置を指定してはじめて教師データとなります。場合によっては、全ての画像データに対してリサイズや色変換など前処理をする必要がある場合もあり、学習には多くの手間がかることに注意してください。

本ドキュメントの物体検出サンプルでは転移学習を用いることで用意する教師データの数を減らしています。

5.4.2. 推論の速度と精度

推論の速度は、推論モデルと推論を行うハードウェアに依存します。ハードウェアは今回は Armadillo で固定なので問題ではありませんが、推論モデルによっては一度の推論に長い時間がかかる場合もあり、速度を要するシステムでは使えないということもあります。速度は一度推論モデルを作ってみなければわからず、基本的に精度とトレードオフですので、高い精度のモデルほど速度は遅いです。

推論の精度は、推論モデルに依存します。学習時にどれだけの数の訓練データを入力したか、訓練データや特徴量は正しいものだったか、過学習は起こっていないかなど、気にするべき箇所は多々あります。さらに、精度はどこまで学習を積んでも 100%になることはありません。速度や精度は推論モデルのチューニングによっては改善が期待できるかもしれませんが、限界があります。

推論モデルの速度と精度については予め実現したいラインを設定し、そこに近づけられるように試行錯誤と検証を重ねる必要があります。最終的に出来上がる推論モデルの性能に合わせて、システム全体で推論の速度や誤検知を許容できる仕様にしておく必要があります。

5.4.3. 開発時の技術的負債

機械学習はドキュメントやテストコードがなかったり、いきあたりばったりの設計が残ってしまうなど、技術的負債が多くなる傾向があります。それは、以下のような機械学習を用いたシステム構築の難しさが原因としてあります。

- 前述の通り試行錯誤回数が多くなる
- 確率的な処理があるために自動テストが難しい
- 長期運用しているとトレンドの変化などで入力の傾向が変化する
- 実験コードやパラメータが残りやすい

6. サンプルアプリケーションの作成

ここまでで Armadillo Base OS での開発環境の整備や、開発手順がわかったところで、いよいよサンプルアプリケーションを作成していきます。

初めに、USB カメラを使用しないサンプルアプリケーションを含む podman コンテナを作成し、その後それをベースに USB カメラに対応した podman コンテナを作成します。

ここからはサンプルアプリケーション作成手順を各ステップ毎に説明していきますが、 こちら[https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/abos-dev-guide-v1.0.0.tar]から完成したアプリケーションを含む podman コンテナイメージをダウンロードできます。

podman コンテナイメージのインポート方法については、「7.2. 作成したコンテナを他の Armadillo に組み込む」を参照してください。

6.1. サンプルアプリケーションの詳細

今回作成するサンプルアプリケーションの大まかな概要は、「2.2. 本ドキュメントの成果物」で示したとおりですが、詳細について紹介します。

6.1.1. 詳細な動作

今回作成するアプリケーションの詳細な動作を「図 6.1. サンプルアプリケーションの詳細な動作の流れ」にまとめます。

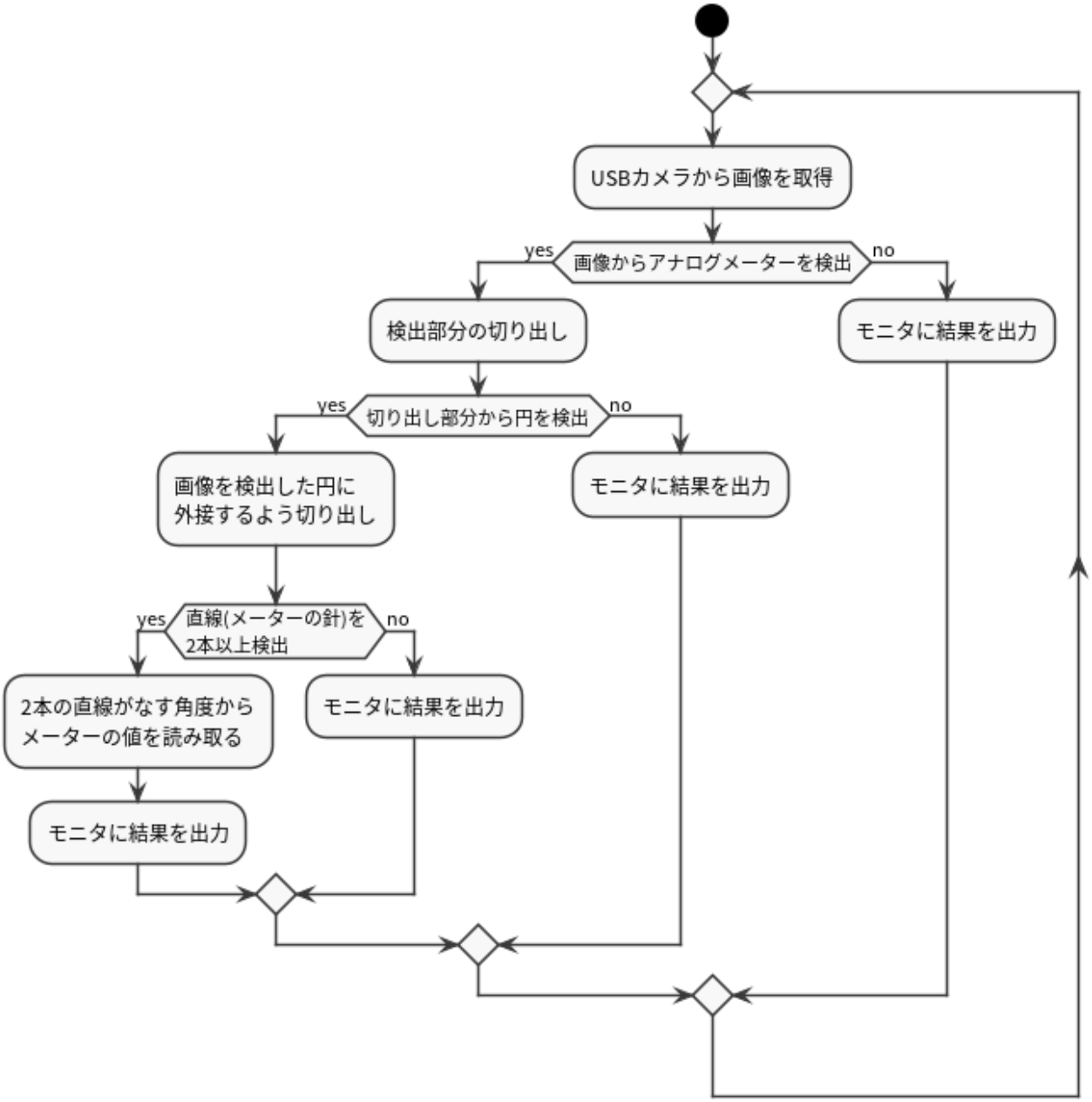


図 6.1 サンプルアプリケーションの詳細な動作の流れ

図中の各処理について説明します。

6.1.1.1. USB カメラから画像を取得

基本的に画像の操作は、Python の OpenCV ライブラリを用いて行っていきます。

初めに、USB カメラまたは任意の動画ファイルから画像を取得します。

図 6.2 元画像

6.1.1.2. 画像からアナログメーターを検出

この部分に機械学習の物体検出を用います。予めアナログメーターの画像を学習させておいた TFLite 形式の推論モデルを用いて、取得した画像に対して推論を実行し、画像内のアナログメーターの有無とその位置を検出します。

検出できたならば次の処理に移り、検出できなければ取得した画像を HDMI モニタに出力して再度画像の取得を行います。

図 6.3 物体検出した箇所をもとに画像を切り出し

6.1.1.3. アナログメーター検出箇所から円を検出

アナログメーターを検出した部分のみを切り出し、その中でハフ変換を用いて円とその中心座標を検出します。

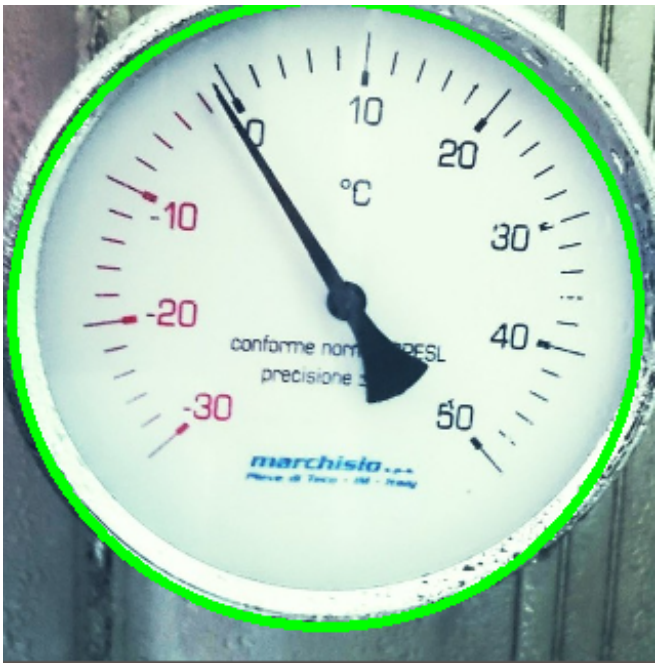


図 6.4 円を検出

検出できたならばその円に外接する四角形で元の画像を切り抜きます。これによって、物体検出による検出位置のズレや誤検出を吸収しつつ、画像内のアナログメーター部分のみを切り抜いています。

図 6.5 画像の枠が円に外接するようにリサイズ

円を検出できなければ、取得した画像を HDMI モニタに出力して再度画像の取得を行います。

6.1.1.4. メーターの読み取り

アナログメーターの針の形をなす 2 本の直線をハフ変換を用いて検出します。その 2 本の直線から針の角度を算出して、予め設定した parameter_sample.json の情報と照らし合わせてアナログメーターが示す値を求めています。

図 6.6 直線を検出

求めた値を元画像に描画した上で HDMI モニタに出力します。

直線が検出できなかった場合や、求めた値がそのアナログメーターでは表示できない値であった場合は、読み取った値は画像に描画されずに HDMI モニタに出力します。

その後、再度画像の取得に戻ります。

以上が、本サンプルアプリケーションの動作です。

6.2. 推論モデルの作成 機械学習

ここからはサンプルアプリケーションを作成手順を紹介していきます。

まずはじめに、アプリケーション本体の前に物体検出モデルを作成します。以下の作業を本ドキュメントでは説明のため ATDE 上で行っていますが、他の PC 上などでも問題ありません。ATDE 以外の環境で実行する際には、適宜各ソフトウェアの使用方法などを調べた上でご使用ください。

6.2.1. 教師データの用意

今回行う物体検出における転移学習では、既存の物体検出用の推論モデルである MobileNet SSD v2 をベースに、新たに検出させたい物が写った画像を含む教師データを用意して学習させることで、対象物を検出するモデルを効率よく作成します。

物体検出における教師データは、検出させたい物が写った画像と、それらの画像内のどこに何があるかをラベル付けしたアノテーション情報の 2 つをまとめたものを指します。

機械学習用に教師データをデータセットという形で公開/販売しているところも存在しているので、使用上のライセンスをよく確認した上でそちらを使用しても問題ありません。

本ドキュメントでは既存のデータセットは使用せず、Web 上から画像を収集して実際にアノテーションを行います。

6.2.1.1. 画像の用意

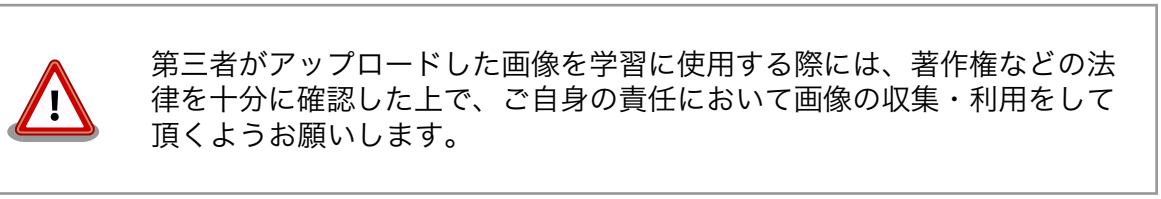
まずは教師データの元となる画像を収集します。一から推論モデルを学習するのであれば数千枚ほどのデータがほしいところですが、今回は転移学習を用いるため、最低 100 枚程度からある程度の精度を出せます。

「表 6.1. 画像収集に使用できるツール」に示すように、様々な機械学習向けに画像を収集できるツールが公開されているので、そちらを利用することで効率よく画像を集めることができます。

表 6.1 画像収集に使用できるツール

ツール名	概要
icrawler [https://github.com/hellock/icrawler]	Python で動作するウェブクローラのフレームワーク
Bing Image Search API [https://docs.microsoft.com/ja-jp/azure/cognitive-services/bing-image-search/overview]	Microsoft 社が無料提供している Bing 画像検索 API
Google Images Download [https://github.com/hardikvasa/google-images-download]	Google の画像検索から画像を収集できる Python スクリプト

上記の他にも、機械学習などで使用できる画像を収集できるツールや Web サイトなどが存在しているので、興味のある方は調べてみてください。



第三者がアップロードした画像を学習に使用する際には、著作権などの法律を十分に確認した上で、ご自身の責任において画像の収集・利用をして頂くようお願いします。

今回は Google Images Download を使用して、画像データを収集していきます。

まずインストールについてですが、Google Images Download は pip でインストールできます。しかし執筆時点では最新の Google 画像検索 API に対応できていませんので、有志がパッチを適用したものをインストールします。Google Images Download のインストール時に必要なパッケージも合わせてインストールします。

```
[ATDE ~]$ sudo apt update && sudo apt upgrade -y
[ATDE ~]$ sudo apt install -y python3-pip
[ATDE ~]$ git clone https://github.com/Joeclinton1/google-images-download.git gid-joeclinton
[ATDE ~]$ cd gid-joeclinton && sudo python3 setup.py install
```

図 6.7 Google Images Download のインストール

Google Images Download を使用して、gauge(アナログメーター)の画像を収集します。Google Images Download は、ダウンロードする画像の拡張子と縦横のサイズ、ダウンロードする数を指定することができます。

ここでは、Google で"gauge"と画像検索した際に得られる画像を jpg 形式で 100 枚ダウンロードします。

```
[ATDE ~/gid-joeclinton]$ cd
[ATDE ~]$ mkdir -p dataset_gauge/annotations && cd dataset_gauge ❶
[ATDE ~/dataset_gauge]$ googleimagesdownload -k "gauge" -l 100 -f jpg ❷
Item no.: 1 --> Item name = gauge
Evaluating...
Starting Download...
: (省略)
Everything downloaded!
Total errors: 2 ❸
Total time taken: 54.75878143310547 Seconds
```

図 6.8 Google Images Download の実行

❶ 作業ディレクトリを作成し、移動します。

❷ Google Images Download を実行します。

❸ ファイルによってはエラーとなるものがあり、必ずしも指定した枚数の画像を収集できるとは限りません。

Google Images Download を用いて 100 枚より多くの画像を収集する際には、別途 Google Chrome と chromedriver をインストールする必要があります。詳しくは 公式ドキュメント [https://google-images-download.readthedocs.io/en/latest/troubleshooting.html#installing-the-chromedriver-with-selenium]を参照してください。

「図 6.8. Google Images Download の実行」の実行例ならば 98 枚の gauge で画像を検索した際の結果が得られました。

```
[ATDE ~/dataset_gauge]$ ls downloads/gauge/ | wc -l
98
```

図 6.9 Google Images Download でダウンロードしたファイル

また、場合によっては壊れたファイルをダウンロードしてくる場合がありますので、その場合は適宜手動で削除してください。

その後、「図 6.10. 画像ファイルを連番でリネーム」を実行して、ファイル名をリネームしておきます。

```
[ATDE ~/dataset_gauge]$ cd downloads/gauge/
[ATDE ~/dataset_gauge/downloads/gauge/]$ ls | awk '{ printf "mv %s %06d.jpg¥n", $0, NR }' | sh
[ATDE ~/dataset_gauge/downloads/gauge/]$ ls
000001.jpg
000002.jpg
: (省略)
```

図 6.10 画像ファイルを連番でリネーム

6.2.1.2. アノテーション

画像が用意できたら、次はその画像内のどこに何が写っているかを示すラベル付け(アノテーション)を行っていきます。画像データの数によりますが、推論モデル学習の一連の流れの中で一番工数がかかる手順がこのアノテーション作業です。

「表 6.2. 物体検出アノテーションに使用できるツール」にアノテーションする際によく使われるツールをまとめました。どれも Windows/Linux 共に使用可能ですので、お好きなものをご使用ください。

表 6.2 物体検出アノテーションに使用できるツール

ツール名	概要
labellmg [https://github.com/tzutalin/labellmg]	Python+Qt で記述されたオープンソースアノテーションソフトウェア
VoTT [https://github.com/microsoft/VoTT]	Microsoft 社が提供しているオープンソースアノテーションソフトウェア
Labelbox [https://github.com/Labelbox/Labelbox]	リッチな GUI で多機能な Labelbox 社製ツール。基本無料で利用量が一定を超えると課金される点に注意

本ドキュメントでは、labellmg を使用してアノテーションを行っていきます。

labellmg は「図 6.11. labellmg をインストールする」に示すコマンドを実行することでインストールできます。

```
[ATDE ~]$ sudo pip3 install labelImg
```

図 6.11 labellmg をインストールする

labellmg を起動するには、「図 6.12. Linux 環境における labellmg 実行例」に示すコマンドを実行します。

```
[ATDE ~]$ labelImg
```

図 6.12 Linux 環境における labellmg 実行例

実行すると、「図 6.13. labellmg の起動」のような labellmg のアプリケーションウィンドウが立ち上がります。

図 6.13 labellmg の起動

まず、アノテーション情報の保存形式を指定します。今回は PascalVOC 形式で保存します。

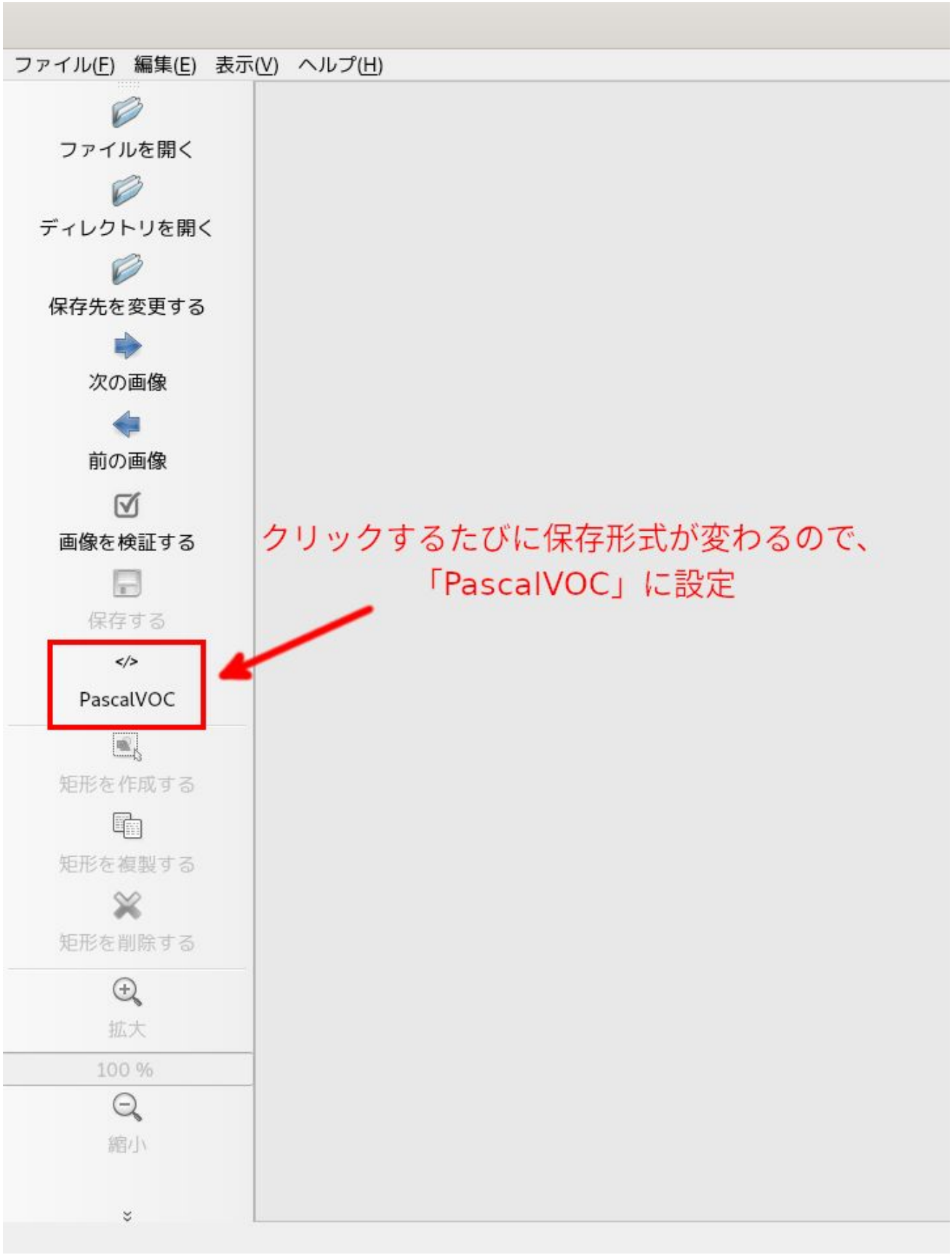


図 6.14 出力形式の変更

アノテーションを開始するために、画面左の「ディレクトリを開く」をクリックして、アノテーションしたい画像が入ったディレクトリ(今回は~/dataset_gauge/downloads/gauge/)を指定します。

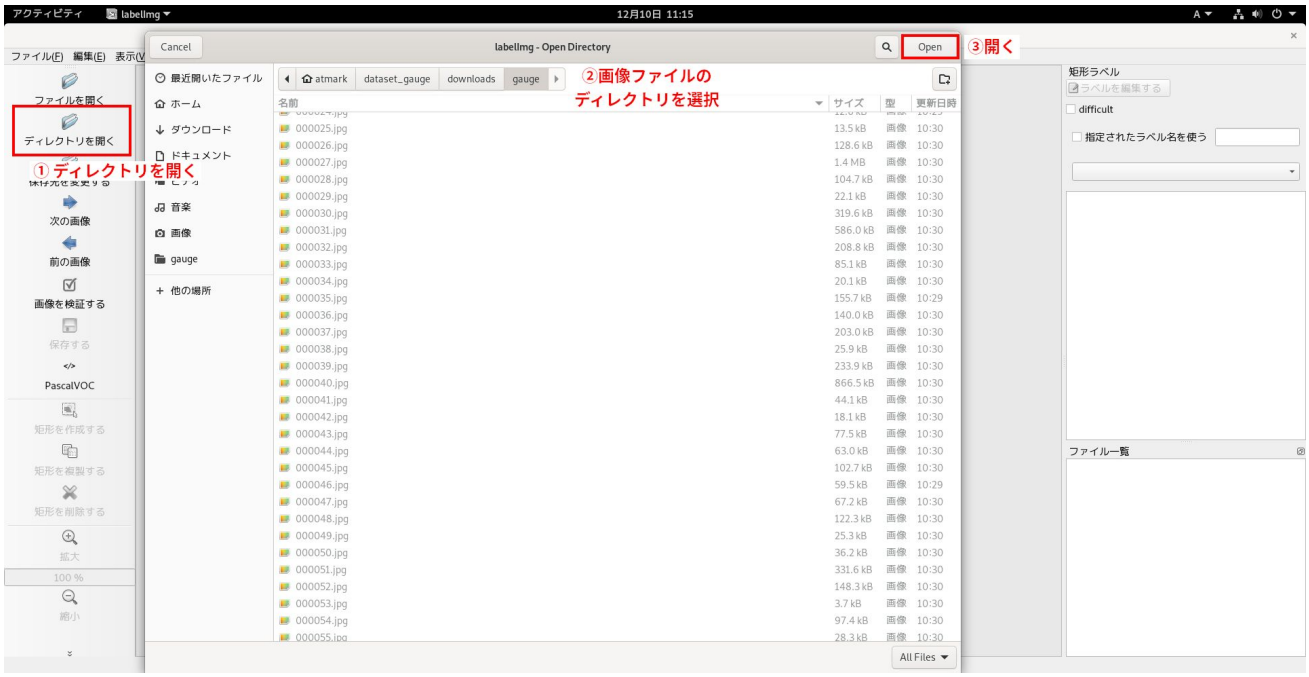


図 6.15 アノテーション元の選択

さらに、アノテーション情報をどこに保存するかを設定します。画面左の「保存先を変更する」をクリックして、アノテーション情報を保存したいディレクトリ(今回は~/dataset_gauge/annotations/)を指定します。

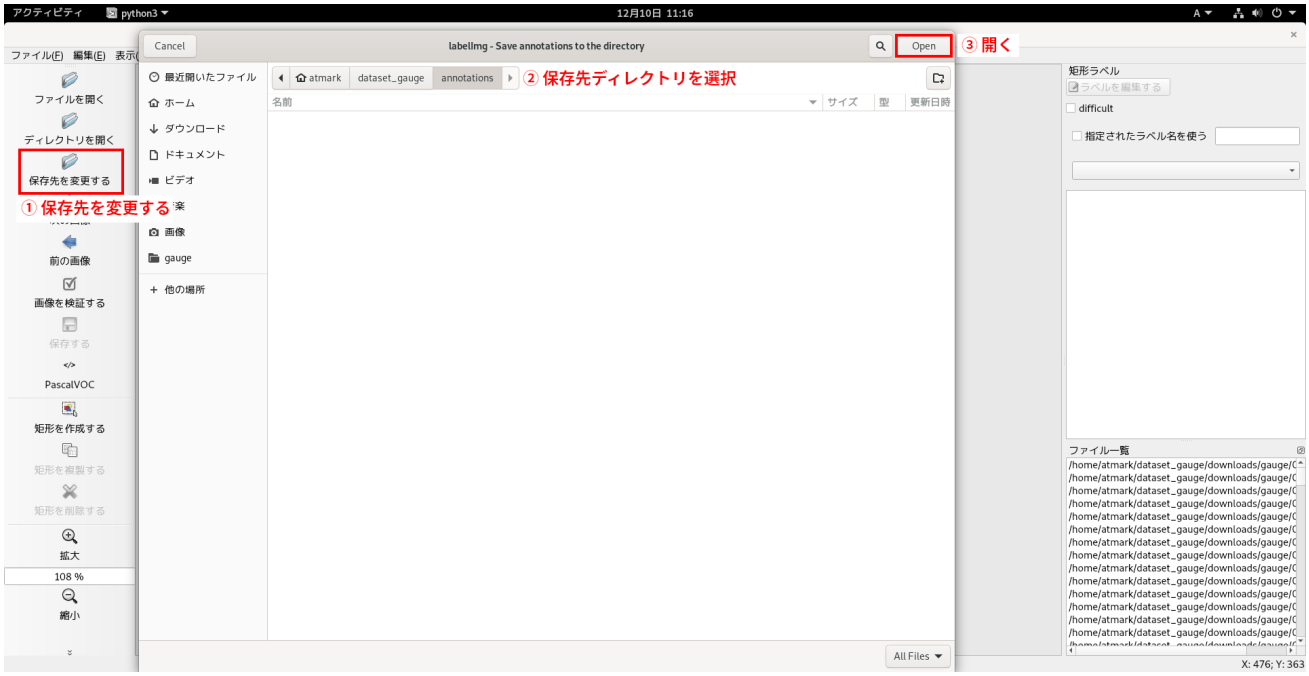


図 6.16 アノテーション情報の保存先の選択

その後作業効率化のために、画面上部のツールバーの[表示]から、以下を有効化をしておくことをお勧めします。

- 自動で保存する
- 単一のラベルだけをアノテーションする

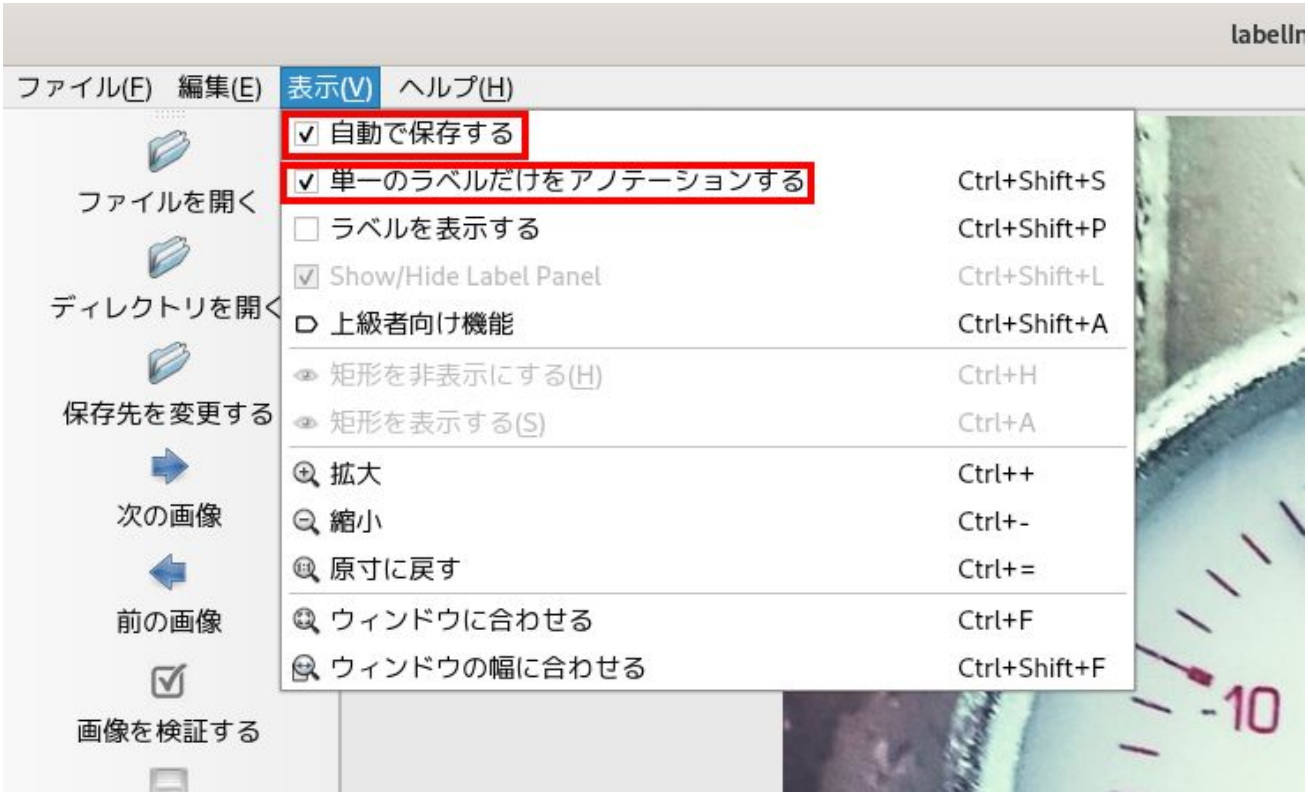


図 6.17 作業効率化のための設定 1

さらに、画面右側にて、「指定されたラベル名を使う」にチェックを入れ、ラベル名を「gauge」とします。

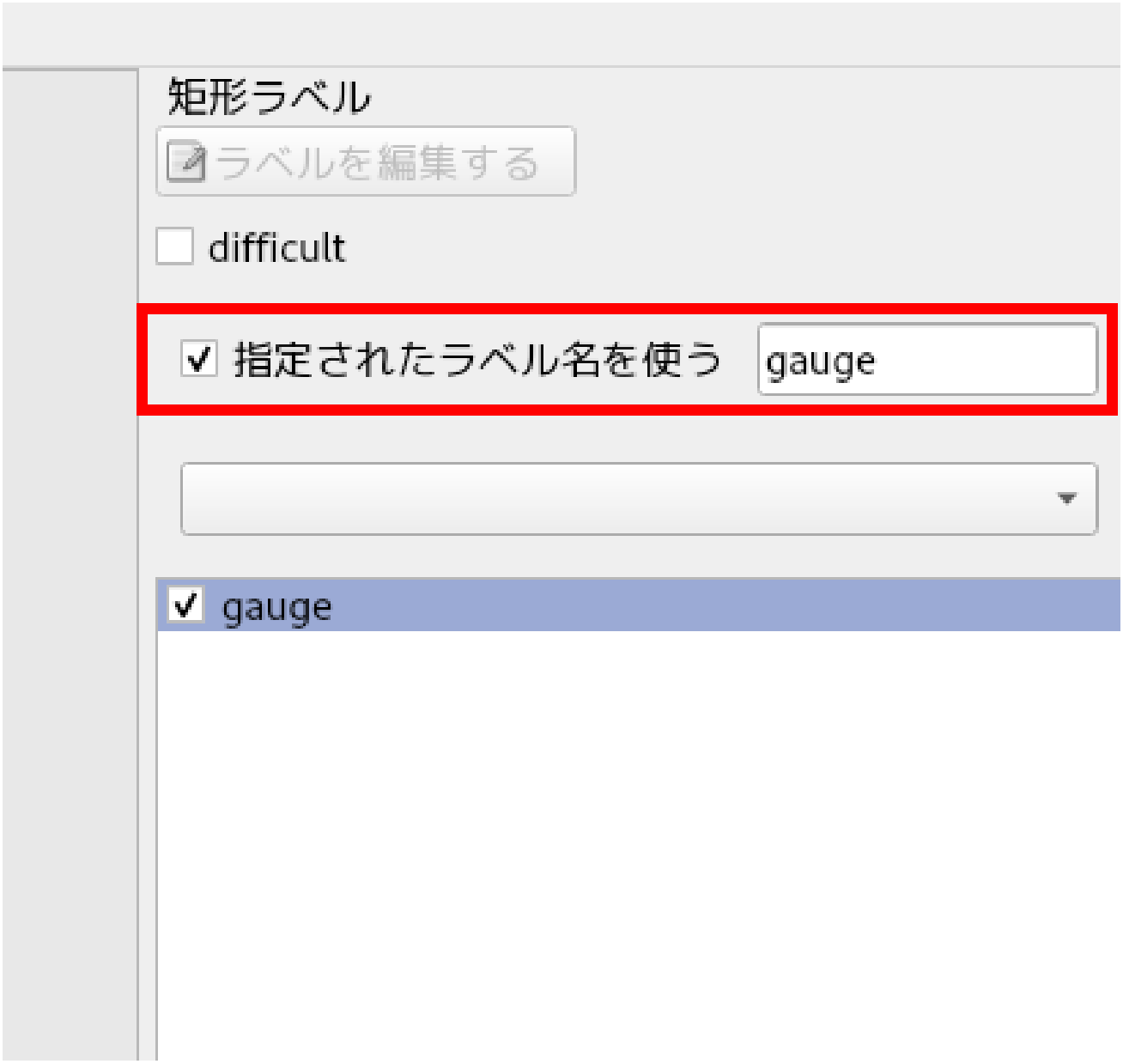


図 6.18 作業効率化のための設定 2

「表 6.3. labellmg でよく使用するショートカットキー一覧」に、labellmg を使用したアノテーション作業中によく用いるショートカットキーを示します。作業効率化のために使用することをお勧めします。詳しくは、labellmg の画面上部のツールバーの[ヘルプ]→[ショートカット一覧を見る(英語)]から確認できます。

表 6.3 labellmg でよく使用するショートカットキー一覧

ショートカットキー	役割
W	矩形選択モードに切り替え
D	次の画像へ
A	前の画像へ

上記設定後、W キーを押下して矩形選択モードに切り替え、マウス左ドラッグで画像内の対象物を選択します。選択すると自動的に選択範囲が gauge でラベル付けされます。

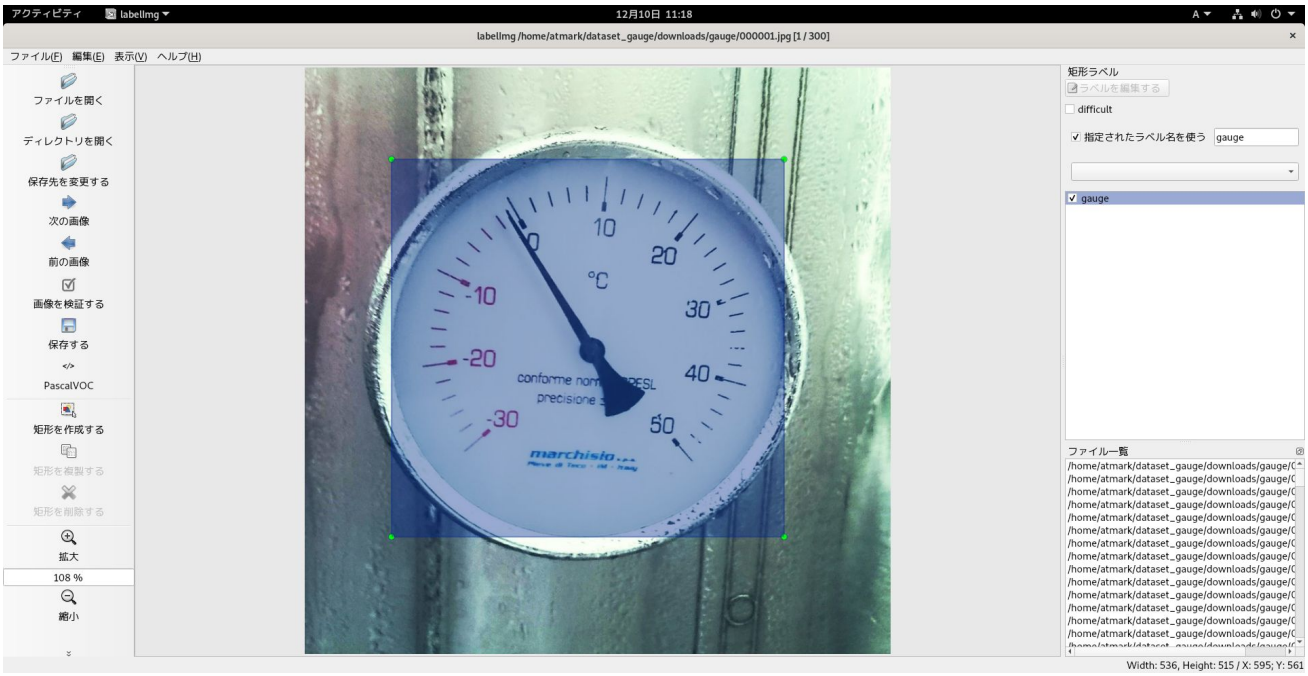


図 6.19 ラベル付け例

D キー押下で次の画像へ切り替えられます。

以上の手順を用意した全ての画像に対して行っていきます。

アノテーションする際の注意事項としまして以下が挙げられます。

- アノテーションのポリシーは作業中一貫させること
  - 見切れているものはアノテーションする/しない
  - 対象物の裏面や別角度の写真をアノテーションする/しない
  - 最終的に検出したい物を常に意識して作業する
- 用意した全ての画像に対して必ずしもアノテーションしなければならないわけではない
  - 画像を用意したが、対象物がよく写っている画像でないならアノテーションしないことも重要

アノテーションが完了したら、保存先のディレクトリにアノテーション情報(.xml ファイル)が保存されていることを確認してください。

```
[ATDE ~/dataset_gauge/annotations]$ ls *.xml
000001.xml
000002.xml
:（省略）
```

図 6.20 アノテーション情報ファイルを確認

次に、label_map.pbtxt というファイルを~/dataset_gauge/xmls/内に作成します。物体検出は、画像を推論モデルに入力し、結果として「どこに」、「何が」あるかが返ってくるのですが、「何が」の部分は 1 から始まる ID で返ってきます。label_map.pbtxt は、その ID とラベル名を紐付けるファイルです。

本サンプルアプリケーションでは"gauge"のみを認識し、画面上にはメーターが指し示す値を表示させる仕様なので、アプリケーション本体ではこのラベルファイルは使用しません。しかし、この後の推論モデルの学習時に使用するので、ここで作成しておきます。

今回作成する label_map.pbtxt の内容を「図 6.21. label_map.pbtxt 作成例」に示します。

```
item {
  id: 1
  name: 'gauge'
}
```

図 6.21 label_map.pbtxt 作成例

以上でアノテーション作業は完了です。

6.2.2. 推論モデルの生成

教師データが用意できたら、いよいよ学習して独自の物体検出モデルを作成します。

「3.2.2. 機械学習向け開発環境の準備 機械学習」でも述べましたが、今回推論モデルの学習は Google Colaboratory 上で行います。

ATDE の Web ブラウザで、Armadillo-IoT_G4_create_detection_model [https://colab.research.google.com/github/atmark-techno/Armadillo-Base-OS-dev-guide-colab/blob/master/Armadillo_IoT_G4_create_detection_model.ipynb] を Google Colaboratory で開いてください。そのノートブック内に転移学習と TFLite 形式への変換の具体的な手順をテキストとソースコード内のコメントにてまとめています。

各セルに対して Ctrl+Enter を押し、セルごとに実行していくことで転移学習済み TFLite モデルを取得することができます。実行には、全体で数十分程度時間がかかることにご注意ください。

結果として Armadillo 上で動作する TFLite 形式のモデル(model.tflite)が得られます。

以上で推論モデルの作成は完了です。

6.3. podman コンテナを作成する

推論モデルが作成できたらアプリケーションの作成を行いますが、先にそのアプリケーションが動作するコンテナイメージを作成します。

今回作成するサンプルアプリケーションでは、NPU や VPU をアプリケーション内で使用するため、アットマークテクノが提供しているサンプルコンテナイメージをベースとして使用します。

6.3.1. 開発前の準備

ここからは Armadillo-IoT ゲートウェイ G4 上で作業を行います。

Armadillo Base OS では全ての変更は RAM 上に保持され、電源を切った段階でそれらの変更は消えて変更前の状態に戻ります。運用時にはこの状態で問題ありませんが、開発時に変更した内容が再起動時に保持されないのは不都合です。

podman コンテナ上での作業も同様に全て消えてしまうので、開発中は基本的に「図 6.22. podman 関連ファイルを eMMC 上に保存する」に示すコマンドを実行して、コンテナイメージやコンテナの中身などの podman 関連のファイルは eMMC 上に保存するよう設定しておきます。

```
[armadillo /]# podman_switch_storage --disk
```

図 6.22 podman 関連ファイルを eMMC 上に保存する

また、予め Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「VPU や NPU を使用する [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.4.0/ch08.html#sct.prepare-library]」の手順を行ってください。

6.3.2. サンプルコンテナをビルド

前述の通り、今回作成するサンプルアプリケーションは、アットマークテクノが提供しているサンプルコンテナイメージをベースに開発を行います。そのため、まずはそのサンプルコンテナイメージを生成します。

Armadillo-IoT ゲートウェイ G4 コンテナ [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/container]から「Debian 11 (bullseye) サンプル Dockerfile」をダウンロードしてください。[VERSION]は適宜読み替えてください。

```
[armadillo /]# wget https://armadillo.atmark-techno.com/files/downloads/armadillo-iot-g4/
container/at-debian-image-dockerfile-[VERSION].tar.gz
```

図 6.23 Dockerfile のダウンロード

「図 6.24. サンプルコンテナのイメージのビルド」に示すコマンドを実行することで、アットマークテクノが提供しているサンプルコンテナイメージを Armadillo-IoT ゲートウェイ G4 上でビルドできます。ここでは"abos-dev-guide"という名称で、"v0.0.0"というタグ名のコンテナを作成しています。

```
[armadillo /]# tar xzf at-debian-image-dockerfile-[VERSION].tar.gz
[armadillo /]# cd at-debian-image-dockerfile-[VERSION]
[armadillo /]# podman build -t abos-dev-guide:v0.0.0 .
:（省略）
[armadillo /]# podman images abos-dev-guide
REPOSITORY                TAG         IMAGE ID      CREATED             SIZE
localhost/abos-dev-guide  v0.0.0      c38bd7464cd8  About a minute ago  234 MB
```

図 6.24 サンプルコンテナのイメージのビルド

6.3.3. コンテナ内に入る

作成したコンテナの中に入り、開発作業を行います。

「図 6.25. コンテナ内に入る」に示すコマンドを実行することで、コンテナの中に入ることができます。この時、後にサンプルアプリケーションを動作させるために必要になるオプションも指定しておきます。

```
[armadillo /]# podman run -ti ¥
        --name=sample_container ¥
        --env=XDG_RUNTIME_DIR=/tmp ¥
        --env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu ¥
        --device=/dev/tty1 ¥
```

```
        --device=/dev/input ¥
        --device=/dev/dri ¥
        --device=/dev/galcore ¥
        --device=/dev/mxc_hantro ¥
        --device=/dev/mxc_hantro_vc8000e ¥
        --device=/dev/ion ¥
        --volume=/run/udev:/run/udev:ro ¥
        --volume=/opt/firmware:/opt/firmware:ro ¥
        --cap-add "SYS_TTY_CONFIG" ¥
        localhost/abos-dev-guide:v0.0.0 /bin/bash
[container /]#
```

図 6.25 コンテナ内に入る

6.4. アプリケーション実行の準備をする

アットマークテクノが提供しているサンプルコンテナには、必要最低限のパッケージしかインストールされていません。このままでは、サンプルアプリケーションのダウンロードや実行ができませんので、最初に必要なパッケージをインストールしておきます。

```
[container /]# apt update
[container /]# apt install -y ¥
   wget ¥
   procps ¥
   python3 ¥
   python3-pip ¥
   python3-opencv ¥
   python3-numpy ¥
   python3-tflite-runtime ¥
   nn-imx ¥
   gstreamer1.0-tools ¥
   gstreamer1.0-plugins-bad ¥
   gstreamer1.0-plugins-good ¥
   gstreamer1.0-imx
[container /]# pip3 install pillow
```

図 6.26 コンテナへパッケージをインストール

6.5. アプリケーションを作成する

podman コンテナ内で実行するアプリケーションを作成します。

以下では開発の際の手順の一例を紹介しますが、サンプルアプリケーションは こちら [https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/read_meter.tar.gz]から作成済みのものをダウンロードしてご使用ください。

コンテナ内で「図 6.27. サンプルアプリケーションのダウンロードと展開」に示すコマンドを実行することで、サンプルアプリケーションをダウンロード・展開できます。今回は、/opt 以下にアプリケーションファイルを展開します。

```
[container /]# cd /opt
[container /opt]# wget https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/read_meter.tar.gz
```

```
[container /opt]# tar xzf read_meter.tar.gz
[container /opt]# rm read_meter.tar.gz
```

図 6.27 サンプルアプリケーションのダウンロードと展開

6.5.1. ファイル構成

ダウンロードしたファイルを展開すると、read_meter というディレクトリが現れます。

アプリケーションを構成するファイルは全て read_meter ディレクトリに配置されています。配置した各ファイルと、その役割を「図 6.28. サンプルアプリケーションを構成するファイル」に示します。

```
read_meter
|-- gauge_sample.mp4 ❶
|-- model.tflite ❷
|-- modules
|   |-- __init__.py
|   |-- inferencer.py ❸
|   `-- meter.py ❹
|-- parameter_sample.json ❺
`-- read_meter ❻
```

図 6.28 サンプルアプリケーションを構成するファイル

❶ アナログメーターが映り続けた動画です。USB カメラの代わりに入力映像として使用できます。

❷ TFLite 形式の推論モデルファイルです。

❸ 物体検出に関わる処理を記した Python モジュールファイルです。

❹ メーター読み取りに関わる処理を記した Python モジュールファイルです。

❺ 読み取るアナログメーターの中心座標や、各スケールの座標などを記録したファイルです。

❻ アプリケーション本体です。

6.5.2. コンテナ内のファイルの編集

実際に Armadillo で動かすソフトウェア開発を行う場合、ATDE 等の他の環境で作成したアプリケーションを Armadillo のコンテナ内に転送して動かすことが一般的です。

転送方法は様々ありますが、開発時に ATDE と Armadillo が同一ネットワーク内にあれば ATDE がデフォルトで起動しているウェブサーバ機能を使用する方法が手軽です。

```
[ATDE ~/]$ sudo cp file /var/www/html

: (以下 Armadillo 内のコンテナ内での作業)
[container /]# wget http://[ATDE の IP アドレス]/file
[container /]# ls file
file
```

図 6.29 ATDE から Armadillo へファイルを送信

もちろん Armadillo 内でファイルの編集をすることも可能です。アットマークテクノが提供しているサンプルコンテナイメージには、デフォルトではエディタが含まれていないので、「図 6.30. コンテナ内にエディタをインストールする」に示すコマンドを実行することでお好きなエディタ(この例では vim)をインストールできます。

```
[container /]# apt update && apt upgrade
[container /]# apt install -y vim
```

図 6.30 コンテナ内にエディタをインストールする

ただし、基本的にはエディタはアプリケーションを実行するだけの本番環境には必要ないため、最終的なコンテナイメージのサイズ削減のためにも、開発完了時にはアンインストールするか別途実行用のコンテナを作成することをお勧めします。

6.5.3. アプリケーションの動作確認

コンテナ内にアプリケーションを配置できたら動作確認をしてみましょう。予め Armadillo-IoT ゲートウェイ G4 を HDMI ケーブルでモニタと接続しておいてください。

今回はサンプルアプリケーションの動作確認として「図 6.31. サンプルアプリケーションの動作確認」に示すコマンドを実行します。ここでは USB カメラは使用せず、gauge_sample.mp4 を映像の入力として動作確認をします。

```
[container /]# cd /opt/read_meter
[container /opt/read_meter]# weston --tty=1 &
[container /opt/read_meter]# ./read_meter --model model.tflite --param_file parameter_sample.json ⏎
--video gauge_sample.mp4
```

図 6.31 サンプルアプリケーションの動作確認

「図 6.32. Armadillo-IoT ゲートウェイ G4 への周辺機器の接続」のような画面が表示されれば正しく動作しています。

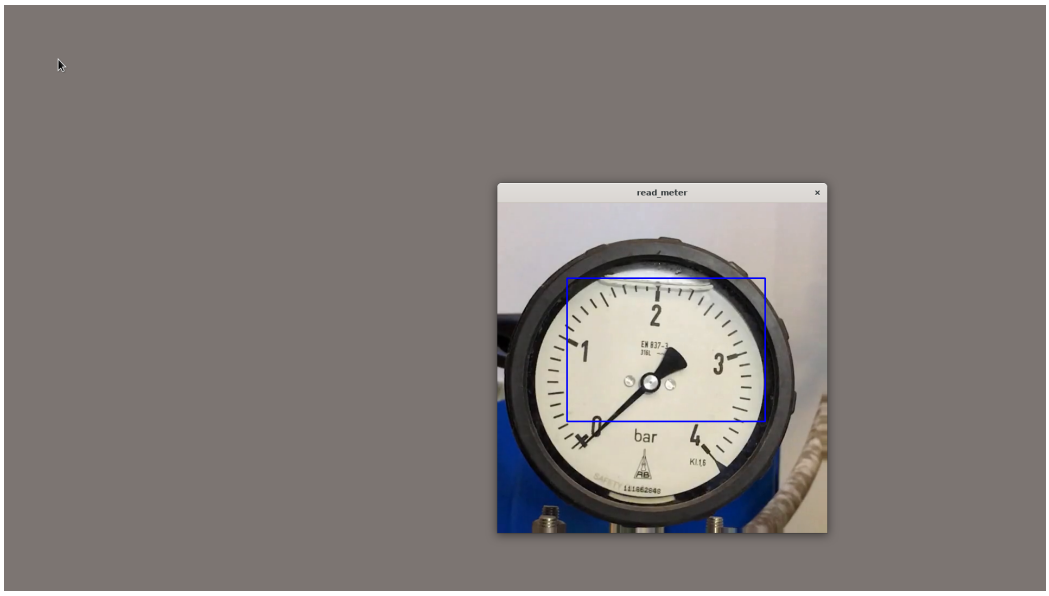


図 6.32 Armadillo-IoT ゲートウェイ G4 への周辺機器の接続

Ctrl+C でサンプルアプリケーションを終了させます。動作確認が完了したら、weston は終了させます。

```
[container /opt/read_meter]# pkill weston //バックグラウンドの weston を終了
```

図 6.33 weston を終了

以上でアプリケーションの作成は完了です。

6.5.4. podman コンテナの保存

コンテナ内の編集が完了しても、この後にコンテナを停止したり Armadillo の電源を OFF にしたりすると、ここまで編集してきたコンテナ内の変更の差分がなくなってしまいます。そのため、podman コンテナ内の整備が完了したらコンテナ内の変更を保存する必要があります。

まず、 exit コマンドを実行してコンテナから抜けます。

```
[container /]# exit
[armadillo ~]# //コンテナから抜けた
```

図 6.34 podman コンテナから抜ける

「図 6.34. podman コンテナから抜ける」の方法でコンテナから抜けると、コンテナは停止します。

起動中のコンテナは「図 6.35. 起動中の podman コンテナを確認する」に示すコマンドで確認できます。

```
[armadillo ~]# podman ps
CONTAINER ID  IMAGE                                 COMMAND     CREATED        STATUS                 ⏎
```

```
PORTS       NAMES
//sample_container が起動中でないことを確認
```

図 6.35 起動中の podman コンテナを確認する

停止中のコンテナは「図 6.36. 停止中の podman コンテナを確認する」に示すコマンドで確認できます。

```
[armadillo ~]# podman ps -a
CONTAINER ID  IMAGE                              COMMAND     CREATED        STATUS                        ⏎
PORTS       NAMES
9c908ab45ed8  localhost/abos-dev-guide:v0.0.0  /bin/bash   3 minutes ago  Exited (0) 12 seconds         ⏎
ago              sample_container
```

図 6.36 停止中の podman コンテナを確認する

次に、 podman commit コマンドで先程までの変更内容を保存します。「図 6.37. podman コンテナの変更を保存する」は、変更後の"sample_container"を"abos-dev-guide"というイメージ名、"v1.0.0"というタグ名で保存する例です。

```
[armadillo ~]# podman commit sample_container abos-dev-guide:v1.0.0
[armadillo ~]# podman images abos-dev-guide //保存されたことを確認
REPOSITORY                TAG         IMAGE ID      CREATED             SIZE
localhost/abos-dev-guide  v1.0.0      d0de8c2e70f0  About a minute ago  901 MB
localhost/abos-dev-guide  v0.0.0      c38bd7464cd8  2 hours ago         234 MB
```

図 6.37 podman コンテナの変更を保存する

タグ名の部分でコンテナイメージのバージョン管理を行うことが可能です。

podman commit することで、コンテナ内で行った変更が eMMC 内に保存され、Armadillo 本体の電源を切ったり、コンテナを削除しても commit した箇所までが保持されます。開発時に Armadiilo の電源を切る場合は、commit を行ったことを確認してから電源を切ってください。

6.5.5. podman コンテナのエクスポート

Armadillo-IoT ゲートウェイ G4 上で作成した podman コンテナイメージは、他の Armadillo-IoT ゲートウェイ G4 上でも動作します。作成した podman コンテナイメージを他の Armadillo に書き込む方法として以下があります。

1. podman コンテナイメージを外部ストレージなどに保存する
2. dockerhub などのリモートリポジトリに保存する

以下では、作成したコンテナイメージを USB メモリに保存する方法を紹介します。予め Armadillo に 4GB 以上の空き容量がある USB メモリを挿入してください。

「図 6.38. podman コンテナイメージを外部ストレージに出力する」は、USB メモリの第 1 パーティション(/dev/sda1)に abos-dev-guide.tar という名前で abos-dev-guide の v1.0.0 を保存する例です。

```
[armadillo ~]# mount /dev/sda1 /mnt ❶
[armadillo ~]# podman save -o /mnt/abos-dev-guide.tar abos-dev-guide:v1.0.0 ❷
```

図 6.38 podman コンテナイメージを外部ストレージに出力する

❶ USB メモリなどの外部記憶デバイスをマウントします。

❷ USB メモリにイメージ名:abos-dev-guide、タグ名:v1.0.0 を abos-dev-guide.tar で保存します。

保存したコンテナイメージは、SWUpdate の swu パッケージに含ませたり、 `podman load` コマンドで読み込んだりすることで、他の Armadillo 上で動かすことができます。

6.5.6. podman コンテナとアプリケーションの自動実行

最後に、Armadillo-IoT ゲートウェイ G4 の起動時に、作成した podman コンテナ及びコンテナ内の自作アプリケーションが自動実行するように設定します。

Armadillo Base OS では、起動時に `podman_start` というスクリプトを実行して、/etc/atmark/containers/以下の.conf ファイルを参照し、そこで指定されているコンテナを起動します。

各種設定方法について、詳しくは製品マニュアルをご確認ください。以下では、本サンプルアプリケーション向けの設定を示します。

6.5.6.1. conf ファイルの作成

コンテナの外で/etc/atmark/containers/sample_container.conf を作成します。

```
image="localhost/abos-dev-guide:v1.0.0"
devices="/dev/tty1 /dev/input /dev/dri /dev/galcore /dev/mxc_hantro /dev/mxc_hantro_vc8000e /dev/ ⏎
ion"
volumes="/run/udev:/run/udev:ro /opt/firmware:/opt/firmware"
readonly=false
append_args --env=XDG_RUNTIME_DIR=/tmp
append_args --env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu
append_args --cap-add=SYS_TTY_CONFIG
append_args -w /opt/read_meter
set_command /bin/bash -c "
weston --tty=1 & ¥
python3 read_meter ¥
        --model model.tflite ¥
        --param_file parameter_sample.json ¥
        --video gauge_sample.mp4"
```

図 6.39 sample_container.conf 作成例

なお、「図 6.39. sample_container.conf 作成例」に示した conf ファイルは こちら [https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/sample_container.conf]からダウンロード可能です。

```
[armadillo /]# wget https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/sample_container.conf -O /etc/atmark/containers/sample_container.conf
```

図 6.40 sample_container.conf のダウンロード

6.5.6.2. 設定の確認

「6.5.6.1. conf ファイルの作成」で作成した conf ファイルの設定が正しく出来ているかを確認します。

「図 6.41. podman_start の実行例」を実行すると、/etc/atmark/containers/sample_container.conf ファイルの内容に従ってコンテナが起動します。これにより正常にコンテナとサンプルアプリケーションが起動することを確認します。

```
[armadillo ~]# podman_start sample_container
```

図 6.41 podman_start の実行例

正常にサンプルアプリケーションが実行されると、「6.5.3. アプリケーションの動作確認」で動作確認した時と同様の画面が表示されます。

ここまでの動作確認が完了したら、/etc/atmark/containers/sample.conf を `persist_file` コマンドで永続化します。

```
[armadillo ~]# persist_file /etc/atmark/containers/sample_container.conf
```

図 6.42 sample_container.conf の永続化

「図 6.42. sample_container.conf の永続化」を実行することにより、作成した sample_container.conf が再起動後も eMMC 内に残り、次回以降 Armadillo に電源を投入した際にコンテナが自動起動します。

`podman_start` によって起動したコンテナも「図 6.43. コンテナの停止・削除」に示すコマンドで停止、削除できます。

```
[armadillo ~]# podman stop sample_container ❶
[armadillo ~]# podman rm sample_container ❷
```

図 6.43 コンテナの停止・削除

❶ sample_container を停止します。

❷ sample_container を削除します。

6.6. 動作確認

最後に動作確認を行います。

周辺機器の接続後、Armadillo-IoT ゲートウェイ G4 に電源を投入してください。

しばらくすると自動的にアプリケーションが起動して、サンプルの動画ウィンドウを HDMI モニタに映し出します。青い四角は、物体検出にてアナログメーターだと検知した箇所です。

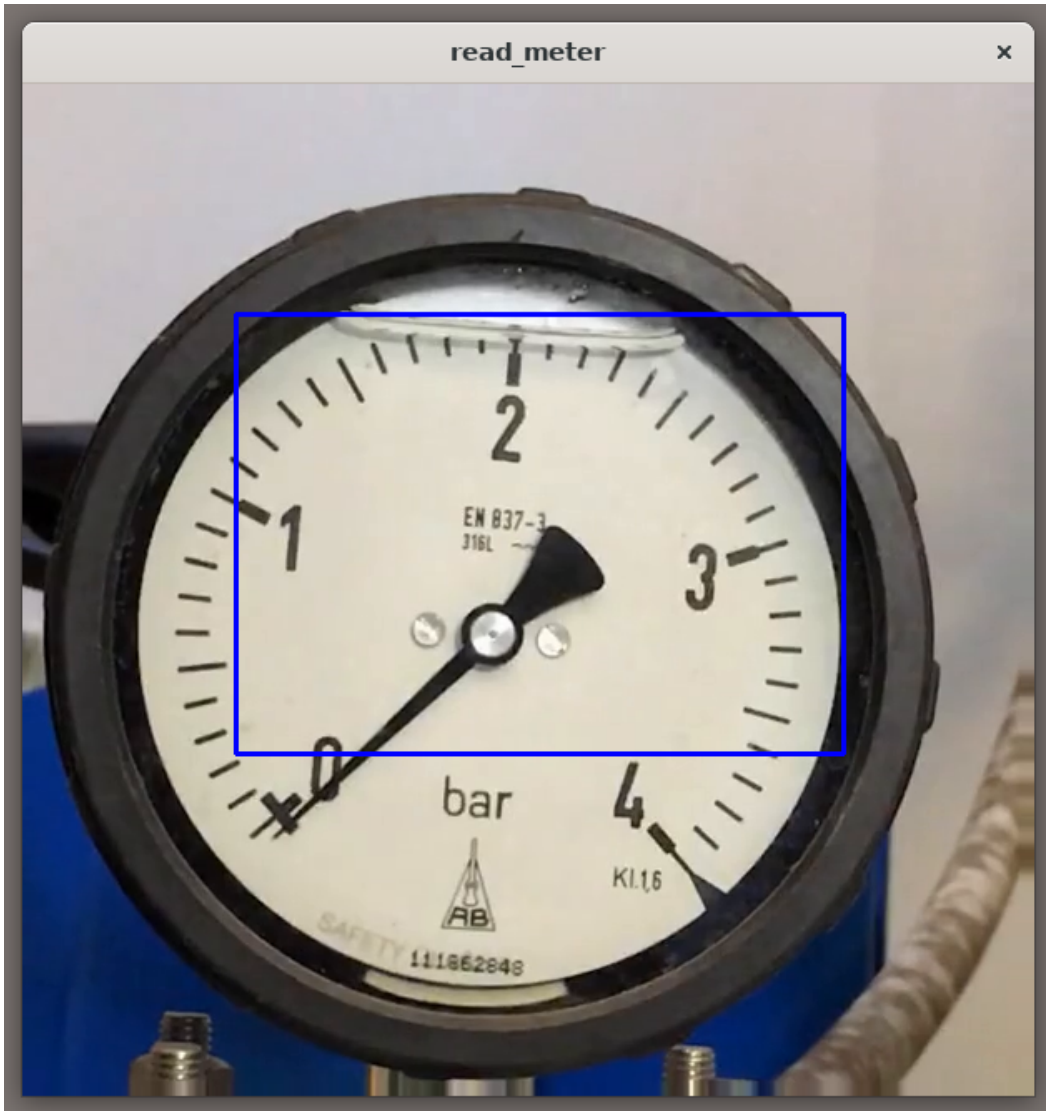


図 6.44 サンプル動画に対して物体検出

この状態でしばらく待つと、アナログメーターが指す値が変わっていき、スケールの最小値である 0 を超えた辺りから青い四角の上に読み取れた数値が表示されます。

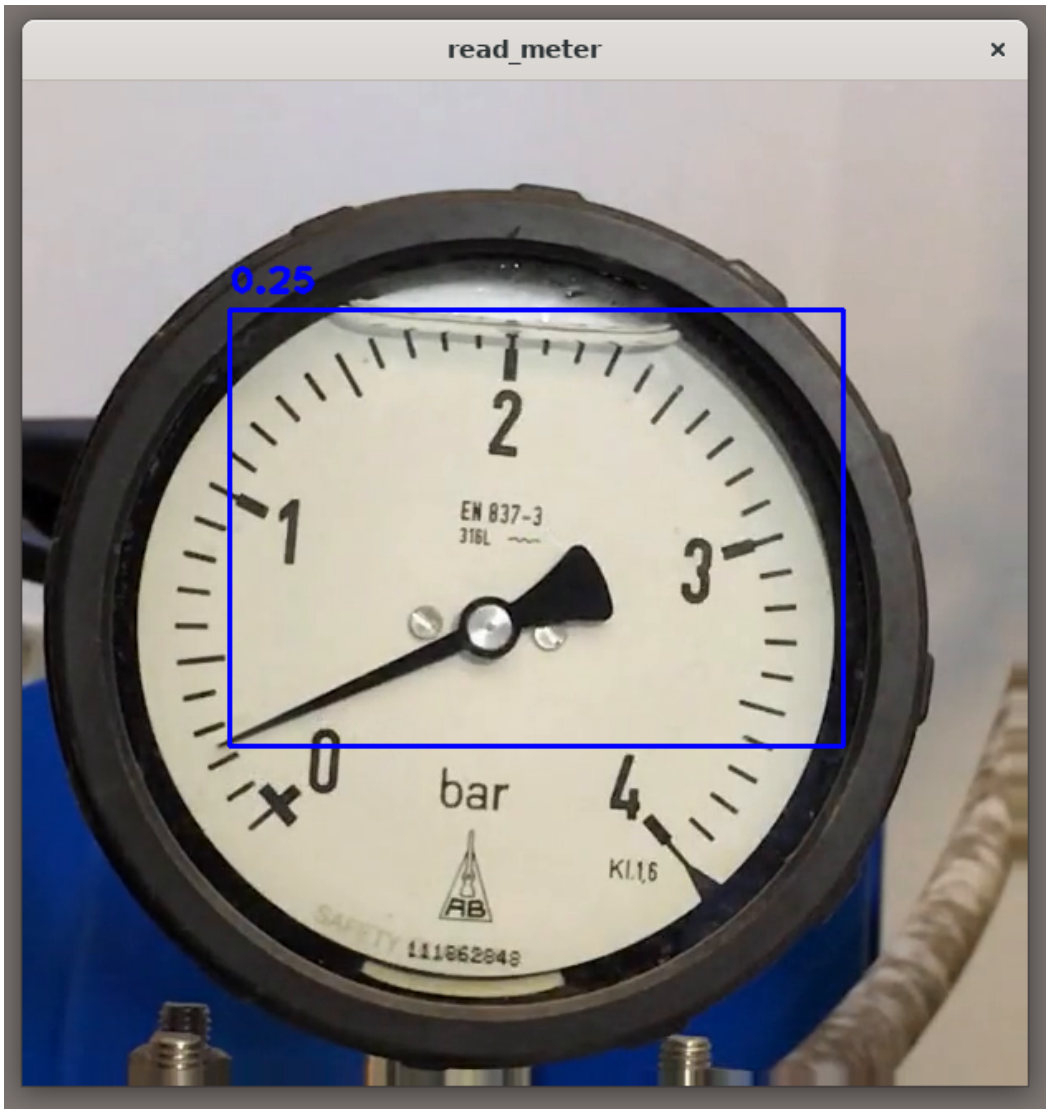


図 6.45 サンプル動画内のアナログメーターを読む

動画が終了すると、アプリケーション及びコンテナが停止します。

ここまでを確認できたならば、本ドキュメントにおけるサンプルアプリケーションは完成です。

6.7. USB カメラから映像を取得するように変更

以降の手順は USB カメラを使用するオプションです。

ここまでで作成したアプリケーションは、サンプル動画を映像入力としてメーターの読み取りを行っています。USB カメラと撮影対象のアナログメーターをお持ちの場合は、USB カメラから映像を取得し、任意のアナログメーターの値を読むことも可能です。

sample_container が起動している場合は、「図 6.43. コンテナの停止・削除」のコマンドを実行して停止・削除しておいてください。

6.7.1. 撮影対象の parameter.json を作成

本サンプルアプリケーションでは、アナログメーターの値を読む為に、各アナログメーターの中心とスケールがなす角度などのパラメータを保存した json ファイルが必要になります。サンプル動画以外のアナログメーターに対応させるためには、撮影対象となるアナログメーターの中心と各スケールの座標等を示す parameter.json を作成し、コンテナ内のアプリケーションから読み出す必要があります。

ここでは、サンプルアプリケーション向けのパラメータファイルを作成するための Python スクリプトについて紹介します。

アナログメーターを正面から捉えた画像から各点の座標を取得できる Python スクリプト(config_param.py)を こちら [https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/config_param.py]からダウンロードできます。

「図 6.46. config_param.py のダウンロードと起動」に、image.png をアナログメーターを正面から捉えた画像とした時の、config_param.py の実行例を示します。

```
[ATDE ~/]$ wget https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/config_param.py
[ATDE ~/]$ python3 config_param.py --image image.png
```

図 6.46 config_param.py のダウンロードと起動

図 6.47 アナログメーターを正面から捉えたサンプル画像(image.png)

実行すると以下のようなウィンドウが出現するので、アナログメーターの中心をクリックしてください。

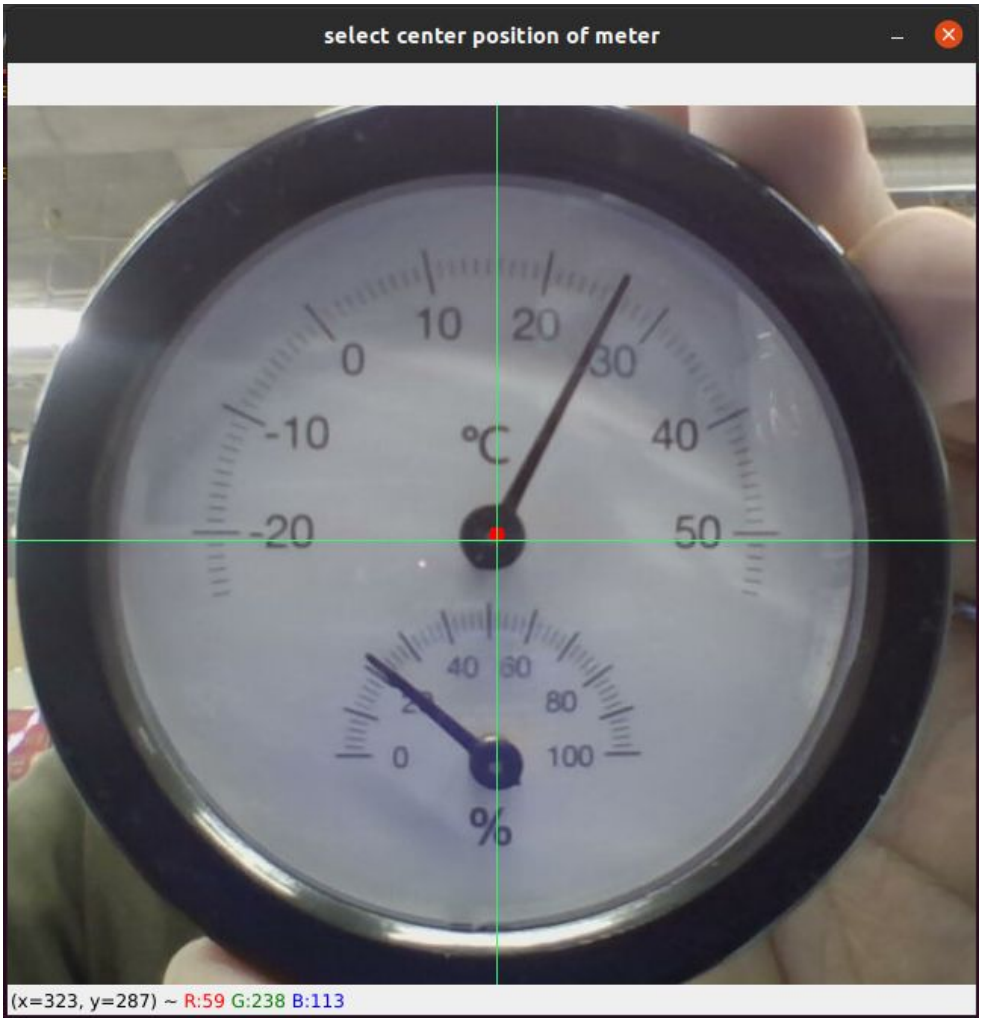


図 6.48 アナログメーターの中心をクリック

選択された中心点は赤い点で描画され、中心の選択は何度もやり直すことができます。選択が完了したら、何かキーを押下してください。

次に、以下のようなウィンドウが出現するので、アナログメーターが示すことができる最低値から順に、スケールの位置をクリックしてください。

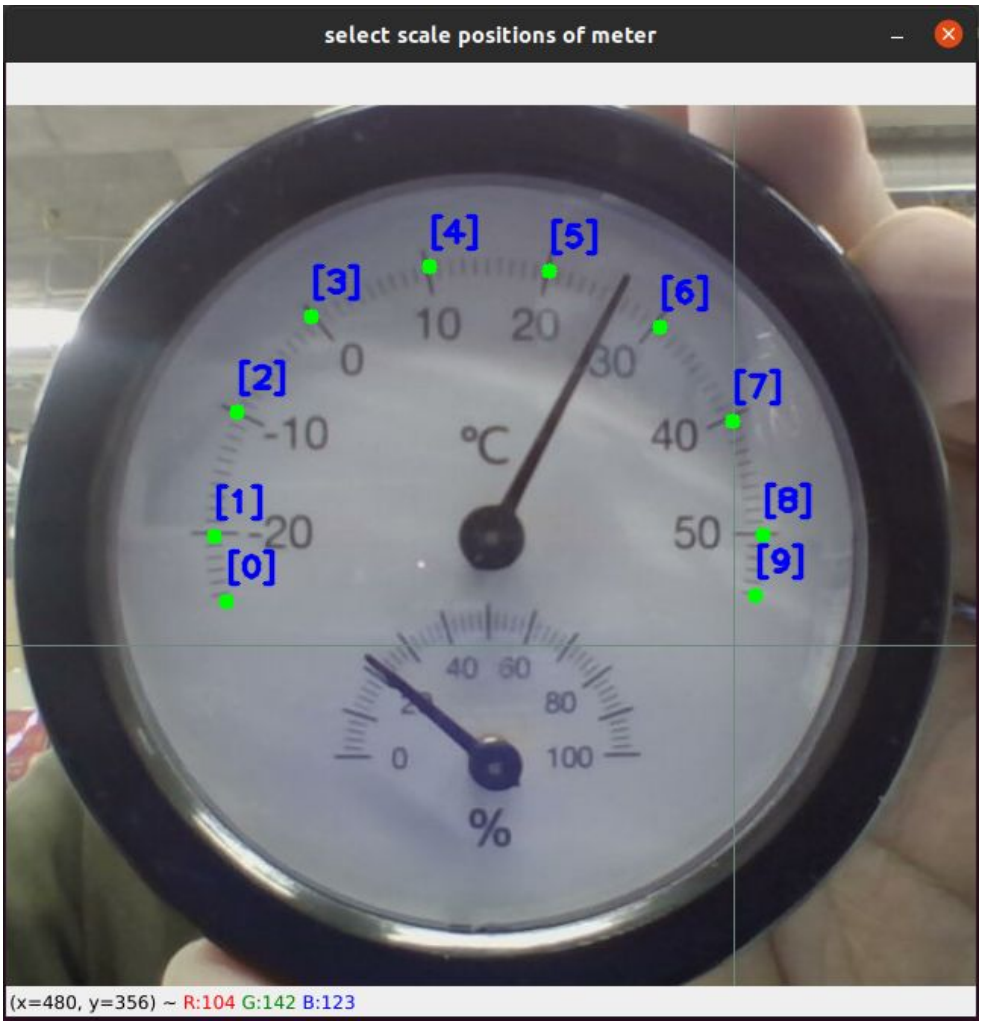


図 6.49 各スケールの位置をクリック

クリックした位置と順番は画面に描画されます。右クリックで一個前の選択を取り消すことができます。選択が完了したら、何かキーを押下してください。

次に、コンソールでクリックした位置の値を入力してください。コンソールで表示されている x 個目と画面内の選択点の数値は対応しています。入力したら Enter キーを押下し、選択した点の数だけこれを繰り返してください。

```
0 個目の数値> -25
1 個目の数値> -20
2 個目の数値> -10
3 個目の数値> 0
4 個目の数値> 10
5 個目の数値> 20
6 個目の数値> 30
7 個目の数値> 40
8 個目の数値> 50
9 個目の数値> 55
Done!
```

図 6.50 各スケールの数値を入力

最後まで入力すると、「Done!」と表示され、parameter.json が出力されています。

```
[ATDE ~/]$ ls parameter.json
parameter.json
```

図 6.51 各スケールの数値を入力

出力された parameter.json は、任意の方法で Armadillo に転送してください。

6.7.2. パラメータファイルの配置

作成したパラメータファイルを、コンテナ内のアプリケーションに読み込ませます。

読み取りたいアナログメーター 1 つにつき、パラメータファイルを 1 つ用意する都合上、何度も書き換えることになりそうなファイルです。このパラメータファイルをコンテナイメージの中に含めるとすると、ファイルの差し替え時には毎回コンテナイメージを作り直す必要があり手間です。

このようなファイルはコンテナイメージ内に含めず、Armadillo Base OS が提供するデータ保存用ディレクトリに保存し、コンテナに volume として渡す方法がお勧めです。

データ保存用ディレクトリは、/var/app/volumes と/var/app/rollback/volumes の 2 種類が用意されています。それぞれの詳細については、Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「データを保存する [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.4.0/ch08.html#sct.data-hozon-container]」を参照してください。

今回は/var/app/rollback/volumes を使用します。

まず、コンテナの外で「図 6.52. /var/app/rollback/volumes/assets の作成」に示すコマンドで、/var/app/rollback/volumes/の下に assets ディレクトリを作成します。

```
[armadillo ~]# mkdir -p /var/app/rollback/volumes/assets
```

図 6.52 /var/app/rollback/volumes/assets の作成

次に、作成したパラメータファイル(parameter.json)を/var/app/rollback/volumes/assets の中に配置します。

```
[armadillo ~]# cp /path/to/parameter.json /var/app/rollback/volumes/assets
```

図 6.53 parameter.json を配置

以上でパラメータファイルの配置は完了です。

なお、データ保存用ディレクトリである/var/app/volumes 及び、/var/app/rollback/volumes に配置したファイルは、eMMC 上に保存されるため再起動後も消えずに残り続けます。

6.7.3. .conf ファイルの設定

以下の 2 点に対応させるために、/etc/atmark/containers/sample_container.conf を「図 6.54. USB カメラを使用する sample_container.conf 作成例」のように編集します。ほとんど USB カメラ対応前と同じですが、異なる部分のみ注釈で解説します。

1. サンプルアプリケーションの映像入力を動画ではなく USB カメラにする
2. 「6.7.2. パラメータファイルの配置」で配置した parameter.json をコンテナ内で使用する

```
image="localhost/abos-dev-guide:v1.0.0"
devices="/dev/tty1 /dev/input /dev/dri /dev/galcore /dev/mxc_hantro /dev/mxc_hantro_vc8000e /dev/ ⏎
ion /dev/video*" ❶
volumes="/run/udev:/run/udev:ro /opt/firmware:/opt/firmware assets:/assets:ro" ❷
readonly=false
append_args --env=XDG_RUNTIME_DIR=/tmp
append_args --env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu
append_args --cap-add=SYS_TTY_CONFIG
append_args -w /opt/read_meter
set_command /bin/bash -c "
weston --tty=1 & ¥
python3 read_meter ¥
        --model model.tflite ¥
        --param_file /assets/parameter.json ¥ ❸
        --camera 3" ❹
```

図 6.54 USB カメラを使用する sample_container.conf 作成例

❶ USB カメラを使用するために/dev/video*を追加します。

❷ 作成した assets ディレクトリをコンテナに渡します。

❸ パラメータファイルはコンテナ内では/assets/parameter.json に配置されていますので、そこを指定します。

❹ "--video…" の行を削除し、"--camera 3" に修正します。

volumes に相対パスを指定すると、/var/app/rollback/volumes 以下のディレクトリを指定したことになります。

サンプルアプリケーションは、起動時のオプションに"--camera <デバイス番号>"を指定することで、/dev/video<デバイス番号>のカメラデバイスから映像を取得します。<デバイス番号>は、USB カメラ接続時のログなどから判断してください。

6.7.4. 自動実行の確認

podman_start コマンドでコンテナが起動することを確認します。

```
[armadillo /]# podman_start sample_container
```

USB カメラから映像とアナログメーターの値を取得できることを確認します。

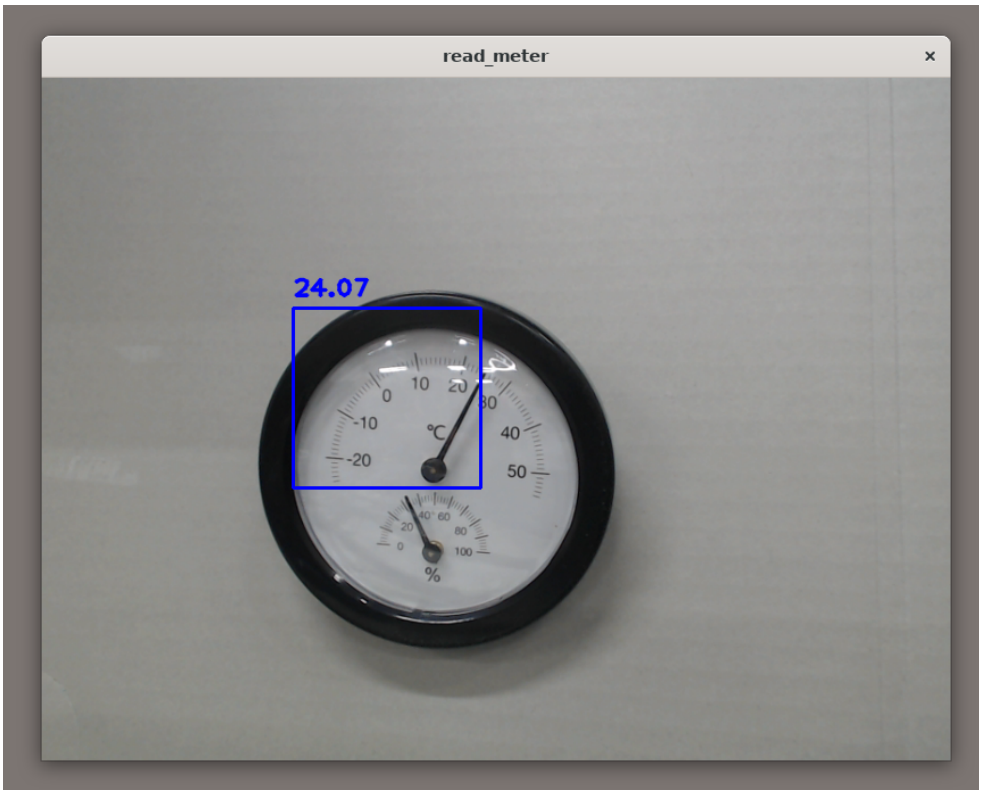


図 6.55 USB カメラの映像からアナログメーターの値を取得

次回 Armadillo 起動時に自動実行させるには、/etc/atmark/containers/sample_container.conf を `persist_file` コマンドで永続化させます。

```
[armadillo ~]# persist_file /etc/atmark/containers/sample_container.conf
```

図 6.56 sample_container.conf の永続化

7. Appendix

この章では、サンプルアプリケーション開発内で扱いきれなかった、Armadillo Base OS での開発に関わる情報を紹介します。

7.1. SWUpdate を用いてソフトウェアをアップデートする

Armadillo Base OS ではソフトウェアのアップデート方法として SWUpdate を利用できます。SWUpdate について詳細は Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「Armadillo のソフトウェアをアップデートする [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.4.0/ch09.html#ch.yakushima-softwareupdate]」を参照してしてください。

今回の例では USB メモリを使用してアップデートしますので、4GB 以上の USB メモリを用意してください。

7.1.1. SWUpdate による初回アップデート

SWUpdate では、ソフトウェアアップデートを行うために swu イメージというファイルを使用しますので、まずは開発用 PC(本ドキュメントでは ATDE)で swu イメージを作成します。

出荷時の Armadillo は署名されていない swu イメージでも SWUpdate を行うようになっています。そのため、まずはじめに自分専用の署名鍵を生成して公開鍵を Armadillo に配置する作業を SWUpdate を用いて行うことで、自分が作成した swu イメージでしかアップデートを行わない Armadillo を作成します。

使用している Armadillo に、過去に一度でもこの初回アップデート作業を行っている場合は二度同じ作業を行うことはできず、する必要もありません。

1. mkswu の取得

   初めに、その swu イメージを作成するソフトウェアである mkswu をインストールします。

```
[ATDE ~/]$ sudo apt update && sudo apt install mkswu
```

図 7.1 mkswu の取得

2. mkswu の初期設定

   「図 7.2. mkswu の初期設定を行う」に示すコマンドを実行することで、swu パッケージの署名に用いる鍵や、公開鍵を Armadillo に書き込むための swu イメージを作成できます。

```
[ATDE ~/]$ mkswu --init
mkdir: ディレクトリ '/home/atmark/mkswu' を作成しました
設定ファイルを更新しました：/home/atmark/mkswu/mkswu.conf
証明書のコモンネーム(一般名)を入力してください: abos-guide-sample ❶
証明書の鍵のパスワードを入力ください（4-1024 文字） ❷
証明書の鍵のパスワード（確認）:
Generating an EC private key
```

```
 -----
writing new private key to '/home/atmark/mkswu/swupdate.key'
アップデートイメージを暗号化しますか？ (N/y) y ❸
/home/atmark/mkswu/swupdate.aes-key を作成しました。
アットマークテクノが作成したイメージをインストール可能にしますか？ (Y/n) ❹
root パスワード: ❺
root パスワード（確認）:
atmark ユーザのパスワード（空の場合は root パスワードを使います）: ❻
atmark ユーザのパスワード（確認）:
BaseOS イメージの armadillo.atmark-techno.com サーバーからの自動アップデートを行いますか？ (y/N) ❼
/home/atmark/mkswu/initial_setup.swu を作成しました。

"/home/atmark/mkswu/initial_setup.swu"をそのまま使うことができますが、モジュールを追加してイメージを再構築する場合は
次のコマンドで作成してください： mkswu "/home/atmark/mkswu/initial_setup.desc"
other_desc_files

インストール後は、このディレクトリを削除しないように注意してください。
鍵を失うと新たなアップデートはデバイスの /etc/swupdate.pem
を修正しないとインストールできなくなります。

[ATDE ~/]$ ls ~/mkswu ❽
initial_setup.desc  initial_setup.swu  mkswu.conf  swupdate.aes-key  swupdate.key  swupdate.pem
```

図 7.2 mkswu の初期設定を行う

❶ 会社や製品が分かるような任意の名称を指定します(今回は abos-guide-sample)。

❷ 証明鍵を保護するパスフレーズを 2 回入力します。

❸ swu イメージ自体を暗号化する場合に「y」を選択します。

❹ アットマークテクノが提供するアップデートイメージをインストールできるようにするかを選びます。例では有効にしています。

❺ ここで Armadillo の root ユーザーのパスワードを設定します。2 回入力してください。

❻ atmark ユーザーのパスワードも同様に設定します。2 回入力してください。

❼ Armadillo Base OS の定期アップデートの有効/無効を設定します。例では無効にしています。

❽ 初期化処理の結果生成されたファイルを確認しています。swupdate.aes-key は swu イメージの暗号化を行うと作成されます。

3. SWUpdate の実行

上記の手順で生成された初回セットアップ用の swu イメージ(initial_setup.swu)を Armadillo に書き込みます。

ATDE に USB メモリを接続し、作成した swu イメージを配置します。

```
[ATDE ~/]$ df -h
Filesystem                Size  Used Avail Use% Mounted on
:（省略）
```

```
/dev/sdb1                  15G   24K   14G    1% /media/atmark/USBDRIVE ❶

[ATDE ~/]$ cp ~/mkswu/initial_setup.swu /media/atmark/USBDRIVE/ ❷
[ATDE ~/]$ umount /media/atmark/USBDRIVE/ ❸
```

図 7.3 swu イメージの配置

❶ USB メモリがマウントされている場所を確認します。

❷ ファイルをコピーします。

❸ /media/atmark/USBDRIVE をアンマウントします。コマンド終了後に USB メモリを取り外してください。

swu イメージを配置した USB メモリを動作中の Armadillo に挿入すると、自動的にアップデートが開始され、アップデートが完了すると再起動します。

再起動後には root ユーザ、atmark ユーザともに、「図 7.2. mkswu の初期設定を行う」で設定したパスワードでログインできます。

これでこの Armadillo に公開鍵が配置され、手元の秘密鍵で作成された swu イメージでアップデート可能になりました。

Armadillo に公開鍵を配置した後に対応する鍵や、 `mkswu.conf` を紛失すると、当該の Armadillo に SWUpdate を行うことができなくなりますのでご注意ください。

7.2. 作成したコンテナを他の Armadillo に組み込む

「6.5. アプリケーションを作成する」では、Armadillo 上でコンテナイメージを作成する方法を紹介してきましたが、他の Armadillo など外部で作成されたコンテナイメージを別な Armadillo に組み込む方法を紹介します。

7.2.1. podman load でコンテナイメージを組み込む

「6.5.5. podman コンテナのエクスポート」で紹介した通り、 `podman images` コマンドで表示されるコンテナイメージは `podman save` コマンドを使用することでファイルとして保存することができました。

このファイルは、 `podman load` コマンドを使用することで、別な Armadillo 上であってもコンテナイメージとしてインポートすることができます。サンプルアプリケーションのコンテナイメージを使用した実行例を「図 7.4. コンテナイメージファイルをインポートする」に示します。

コンテナイメージのサイズはそのイメージの内容によって異なりますが、1GB を超えることもよくあります。OverlayFS 上で保存できるファイルサイズでは保存しきれないことがありますので、この方法でコンテナイメージをインポートする際には、イメージファイルは外部の USB メモリや SD カードに配置しておくことをお勧めします。

```
[armadillo /]# podman_switch_storage --disk ❶
[armadillo /]# mount /dev/sda1 /mnt && cd /mnt ❷
[armadillo /mnt]# wget https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base- ⏎
```

```
os-dev-guide/abos-dev-guide-v1.0.0.tar ❸
[armadillo /mnt]# podman load -i abos-dev-guide-v1.0.0.tar ❹
[armadillo /mnt]# podman images abos-dev-guide ❺
REPOSITORY                TAG         IMAGE ID      CREATED       SIZE
localhost/abos-dev-guide  v1.0.0      05304c68979f  15 hours ago  917 MB
```

図 7.4 コンテナイメージファイルをインポートする

❶ podman コンテナイメージの保存先を eMMC に変更します。

❷ /dev/sda1 として認識されている USB メモリ(もしくは SD カード)を/mnt にマウントしディレクトリ移動します。

❸ USB メモリ上にサンプルアプリケーション用コンテナイメージをダウンロードします。

❹ podman load コマンドでコンテナイメージをインポートします。

❺ podman images コマンドでインポートできていることを確認します。

7.2.2. SWUpdate でコンテナイメージを組み込む

「6.5.5. podman コンテナのエクスポート」で紹介した、 podman save コマンドを用いて生成したコンテナイメージファイルは、SWUpdate によって他の Armadillo に適用することも可能です。以下では SWUpdate を用いてサンプルアプリケーションを含むコンテナイメージと、コンテナを自動実行するための sample_container.conf を Armadillo に書き込む手順を紹介します。

SWUpdate については「7.1. SWUpdate を用いてソフトウェアをアップデートする」を参照してください。以下の手順を実行するためには、予め「7.1.1. SWUpdate による初回アップデート」の手順を実行して SWUpdate の初回アップデートを実施しておく必要があります。

1. コンテナインストール用の swu イメージを作成する

   導入したいコンテナイメージを含んだ swu イメージを作成します。

   まず、~/mkswu/abos-dev-guide-descs ディレクトリを作成し、その中に Armadillo に組み込みたいコンテナイメージと、自動実行用の.conf ファイルを配置します。コンテナの自動実行については「6.5.6. podman コンテナとアプリケーションの自動実行」を参照してください。

   ```
   [ATDE ~/mkswu]$ mkdir -p abos-dev-guide-descs && cd abos-dev-guide-descs
   [ATDE ~/mkswu/abos-dev-guide-descs]$ wget https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/abos-dev-guide-v1.0.0.tar
   [ATDE ~/mkswu/abos-dev-guide-descs]$ wget https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/sample_container.conf
   ```

   図 7.5 作業用ディレクトリの作成と書き込むファイルのダウンロード

   次に、swu イメージ作成に用いる usb_container_sample.desc を以下のように作成します。

   ```
   [ATDE ~/mkswu/abos-dev-guide-descs]$ cat usb_container_sample.desc
   version=1
   ```

```
swdesc_usb_container "abos-dev-guide-v1.0.0.tar" ❶
swdesc_files --extra-os --dest /etc/atmark/containers sample_container.conf ❷
```

図 7.6 usb_container_sample.desc 作成例

❶ abos-dev-guide-v1.0.0.tar をコンテナイメージとして組み込みます。

❷ sample_container.conf を Armadillo 内の/etc/atmark/containers に配置します。

desc ファイルの詳細については Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「 mkswu の desc ファイル [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.4.0/ch09.html#sct.mkswu-desc]」を参照してください。/usr/share/mkswu/examples 以下の desc ファイルはサンプルですので、そちらも参考にしてください。

swu イメージを作成します。

```
[ATDE ~/mkswu/abos-dev-guide-descs]$ mkswu usb_container_sample.desc
Enter pass phrase for /home/atmark/mkswu/swupdate.key: ❶
以下のファイルを USB メモリにコピーしてください：
'/home/atmark/mkswu/abos-dev-guide-descs/usb_container_sample.swu' '/home/atmark/mkswu/
abos-dev-guide-descs/abos-dev-guide-v1.0.0.tar' '/home/atmark/mkswu/abos-dev-guide-
descs/.usb_container_sample/abos-dev-guide-v1.0.0.tar.sig'

usb_container_sample.swu を作成しました。
```

図 7.7 swu イメージの作成

❶ 「図 7.2. mkswu の初期設定を行う」で設定した証明書の鍵のパスワードを入力してください。

これでコンテナイメージをインストールし、Armadillo 起動時に自動実行するための swu イメージが完成しました。

2. USB メモリに swu イメージを書き込む

mkswu 実行時に表示された指示に従って、作成した swu イメージと付属のファイル群を USB メモリに配置します。ATDE に USB メモリを接続して以下を実行してください。

```
[ATDE ~/mkswu/abos-dev-guide-descs]$ df -h
Filesystem                Size  Used Avail Use% Mounted on
:（省略）
/dev/sdb1                 15G   24K   14G    1% /media/atmark/USBDRIVE ❶

[ATDE ~/mkswu/abos-dev-guide-descs]$ sudo cp usb_container_sample.swu ¥
abos-dev-guide-v1.0.0.tar ¥
.usb_container_sample/abos-dev-guide-v1.0.0.tar.sig ¥
/media/atmark/USBDRIVE/ ❷
[ATDE ~/mkswu/abos-dev-guide-descs]$ umount /media/atmark/USBDRIVE/ ❸
```

図 7.8 アップデート用ファイル群の配置

❶ USB メモリがマウントされている場所を確認します。

❷ ファイル群をコピーします。

❸ /media/atmark/USBDRIVE をアンマウントします。コマンド終了後に USB メモリを取り外してください。

3. SWUpdate の実行

   initial_setup と同様に、Armadillo に USB メモリを挿入すると自動的にアップデートが実行され、コンテナイメージの組み込みとコンテナの自動実行設定が行われます。その後 Armadillo が自動的に再起動し、サンプルアプリケーションが自動実行します。

   以上で、SWUpdate を用いて Armadillo にコンテナイメージと自動実行用の conf ファイルの配置ができました。他にも SWUpdate を用いて出来ることは多々ありますので、詳細は Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「 Armadillo のソフトウェアをアップデートする [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.4.0/ch09.html#ch.yakushima-softwareupdate]」を参照してください。

7.3. Device Tree を変更しハードウェアを拡張する

Armadillo の拡張インターフェースに外部デバイスを接続する際には、対象のピンの機能やパラメータを変更しなければならない場合があります。具体的には、Armadillo の各ピンの機能やパラメータを記述しているファイルである Device Tree Source を編集・ビルドし、Armadillo に書き込み、起動時にロードする必要があります。

以下では、Device Tree についての説明と、開発時及び製造・運用時それぞれの場合において Armadillo に Device Tree を書き込む手順について紹介します。具体的な Device Tree の記述方法については触れませんが、アットマークテクノが提供する Device Tree 作成ツールを紹介します。

7.3.1. Device Tree とは

Device Tree とは、ハードウェア情報を記述したデータ構造体であり、前述のハードウェア拡張時のパラメータを指定するものです。

Device Tree に対応しているメリットの 1 つは、ハードウェアの変更に対するソフトウェアの変更が容易になることです。例えば、Armadillo-IoT ゲートウェイ G4 では CON11(拡張インターフェース 1)及び、CON12(拡張インターフェース 2)に接続する拡張基板を作成した場合、主に C 言語で記述された Linux カーネルのソースコードを変更する必要はなく、やりたいことをより直感的に記述できる DTS(Device Tree Source)の変更で対応できます。

ただし、Device Tree は「データ構造体」であるため、ハードウェアの制御方法などの「処理」を記述することはできない点に注意してください。Device Tree には、CPU アーキテクチャ、RAM の容量、各種デバイスのベースアドレスや割り込み番号などのハードウェア構成情報のみが記述されます。

Device Tree のより詳細な情報については、Linux カーネルのソースコードに含まれているドキュメント(`Documentation/devicetree/`)、devicetree.org で公開されている「Device Tree Specification」を参照してください。

> DeviceTree: The Devicetree Specification
>
> https://www.devicetree.org/

7.3.2. Device Tree をカスタマイズする

DTS ファイルを編集し、コンパイルすることで DTB(Device Tree Blob)ファイルが生成されます。この DTB ファイルを Armadillo に書き込むことで、カスタマイズした Device Tree で Armadillo を起動することができます。

もちろんエディタで DTS ファイルを編集することも可能ですが、アットマークテクノが提供している at-dtweb というマウス操作で DTS 及び DTB を生成できるアプリケーションを使用するのが手軽でおすすめです。

at-dtweb を使用した Device Tree のカスタマイズ方法については、Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「Device Tree をカスタマイズする [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.4.0/ch09.html#ch.customize-dts] 」を参照してください。

7.3.3. 開発中の DTB ファイルの書き換え

Device Tree はその特性上、記述内容に誤りがあるとうまく接続デバイスが動作しない場合や、最悪 Armadillo が起動すらしない場合があります。最初から正しく動作する DTB ファイルを作成できるとは限りませんので、開発中は Armadillo の起動に使用する DTB ファイルを何度も書き換えなければならない場合があります。そのようなシステム開発中における DTB ファイルの書き換え方法の一例について紹介します。

1. 生成した DTB ファイルを Armadillo に転送

   at-dtweb などで生成した DTB ファイルを Armadillo に転送します。転送方法については USB メモリを使用してして転送したり、「図 6.29. ATDE から Armadillo へファイルを送信」で紹介しているようにウェブサーバ経由で転送したりするなど、お好きな方法で転送してください。

2. DTB ファイルの書き込み

   転送した DTB ファイルを所定のディレクトリ(/boot)に armadillo.dtb というファイル名で書き込むことで、Armadillo の起動時にその DTB ファイルを使用できるようになります。/boot/armadillo.dtb は最初から存在しますが、これは上書きしてしまって問題ありません。

   「図 7.9. DTB ファイルの配置」に示すコマンドを実行して、作成した DTB ファイル(以下の例では armadillo_iotg_g4-at-dtweb.dtb)を/boot/armadillo.dtb に上書きします。

```
[armadillo ~/]# ls armadillo_iotg_g4-at-dtweb.dtb
armadillo_iotg_g4-at-dtweb.dtb
[armadillo ~/]# rm -f /boot/armadillo.dtb
[armadillo ~/]# cp armadillo_iotg_g4-at-dtweb.dtb /boot/armadillo.dtb
[armadillo ~/]# persist_file /boot/armadillo.dtb
```

図 7.9 DTB ファイルの配置

   上記手順で DTB ファイルを書き込んだ後に Armadillo を再起動することで、次回以降は作成した DTB を使用して起動します。

7.3.3.1. 作成した DTB ファイルに差し替えて期待した動作をしなかった場合

ここまでの手順を実行して、作成した DTB ファイルを使用して起動する際に、DTS の記述に誤りがあり期待した動作をしない場合があります。誤った DTB ファイルに書き換えた場合の Armadillo の挙動として、以下の 3 つのパターンが考えられます。

1. 正常に起動するが、拡張したいデバイスや今まで動いていたデバイスを認識しない
2. Armadillo が正常に起動せず、Armadillo Base OS のロールバック機能が作動する
3. Armadillo は起動しているが、シリアルコンソールに何も表示されず、何も入力できない

上記のそれぞれのパターンについて対処法を紹介します。

1. 正常に起動するが、拡張したいデバイスや今まで動いていたデバイスを認識しない場合

   新規に DTS ファイルに追加した記述が誤っている、既存の記述を削除や上書きしてしまったなど、作成した DTS ファイルの記述に問題がある可能性が高いです。そのような場合は再度 at-dtweb などで DTS ファイルを修正し、新たに生成した DTB ファイルを Armadillo に転送する手順からやり直してください。

2. Armadillo が正常に起動せず、Armadillo Base OS のロールバック機能が作動する場合

   書き込んだ DTB ファイルが壊れている、DTS ファイル内の起動に関わる重要な箇所に誤りがあるなどの可能性が高いです。

   Armadillo Base OS は何らかの原因によってルートファイルシステムが破損し正常に起動できない場合に、前のバージョンに戻して再起動するロールバック機能が自動で働きます。これによって起動した Armadillo は作成した DTB ファイルを書き込む前の状態に戻っているので起動できるはずです。

   Armadillo Base OS のロールバック機能の詳細については、「 ロールバックの確認 [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.4.0/ch09.html#idm6035]」を参照してください。

   ロールバック機能が働いたかどうかは起動ログでも確認できますが、以下のコマンドで確認できます。

   ```
   [armadillo /]# abos-ctrl status
   Currently booted on /dev/mmcblk2p1
   WARNING: Currently running on non-latest version (expected /dev/mmcblk2p2 installed on Mon Apr  4 15:39:42 JST 2022)
   rollback-status: rolled back ❶
   ```

   ❶ rollback-status が rolled back となっているため、ロールバック機能が働いて起動したことがわかります。

   このような場合は再度 at-dtweb などで DTS ファイルを修正し、新たに生成した DTB ファイルを Armadillo に転送する手順までをやり直してください。修正した DTB ファイルを Armadillo へ転送後、今起動していない方のルートファイルシステムに修正後の DTB ファイルを配置します。

   ```
   [armadillo /]# ls armadillo_iotg_g4-at-dtweb.dtb
   armadillo_iotg_g4-at-dtweb.dtb
   [armadillo /]# abos-ctrl mount-old && mount -o remount,rw /target ❶
   [armadillo /]# cp armadillo_iotg_g4-at-dtweb.dtb /target/boot/armadillo.dtb ❷
   ```

   ❶ 起動していない方のルートファイルシステムを/target に読み書き可能モードでマウントします。

❷ armadillo_iotg_g4-at-dtweb.dtb を起動していない方のルートファイルシステムの/boot/armadillo.dtb に上書き保存します。

その後、以下のコマンドを実行し、現在起動していない方のルートファイルシステムを用いて起動します。

```
[armadillo /]# abos-ctrl rollback
[armadillo /]# reboot
```

修正した DTB が誤りがなければ正常に起動します。誤りが解消されなければ再度ロールバックしますので、再度同様の手順で復旧するまで DTB を修正してください。

3. Armadillo は起動しているが、シリアルコンソールに何も表示されず、何も入力できない場合

DTB 内のシリアルコンソールを設定する記述に誤りがあり、起動こそしていても何も入出力できない状態になっている可能性が高いです。

この場合は一度 Armadillo の電源を投入し直し、U-boot の環境変数を変更することで、Armadillo-IoT ゲートウェイ G4 の標準の DTB で起動するように設定します。

電源投入後、U-boot にて以下のコマンドを実行します。

```
U-Boot 2020.04-at6 (Mar 25 2022 - 08:09:26 +0000)
:(省略)
Hit any key to stop autoboot:  0 ❶
u-boot=> setenv fdt_file boot/armadillo_iot_g4.dtb ❷
u-boot=> boot ❸
```

❶ ここでカウントダウンが 0 になる前に何かキー入力をすることでプロンプトに入ります。

❷ Armadillo-IoT ゲートウェイ G4 の標準の DTB(armadillo_iot_g4.dtb)で起動するよう環境変数を設定します。

❸ 起動します。

起動後、再度 at-dtweb などで DTS ファイルを修正し、新たに生成した DTB ファイルを Armadillo に転送する手順からやり直してください。

次回以降の起動は、また/boot/armadillo.dtb を使用して起動するようになっています。

7.3.4. DTB 確定後の書き換え

「7.3.3. 開発中の DTB ファイルの書き換え」で紹介した方法はあくまでも開発中に何度も書き換える際にのみ推奨している手順です。最終的に期待した動作をする DTB ファイルが完成し、それを Armadillo に書き込む際には前述の手順は非推奨です。最終的に確定した DTB ファイルを Armadillo に書き込む際には、Armadillo Base OS にて提供されている SWUpdate を使用してください。SWUpdate については、「7.1. SWUpdate を用いてソフトウェアをアップデートする」を参照してください。

7.3.4.1. SWUpdate による初回アップデート

「7.1.1. SWUpdate による初回アップデート」の手順を実行し、自分専用の署名鍵で SWUpdate が実行できるようにしておきます。

7.3.4.2. DTB ファイルアップデート用の swu イメージを作成する

1. 作成した DTB ファイル配置用の swu イメージを作成する

   まず、適当な作業用ディレクトリ(以下の例では~/mkswu/update-dtb-descs)を作成し、その中に作成した DTB ファイルを armadillo.dtb にリネームして配置します。DTB ファイルへのパスは適宜読み替えてください。また、今回の swu イメージを作成時に使用するスクリプトファイルもあわせて配置します。

```
[ATDE ~/]$ cd mkswu
[ATDE ~/mkswu]$ mkdir -p update-dtb-descs && cd update-dtb-descs
[ATDE ~/mkswu/update-dtb-descs]$ cp /path/to/armadillo_iotg_g4-at-dtweb.dtb ./armadillo.dtb
[ATDE ~/mkswu/update-dtb-descs]$ cp /usr/share/mkswu/examples/update_preserve_files.sh ./
```

図 7.10 作業用ディレクトリの作成と各種ファイルの配置

次に、swu イメージ作成に用いる usb_dtb_sample.desc を以下のように作成します。

```
[ATDE ~/mkswu/update-dtb-descs]$ cat usb_dtb_sample.desc
component=extra_os.dtb ❶
version=1 ❷

swdesc_files --dest /boot ¥
        "armadillo.dtb" ❸

swdesc_script update_preserve_files.sh -- ¥
        "/boot/armadillo.dtb" ❹
```

図 7.11 usb_dtb_sample.desc 作成例

❶ component を指定します。今回は rootfs の変更を行うので extra_os を指定しています。

❷ バージョンを指定します。今後さらにアップデートする際にはこの数値を上げます。

❸ /boot に armadillo.dtb を配置します。

❹ Armadillo に書き込んだ/boot/armadillo.dtb が今後の OS アップデートによって削除されないように設定します。詳しくは、Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「swupdate_preserve_files について [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.4.0/ch09.html#sct.swupdate-preserve-files]」を参照してください。

desc ファイルの詳細については Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「mkswu の desc ファイル [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.4.0/ch09.html#sct.mkswu-desc]」を参照してください。/usr/share/mkswu/examples 以下の desc ファイルはサンプルですので、そちらも参考にしてください。

swu イメージを作成します。

```
[ATDE ~/mkswu/update-dtb-descs]$ mkswu usb_dtb_sample.desc
Enter pass phrase for /home/atmark/mkswu/swupdate.key: ❶
usb_dtb_sample.swu を作成しました。
```

図 7.12 swu イメージの作成

❶ 「図 7.2. mkswu の初期設定を行う」で設定した証明書の鍵のパスワードを入力してください。

これで DTB ファイルを Armadillo の所定のディレクトリに配置する swu イメージが完成しました。

2. USB メモリに swu イメージを書き込む

   mkswu 実行時に表示された指示に従って、作成した swu イメージを USB メモリに配置します。ATDE に USB メモリを接続して以下を実行してください。

```
[ATDE ~/mkswu/update-dtb-descs]$ df -h
Filesystem                Size  Used Avail Use% Mounted on
:（省略）
/dev/sdb1                  15G   24K   14G    1% /media/atmark/USBDRIVE ❶

[ATDE ~/mkswu/update-dtb-descs]$ sudo cp usb_dtb_sample.swu /media/atmark/USBDRIVE/ ❷
[ATDE ~/mkswu/update-dtb-descs]$ umount /media/atmark/USBDRIVE/ ❸
```

図 7.13 アップデート用ファイル群の配置

❶ USB メモリがマウントされている場所を確認します。

❷ swu イメージを USB メモリへコピーします。

❸ USB メモリをアンマウントします。コマンド終了後に USB メモリを取り外してください。

3. SWUpdate の実行

   initial_setup と同様に、Armadillo に USB メモリを挿入すると自動的にアップデートが実行され、コンテナイメージの組み込みとコンテナの自動実行設定が行われます。

   自動的に再起動するので、再度 Armadillo にログインし、DTB ファイルが書き換わって起動していることをご確認ください。

   以上で、作成した DTB ファイルを SWUpdate を用いて Armadillo に配置することができました。

7.4. コンテナデザインパターン

Armadillo Base OS では基本的にユーザーアプリケーションはコンテナ内に格納されます。場合によっては機能毎にコンテナを複数使用したり、コンテナ内からホストである Armadillo Base OS 側のリソースへのアクセスやコマンド実行をしたりしたい場合などがあります。この章では、上記のような場合に Armadillo Base OS 上でコンテナをどう構築するのが良いかを例を示しつつ紹介します。

7.4.1. ホストコマンドを実行する

ホスト(Arnadillo Base OS)のコマンドは、コンテナ内から直接実行することはできません。しかし、実現したいシステムによってはコンテナ内からホストコマンドを実行したいことがあります。

コンテナ内からホストコマンドを実行する方法を以下の 2 つのパターンに分けて紹介します。

- ボタン押下や、USB 挿抜などのイベントをトリガにホストコマンドを実行する場合
- コンテナ内から任意のタイミングでホストコマンドを実行する場合

7.4.1.1. イベントをトリガにホストコマンドを実行する

Armadillo Base OS が特定のイベントを検知した際にホストコマンドを実行するように設定できます。そのため、厳密に言うとコンテナ内からホストコマンドを実行することにはなりませんが、コンテナ内のアプリケーションが動作中でもホストコマンドが実行されます。

トリガとするイベントによって設定方法が異なります。以下では udev を用いてデバイスの挿抜を検知してホストコマンドを実行するパターンと、buttond を用いてユーザースイッチや外付けのキーボード等の入力を検知してホストコマンドを実行するパターンの 2 パターンについて説明します。

1. udev を用いて USB メモリの挿入を検知し mount コマンドを実行する

   udev は、Linux カーネル用のデバイス管理ツールです。デバイスが接続もしくは接続解除された時にカーネルは uevent を udev に通知します。この時 udev は予め設定しておいたルールに従って、受け取った uevent に対応した処理を行います。

   udev を活用することでデバイスの挿抜をトリガに任意のホストコマンドを実行できますので、コンテナ内で何かを設定する必要はありません。

   以下では USB メモリの挿抜時に mount コマンドを実行し、/mnt ディレクトリにマウントする例を紹介します。

   まず、/etc/udev/rules.d/に、99-usb-automount.rules というファイルを作成して、以下のように内容を編集します。

   ```
   [armadillo ~/]# cat /etc/udev/rules.d/99-usb-automount.rules
   ACTION==”add”, KERNEL==”sd*”, SUBSYSTEM==”block”, ENV{ID_FS_USAGE}==”filesystem”, RUN+=”/bin/mount /dev/%k /mnt”
   ```

   その後、以下のコマンドを実行して udev ルールの再読込みを行います。

   ```
   [armadillo ~/]# udevadm control -R
   ```

   以上で設定完了ですので、動作確認をしてみます。

   ```
   [armadillo ~/]# mount | grep /mnt ❶
   [armadillo ~/]# ❷
   [84250.573893] usb 1-1: new high-speed USB device number 8 using xhci-hcd
   [84250.732277] usb-storage 1-1:1.0: USB Mass Storage device detected
   : (省略)
   [84252.013348]  sda: sda1
   [84252.020039] sd 0:0:0:0: [sda] Attached SCSI removable disk
   [84252.214274] EXT4-fs (sda1): mounted filesystem with ordered data mode. Opts: (null)
   [armadillo ~/]# mount | grep /mnt ❸
   /dev/sda1 on /mnt type ext4 (rw,relatime)
   ```

❶ /mnt ディレクトリに何もマウントされていないことを確認します。

❷ ここで USB メモリを挿入します。

❸ USB メモリが/mnt ディレクトリに自動的にマウントされていることを確認します。

動作確認後、persist_file コマンドを実行して設定した udev ルールを永続化します。

```
[armadillo ~/]# persist_file /etc/udev/rules.d/99-usb-automount.rules
```

上記手順では mount を例に紹介しましたが、他のコマンドやシェルスクリプトの実行なども可能です。

2. buttond を用いてユーザースイッチの押下を検知してホストコマンドを実行する

Armadillo Base OS には、ユーザースイッチや外付けのキーボードなどの入力をイベントとして検知し、予め定義したルールによってそのイベントに対応させた処理を実行できる buttond という仕組みがあります。

buttond についての詳細は、Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「 ボタンやキーを扱う [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.4.0/ch09.html#idm6048]」を参照してしてください。

以下では、buttond を用いてユーザースイッチを 5 秒以上長押しすると reboot コマンドを実行する例を紹介します。

まず、/etc/atmark/buttond.conf を以下のように作成して、persist_file コマンドで永続化します。また、buttond サービスを再起動させます。

```
[armadillo ~/]# vi /etc/atmark/buttond.conf ❶
BUTTOND_ARGS="$BUTTOND_ARGS -l prog1 -t 5000 -a 'reboot'"
[armadillo ~/]# persist_file /etc/atmark/buttond.conf ❷
[armadillo ~/]# rc-service buttond restart ❸
```

❶ /etc/atmark/buttond.conf を作成・編集します。

❷ persist_file コマンドで永続化します。

❸ buttond サービスを再起動させます。

その後、Armadillo-IoT ゲートウェイ G4 のユーザースイッチ(SW1)を 5 秒長押しして再起動することを確認してください。

上記手順では reboot を例に紹介しましたが、他のコマンドやシェルスクリプトの実行なども可能です。

7.4.1.2. 任意のタイミングでホストコマンドを使用する

「7.4.1.1. イベントをトリガにホストコマンドを実行する」で紹介した方法は、デバイスの挿抜やスイッチの押下などのイベントをトリガとしてホストコマンドを実行するものでした。

イベントをトリガとせずに任意のタイミングでホストコマンドを実行したい場合は、コンテナからホストに対して ssh で接続して実行します。

1. Armadillo Base OS 側の準備 1

   まず、ssh サーバとなる Armadillo Base OS 側の設定を行います。以下のコマンドを実行して、ssh サーバを自動的に起動するよう設定し、sshd サービスを再起動します。

```
[armadillo ~/]# rc-update add sshd
 * service sshd added to runlevel default
[armadillo ~/]# persist_file /etc/runlevels/default/sshd
[armadillo ~/]# rc-service sshd restart
```

2. コンテナの準備

   以下では説明の為に at-debian-image をベースとしたコンテナを作成します。予め「6.3.2. サンプルコンテナをビルド」の手順に従って、at-debian-image の Dockerfile からイメージを生成しておく必要があります。

```
[armadillo ~/]# podman run -it --name=ssh_sample ¥
localhost/at-debian-image:latest /bin/bash
[container /]#
```

   コンテナ内で以下のコマンドを実行して ssh の準備をします。

```
[container /]# apt install -y openssh-client ❶
[container /]# ssh-keygen -t rsa -N "" -f /root/.ssh/id_rsa ❷
[container /]# ls /root/.ssh/id_rsa*    ❸
/root/.ssh/id_rsa     /root/.ssh/id_rsa.pub
[container /]# <Ctrl+P> <Ctrl+Q> ❹
[armadillo ~/]#
```

   ❶ openssh-client をコンテナ内にインストールします。

   ❷ 公開鍵認証のための認証鍵を生成します。

   ❸ 公開鍵・秘密鍵が生成されていることを確認します。

   ❹ コンテナから抜けます。

3. Armadillo Base OS 側の準備 2

   ssh サーバである Armadillo Base OS の方に、先程コンテナ内で生成した公開鍵を登録します。

```
[armadillo ~/]# mkdir /root/.ssh
[armadillo ~/]# chmod 600 /root/.ssh
[armadillo ~/]# podman exec -it ssh_sample /bin/cat /root/.ssh/id_rsa.pub >> /root/.ssh/ ⏎
authorized_keys
```

4. ssh でコンテナ内からホストコマンドを実行

   再度コンテナ側に戻り、ssh 経由でホストコマンドである reboot を実行してみます。コンテナから Armadillo Base OS へは IP アドレス 10.88.0.1 でアクセスできます。

```
[armadillo ~/]# podman attach ssh_sample
[container /]# ssh root@10.88.0.1 reboot
:（省略）
```

初回 ssh 時に、「Are you sure you want to continue connecting (yes/no/[fingerprint])?」と表示される場合があります。手動実行であれば y を選択すれば問題ありませんが、シェルスクリプトなど非手動実行の場合は ssh 実行時に -o StrictHostKeyChecking=no オプションを指定すると上記質問を回避できます。

上記手順では reboot を例に紹介しましたが、他のコマンドやシェルスクリプトの実行なども可能です。

Armadillo Base OS 開発ガイド

改訂履歴

バージョン	年月日	改訂内容
1.0.0	2021/12/28	・初版発行
1.1.0	2022/04/27	・「7.1. SWUpdate を用いてソフトウェアをアップデートする」 を追加 ・「7.3. Device Tree を変更しハードウェアを拡張する」 を追加 ・「7.4. コンテナデザインパターン」 を追加 ・その他軽微な修正

Armadillo Base OS 開発ガイド
Version 1.1.0
2022/04/27