ROS 2とXilinx CNN推論エンジンの活用

ROSCon JP 2021

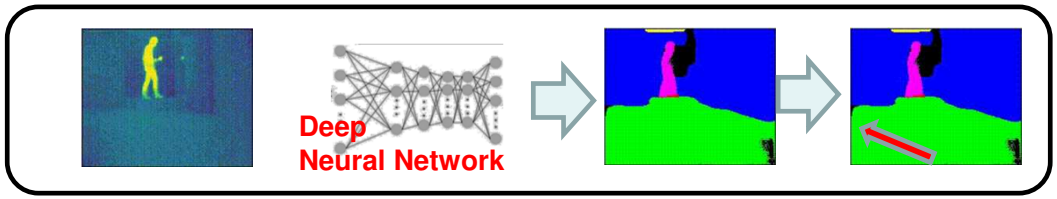
2021年9月16日

防衛装備庁 次世代装備研究所
丹羽雄一郎

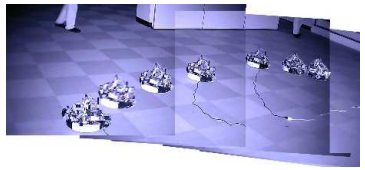
イーソル株式会社 SE事業部
藤井　大樹

本日お話する内容

1. 深層学習による実時間環境認識技術の紹介

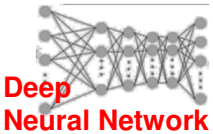


ROSロボット



Realtime Obstacle Detection & Avoidance

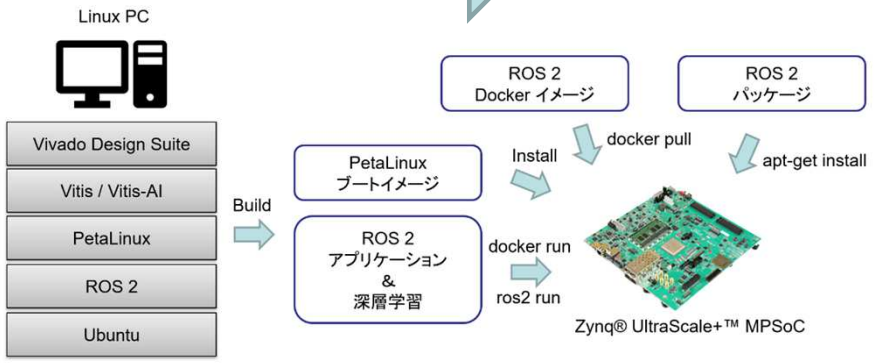
2. Xilinx FPGA SoCを用いた深層学習環境



Xilinx DNNDK
ROS2

Xilinx FPGA SoC

3. 環境構築のポイント

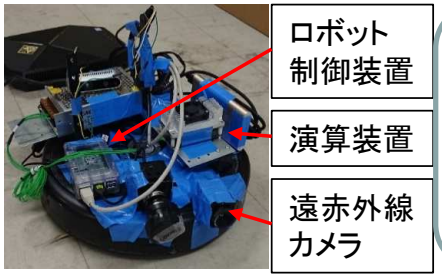


本日お話する内容

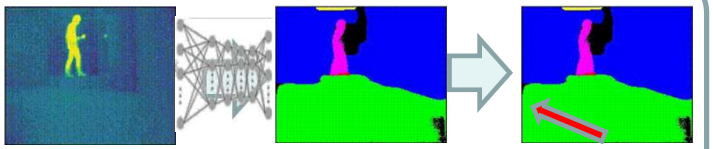
1. 深層学習による実時間環境認識技術の紹介
2. Xilinx FPGA SoCを用いた深層学習環境
3. 環境構築のポイント

ROSCON JP 2019LT　ROSを用いた実時間障害物回避

テスト環境（照明無し）

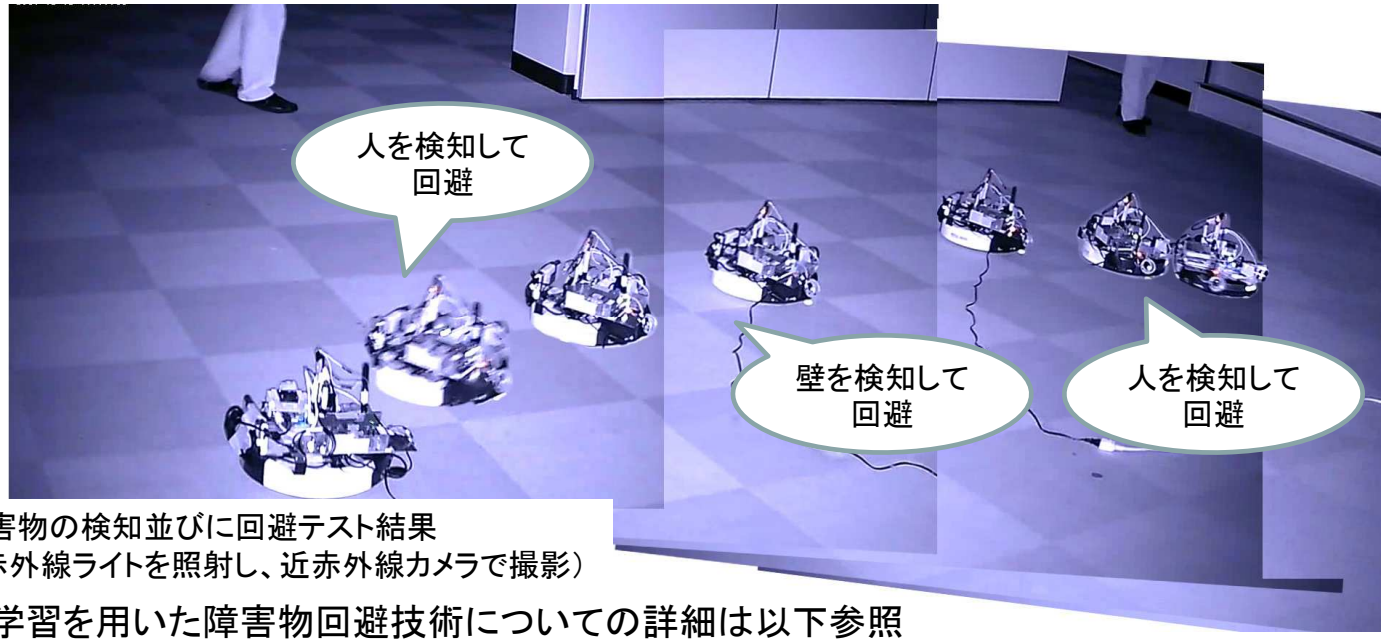


供試体（小型ロボット）



処理フロー

nVidia Jetson TX2（Mobile GPU）実装
CNN の縮小化に nVidia TensorRT利用



暗中の人、障害物の検知並びに回避テスト結果
（不可視な近赤外線ライトを照射し、近赤外線カメラで撮影）

試験内容

- 暗所室内において、無照明、無投光下での移動ロボットの障害物検知及び回避の実現
- 外界センサに遠赤外線カメラのみを利用した障害物検知及び回避の実現

https://roscon.jp/2019/presentations/lightning_talks/ROSCon_JP_2019_lightning_talk_3.pdf

ロボットや深層学習を用いた障害物回避技術についての詳細は以下参照
https://www.matlabexpo.com/jp/2020/proceedings.html　遠赤外線画像へのGANの適用と自律移動ロボットの制御（丹羽）

問題点と解決案

遅い！
もっと処理を高速化したい

- CNN@Mobile GPUによる遠赤外線画像を用いた障害物検知・回避
  - 遠赤外画像の領域分割処理 4．3Hz（233m秒）
  - ロボット制御周期　3Hz　（333m秒）
  - 消費電力（nVidia JetsonTX2）10W

FPGAを用いたCNNの高速処理技術に着目

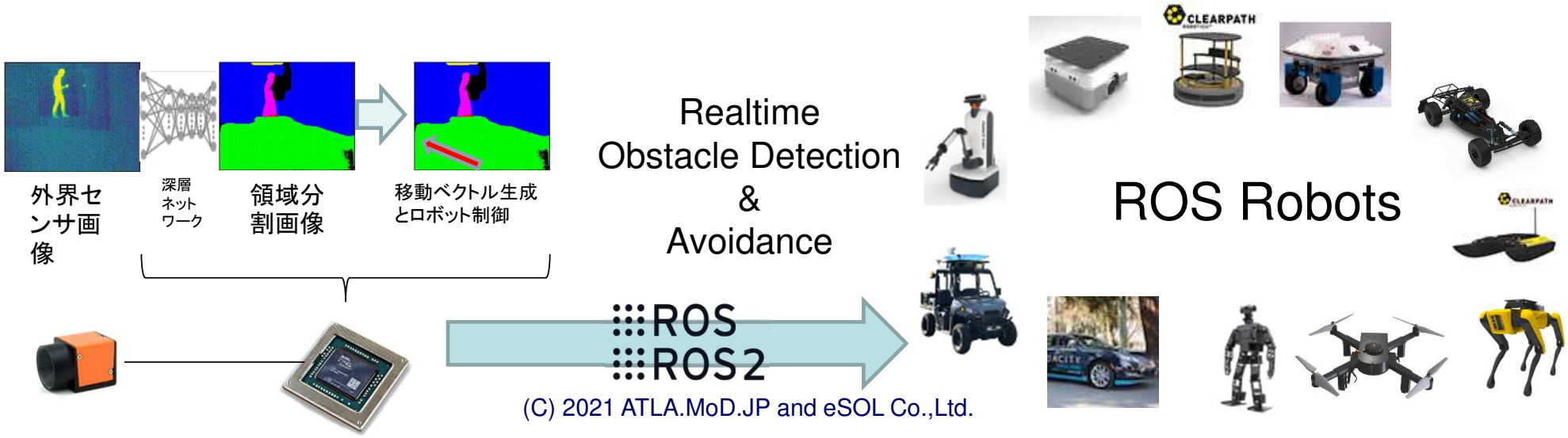
33倍の高速化
消費電力1／2

- CNN@FPGAによる遠赤外線画像を用いた障害物検知・回避
  - 遠赤外画像の領域分割処理 102Hz（9.8m秒）
  - ロボット制御周期　100Hz　（10m秒）
  - 消費電力（Xilinx zcu102）5W

詳細は、丹羽,藤井「自律ロボットのための遠赤外線画像意味的領域分割とそのFPGA実装」,電子情報通信学会,2021.01
https://www.ieice.org/publications/ken/summary.php?contribution_id=111777

本発表でのROSコミュニティへの貢献

- FPGA SoC（FPGA ＋ CPU ＋ I/O）へのROS 2の導入事例
  - Xilinx Zynq® UltraScale+™ MPSoCへのROS 2ノード実装
- 深層ニューラルネットワークのFPGA SoC実装事例
  - Xilinx Vitis AIによるCNNの実装
- 自律移動ロボットにおける障害物検知及び回避処理の高速化並びに低消費電力化実装事例



本日お話する内容

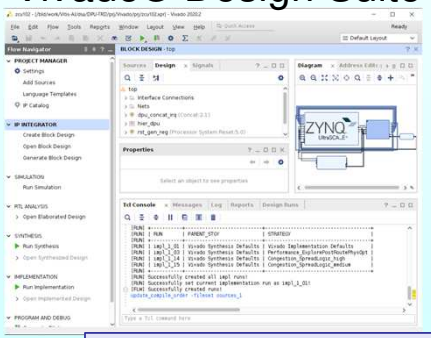
1. 深層学習による実時間環境認識技術の紹介
2. Xilinx FPGA SoCを用いた深層学習環境
3. 環境構築のポイント

Xilinx FPGA SoCを用いた深層学習環境

本日お話する範囲

CUI toolkit

Vivado® Design Suite



XSA

(1) CPU/FPGAのビルド
(Vivado Flow)

(2) Linuxのビルド
(Vivado Flow)

PetaLinux

Boot Image

rootfs

SDK

Vitis

(3) Vitis実行環境のビルド
(Vitis Flow)

(4) 深層学習の実施

Vitis
Vitis AI
DNNDK

Deep Learning
Environment

(5) アプリケーションの開発

Acceleration
Application

Install to SD Card

Zynq® UltraScale+™ MPSoC

https://japan.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html

Vivado® Design Suite

- Xilinx FPGA SoCのハードウェアを開発する統合環境
  - RTL（VerilogやVHDL）
  - FPGA回路
  - IPのインテグレーション
- ターゲットボード用のXSAファイル
  - ハードウェア情報
  - SoC(DDR、PLLなど)の初期化コード
- ソフトウェアライセンス
  - ZCU102では有償版か無償評価版(30日間)を利用する

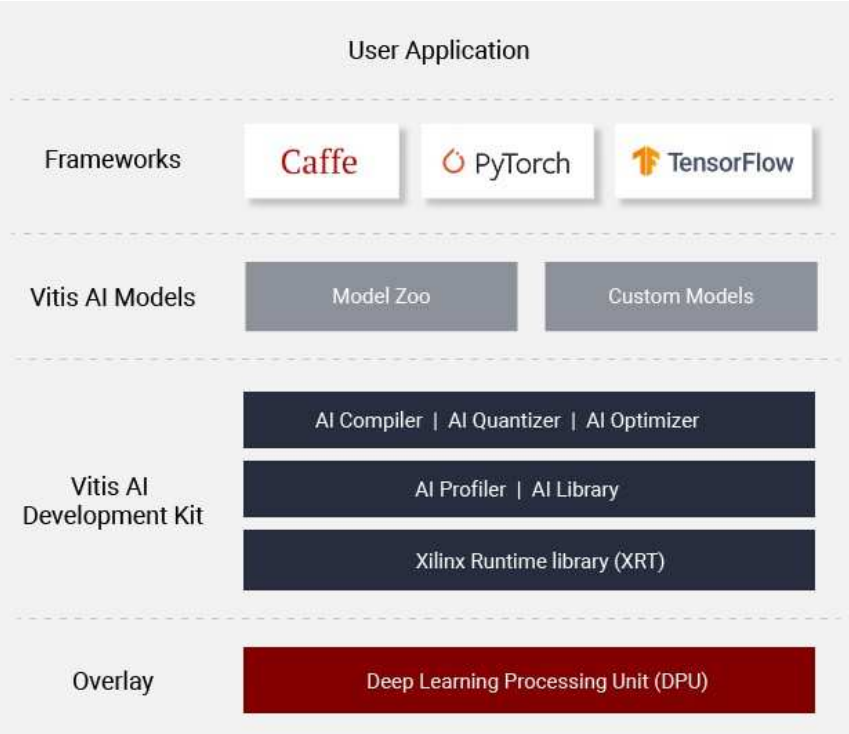
PetaLinux

- Xilinx FPGA SoC上で動作するLinux環境を構築するツール
- Xilinx FPGA SoC上で動作するLinuxディストリビューション
- ビルドシステムとしてYoctoを採用
  - 任意のパッケージやレシピファイルを追加することでカスタマイズできる
- ROS 2による深層学習の利用ではこの環境に手を加える

Vitis / Vitis AI

一言でいうと深層学習のための開発環境

※正確には、従来のXilinx SDK + SDSoC + SDAccelに
AI Frameworkを統合したもの



フレームワーク

学習済みモデル

コンパイラ、解析ツール

AIライブラリ

深層学習用のIP(ハードウェア)

https://japan.xilinx.com/products/design-tools/vitis/vitis-ai.html

DNNDK

- Vitis登場前から利用できる深層学習用のSDK
  - Vitis 1.3ではレガシAPIとしてサポート
  - 先に紹介した研究開発で利用
- シンプルなAPI
- 2021年7月リリースのVitis 1.4でDNNDKは削除された
  - 新しく始める場合はVitis AIランタイム(VART)の利用を推奨
  - 本資料に記載した内容はVART利用にも適合するはず

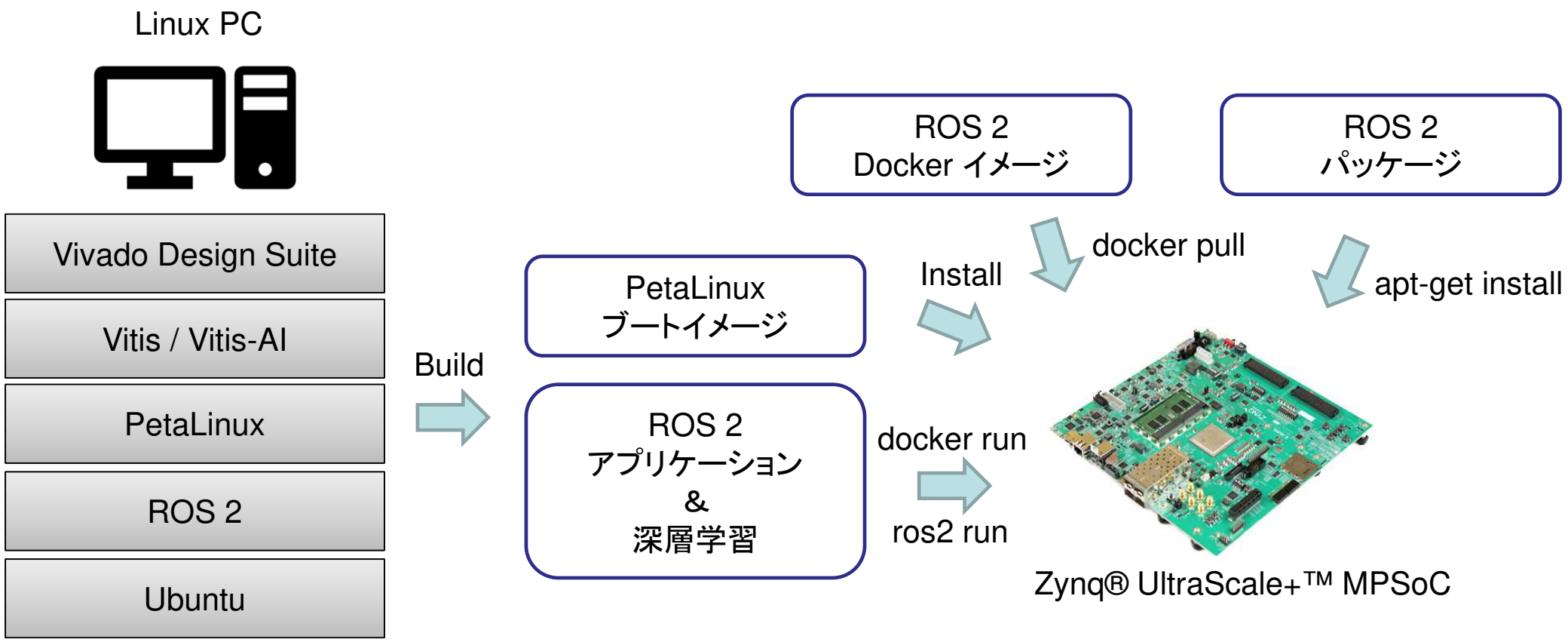
PetaLinuxでROS 2を利用する方法

開発効率が高いID-2の方法を採用した

ID	方法	メリット	デメリット
1	meta-rosを利用する	• ブート直後からROS 2が利用できる • フットプリントが小さい	• 開発中にパッケージ追加や更新を行う場合にPetaLinuxの再ビルドが必要 • 複数のROSバージョン共存が困難 • 野良パッケージの依存関係解消が困難
2	Dockerを利用する	• 公式のDockerイメージが利用できる • 公式のクロスビルドツールチェーンが利用できる • 任意のROSバージョンが利用できる	• Docker由来の問題解決が必要 • パフォーマンス低下のリスク

開発効率が高い ＝ PC Linuxと同じ方法で開発可能
aptによるパッケージの追加・更新
colconを使ったビルド
GDBによるリモートデバッグ
など

開発環境の大まかなイメージ



本日お話する内容

1. 深層学習による実時間環境認識技術の紹介
2. Xilinx FPGA SoCを用いた深層学習環境
3. 環境構築のポイント

準備するハードウェア

- Vitis AIがサポートするFPGAボード 1台
  - 情報量の豊富なZCU102評価ボードがお勧め
  - https://japan.xilinx.com/products/design-tools/vitis/vitis-ai.html#deployment
- 開発用PC 1台
  - できる限り高性能なCPUと32GB以上のメモリ、100GB以上のストレージ
  - できる限り高性能なNVIDIA製のGPU
    - NVIDIA GeForce GTX 10 Series または NVIDIA GeForce RTX 20 Series
    - 深層学習を効率よく行うにはCUDAコア数とVRAMサイズが重要

    ※ NVIDIA GeForce RTX 30 Series は非対応の可能性がある（後述）
  - スペック例
    - CPU：Intel Core i7-9700K
    - メモリ：32GB
    - ストレージ：SSD 512GB + HDD 2TB
    - GPU：NVIDIA GeForce RTX 2080 SUPER

準備するソフトウェア

各ソフトウェアのバージョン確認が重要

ID	ソフトウェア	バージョン	説明
1	Ubuntu	18.04 LTS	Vitis AI 1.3.2がサポートするバージョン
2	CUDA toolkit	9.0 または 10.0	NVIDIA Graphicsドライバのバージョンに依存 ・GTX 10 Seriesの場合はCUDA 9.0 ・RTX 20 Seriesの場合はCUDA 10.0 ※RTX 30 Seriesの場合はCUDA 11.0となるので対象外 ※CUDA 10.1等のマイナーアップデート版も利用しない ※https://docs.nvidia.com/deploy/cuda-compatibility/
3	NVIDIA Graphics Driver	384以降または410以降	CUDAの実行に必要 ・GTX 10 Seriesの場合はNVIDIA-384以降 ・RTX 20 Seriesの場合はNVIDIA-410以降 ※NVIDIAのインストーラによってインストールされるドライバを利用
4	NVIDIA container toolkit	最新版	Vitis AIの深層学習実行環境で利用
5	Vivado Design Suite	2020.2	Vitis AI 1.3.2で動作確認済みのバージョン
6	PetaLinux	2020.2	Vitis AI 1.3.2で動作確認済みのバージョン
7	Vitis AI	1.3.2	DNNDKをサポートする最新バージョン

事前に確認するドキュメント

作業前に用語や手順を確認する

- Vitis AI - https://github.com/Xilinx/Vitis-AI/tree/1.3.2
- DPU TRD - https://github.com/Xilinx/Vitis-AI/tree/1.3.2/dsa/DPU-TRD
- Vitis AI 1.3 ユーザーガイド - https://www.xilinx.com/support/documentation/sw_manuals_j/vitis_ai/1_3/ug1414-vitis-ai.pdf
- PetaLinux Tools - https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_2/ug1144-petalinux-tools-reference-guide.pdf
- Vitis開発資料 - https://japan.xilinx.com/html_docs/xilinx2020_2/vitis_doc/index.html

アプリケーション開発までの流れ

1. NVIDIA Graphics Driver、CUDAのインストール
2. DockerとNVIDIA container toolkitのインストール
3. VivadoとPetaLinuxのインストール
4. Vitis AIの深層学習実行環境のビルド
5. CPU/FPGAのビルド
6. Linuxのビルド
7. Vitis実行環境のビルド
8. SDカードへのインストールとXilinx FGPA SoCでの起動
9. 深層学習と量子化およびFPGAモジュールのビルド
10. ROS 2アプリケーションのビルドと実行

開発環境の構築

深層学習の実施

アプリケーションの開発

1. NVIDIA Graphics Driver、CUDAのインストール

- CUDAのインストール
  - CUDA Toolkit Archiveから対象のバージョンを探してインストール
    - https://developer.nvidia.com/cuda-toolkit-archive
  - 自動的にNVIDIA Graphics Driverもインストールされる
  - インストール時にCUDAのバージョンを指定する
    - $ sudo apt-get install cuda-10-0
- 環境変数の設定
  - ~/.bashrcへ追記
    - $ export PATH="/usr/local/cuda/bin:$PATH“
    - $ export LD_LIBRARY_PATH="/usr/local/cuda/lib64:$LD_LIBRARY_PATH“

2. DockerとNVIDIA container toolkitのインストール

- Docker-ceのインストール
  - https://docs.docker.com/engine/install/ubuntu/
- NVIDIA container toolkitのインストール
  - https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html#setting-up-nvidia-container-toolkit

3. VivadoとPetaLinuxのインストール

- 必要なインストーラ
  - ザイリンクス統合インストーラー 2020.2 (Vivado Design Suite)
  - PetaLinux 2020.2 インストーラー
- インストール前に実施すること
  1. インストール先のフォルダの作成
     - $ sudo mkdir –p /opt/Xilinx /opt/Xilinx/petalinux/2020.2
     - $ sudo chown -R $USER:$USER /opt/Xilinx
  2. 依存パッケージのインストール
     - $ wget https://www.xilinx.com/Attachment/plnx-env-setup.sh
     - $ chmod +x plnx-env-setup.sh
     - $ ./plnx-env-setup.sh

4. Vitis AIの深層学習実行環境のビルド

- ドキュメントやソースコードはバージョンを指定する
  - https://github.com/Xilinx/Vitis-AI/tree/1.3.2
  - git clone --recurse-submodules https://github.com/Xilinx/Vitis-AI -b v1.3.2
- 手順書のコピペに注意!
  - 文章中のURLにはバージョン指定のないところが多数ある

5. CPU/FPGAのビルド

- ドキュメントはバージョンを指定する
  - https://github.com/Xilinx/Vitis-AI/blob/1.3.2/dsa/DPU-TRD/prj/Vivado/README.md
- ビルド済みのXSA(ZCU102用)を利用すると手順を省略できる(かも)
  - 上記方法でDNNDKの利用可否は不明

6. Linuxのビルド

- ドキュメントはバージョンを指定する
  - https://github.com/Xilinx/Vitis-AI/blob/1.3.2/dsa/DPU-TRD/prj/Vivado/README.md
- Yoctoのカスタマイズはpetalinux-configコマンドを実行前に行う
  1. Linuxカーネルのコンフィグレーション変更
  2. Docker関連のパッケージ追加

6. Linuxのビルド

project-spec/meta-user/recipes-kernel/linux/linux-xlnx/user.cfgへ追加

```
CONFIG_NAMESPACES=y
CONFIG_NET_NS=y
CONFIG_PID_NS=y
CONFIG_IPC_NS=y
CONFIG_UTS_NS=y
CONFIG_CGROUP_CPUACCT=y
CONFIG_CGROUP_DEVICE=y
CONFIG_CGROUP_FREEZER=y
CONFIG_CGROUP_SCHED=y
CONFIG_CPUSETS=y
CONFIG_MEMCG=y
CONFIG_VETH=y
CONFIG_IP_NF_TARGET_MASQUERADE=y
CONFIG_NETFILTER_XT_MATCH_ADDRTYPE=y
CONFIG_NETFILTER_XT_MATCH_IPVS=y
CONFIG_IP_NF_NAT=y
CONFIG_USER_NS=y
CONFIG_SECCOMP=y
CONFIG_CGROUP_PIDS=y
CONFIG_MEMCG_SWAP=y
CONFIG_MEMCG_SWAP_ENABLED=y
CONFIG_BLK_CGROUP=y
CONFIG_BLK_DEV_THROTTLING=y
CONFIG_CFQ_GROUP_IOSCHED=y
CONFIG_CGROUP_PERF=y
CONFIG_CGROUP_HUGETLB=y
CONFIG_NET_CLS_CGROUP=y
CONFIG_CGROUP_NET_PRIO=y
CONFIG_CFS_BANDWIDTH=y
CONFIG_FAIR_GROUP_SCHED=y
CONFIG_RT_GROUP_SCHED=y
```

6. Linuxのビルド

project-spec/meta-user/recipes-kernel/linux/linux-xlnx/user.cfgへ追加

```
CONFIG_IP_NF_TARGET_REDIRECT=y
CONFIG_IP_VS=y
CONFIG_IP_VS_NFCT=y
CONFIG_IP_VS_PROTO_TCP=y
CONFIG_IP_VS_PROTO_UDP=y
CONFIG_IP_VS_RR=y
CONFIG_EXT3_FS_XATTR=y
CONFIG_EXT3_FS_POSIX_ACL=y
CONFIG_EXT3_FS_SECURITY=y
CONFIG_VXLAN=y
CONFIG_INET_ESP=y
CONFIG_IPVLAN=y
CONFIG_MACVLAN=y
CONFIG_DUMMY=y
CONFIG_NF_NAT_FTP=y
CONFIG_NF_CONNTRACK_FTP=y
CONFIG_NF_NAT_TFTP=y
CONFIG_NF_CONNTRACK_TFTP=y
CONFIG_AUFS_FS=y
CONFIG_BTRFS_FS_POSIX_ACL=y
CONFIG_BLK_DEV_DM=y
CONFIG_DM_THIN_PROVISIONING=y
CONFIG_OVERLAY_FS=y
CONFIG_MD=y
CONFIG_NET_SCHED=y
CONFIG_NET_L3_MASTER_DEV=y
CONFIG_BRIDGE=m
CONFIG_OVERLAY_FS=y
CONFIG_BPF_SYSCALL=y
CONFIG_CGROUP_BPF=y
```

6. Linuxのビルド

project-spec/meta-user/conf/petalinuxbsp.confへ追加

```
IMAGE_INSTALL_append = " containerd-opencontainers"
IMAGE_INSTALL_append = " docker-ce"
IMAGE_INSTALL_append = " runc-opencontainers"
IMAGE_INSTALL_append = " libcgroup"
IMAGE_INSTALL_append = " cgroup-lite"
```

7. Vitis実行環境のビルド

- ドキュメントはバージョンを指定する
  - https://github.com/Xilinx/Vitis-AI/blob/1.3.2/dsa/DPU-TRD/prj/Vitis/README.md
- 必要なソフトウェア
  - XRT 2020.2
    - ソースコードからビルド
  - zcu102 base platform 2020.2
    - Xilinx社のホームページからダウンロード
- 手順上の注意点
  - 「6. Linuxのビルド」で作成したイメージのフォルダを指定
    - $ export EDGE_COMMON_SW=<work>/xilinx-zcu102-trd/images/linux/
  - 「Modify Makefile $XOCC_OPTS parameters」の実施

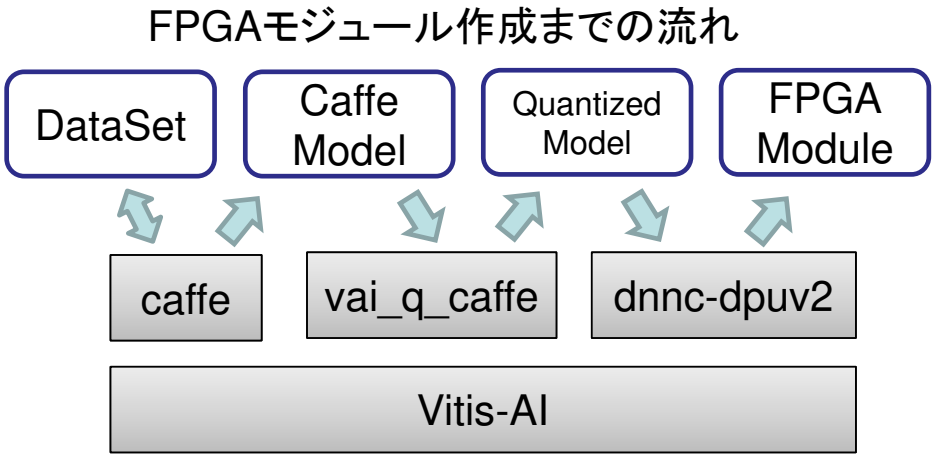
8. SDカードへのインストールとXilinx FGPA SoCでの起動

- Vitis Flowで作成したSDカードイメージを利用する
  - Vitis-AI/dsa/DPU-TRD/prj/Vitis/binary_container_1/sd_card.img
  - Vivado Flowで作成したイメージではDNNDKは利用できない
- ブート後、USBシリアルやSSH経由でログイン可能

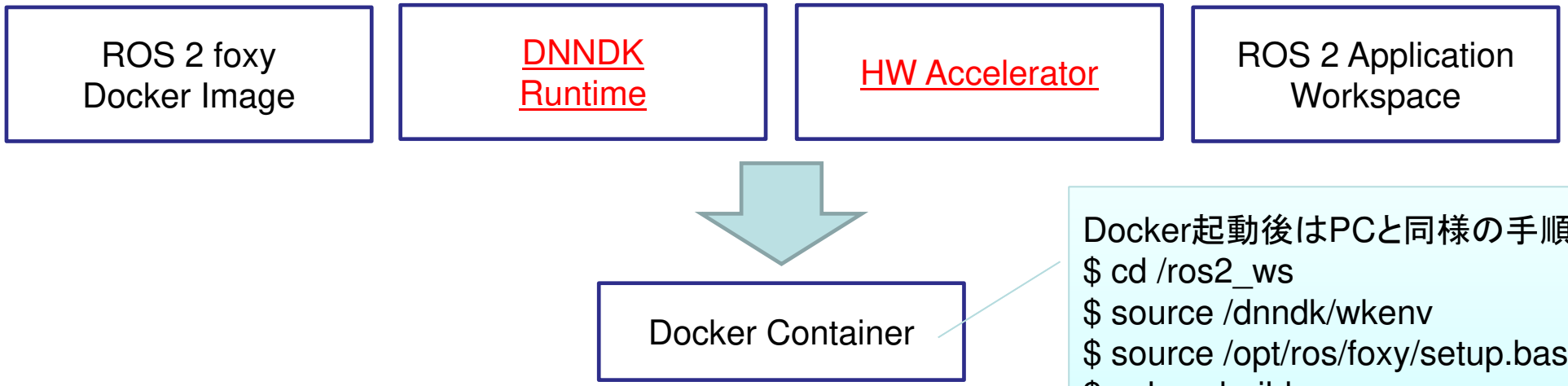
9. 深層学習と量子化およびFPGAモジュールのビルド

- Vitis-AI実行環境の起動
  - ./docker_run.sh xilinx/vitis-ai-gpu:latest
- フレームワークを選択
  - conda activate vitis-ai-caffe
- データセットの作成とトレーニング
  - Caffeの学習済みモデルを出力(*.caffemodel)
  - caffe train コマンドを使用
- Xilinx FPGA SoC用の量子化
  - deploy.prototxtおよびdeploy.caffemodelを出力
  - vai_q_caffe quantize コマンドを使用
- FPGAモジュールのビルド
  - DNNDKの場合：C/C++とリンク可能なFPGAモジュール(<module name>.elf)をビルド
  - dnnc-dpuv2 コマンドを使用

※Vitisアプリケーションの場合はvai_c_caffeを実行し*.xmodelを出力



10. ROS 2アプリケーションのビルドと実行



Docker起動後はPCと同様の手順
$ cd /ros2_ws
$ source /dnndk/wkenv
$ source /opt/ros/foxy/setup.bash
$ colcon build
$ ros2 run pkg node

Dockerの起動例

```
$ docker run –it --rm ¥
      -v /home/user/ros2_ws:/ros2_ws ¥                    # ROS 2 Application Workspace
      -v /home/user/dnndk:/dnndk ¥                        # DNNDK Runtime
      --device /dev/dri/renderD128:/dev/dri/renderD128 ¥  # HW Accelerator
      osrf/ros:foxy-desktop ¥                             # Docker Image
      bash                                                # Launch command
```

10. ROS 2アプリケーションのビルドと実行

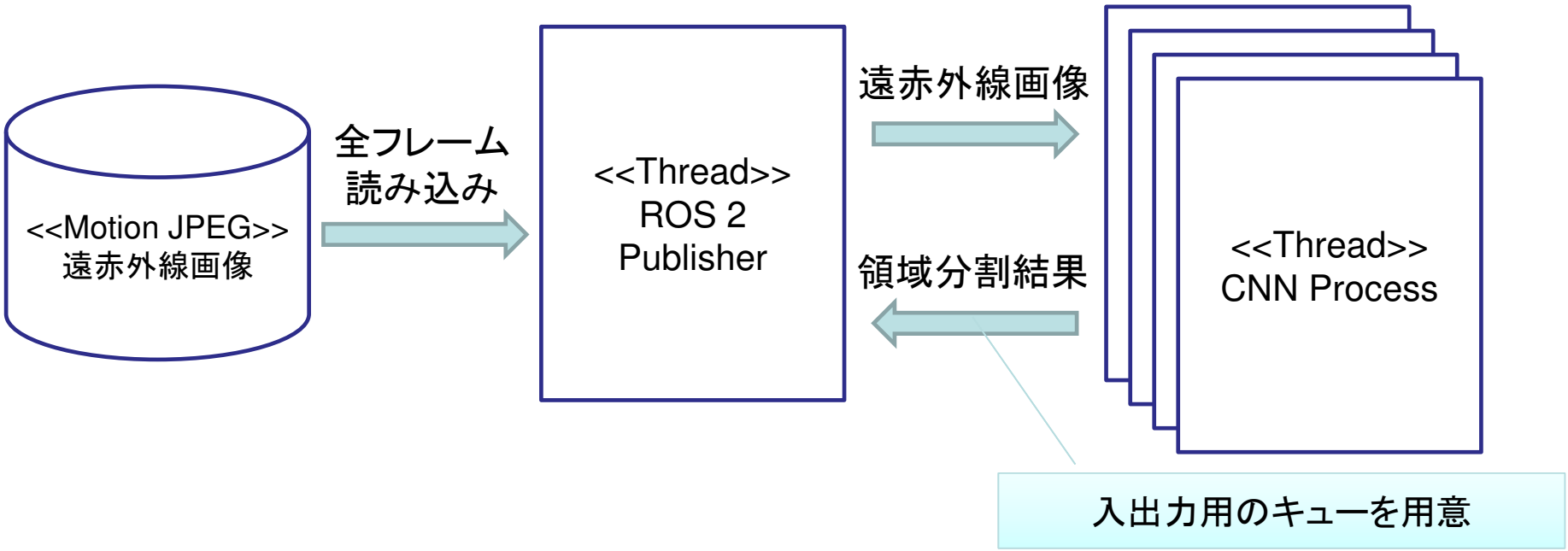
- DNNDK Runtime
  - DockerでDNNDKを利用するために必要なライブラリー式
  - https://www.xilinx.com/bin/public/openDownload?filename=vitis-ai_v1.3_dnndk.tar.gz
  - vitis-ai_v1.3_dnndk¥pkgs¥usr 以下のフォルダをdnndk以下へコピー
    - コピーするフォルダ：bin、include、libフォルダ
- dnndk/wkenv
  - DockerでDNNDK Runtimeを利用できるように環境変数を設定
    - export LD_LIBRARY_PATH=/dnndk/lib:$LD_LIBRARY_PATH
    - export PATH=/dnndk/bin:$PATH

10. ROS 2アプリケーションのビルドと実行

- DNNDKのサンプルコード
  - https://github.com/Xilinx/Vitis-AI/tree/1.3.2/demo/DNNDK
- FPGAモジュール(*.elf)のリンク方法
  - set(DPU_SEGMATION_ELF model/dpu_segmentation_0.elf)
  - add_executable(app src/main.cc ${DPU_SEGMATION_ELF})
  - SET_SOURCE_FILES_PROPERTIES(
  - ${DPU_SEGMATION_ELF}
  - PROPERTIES
  - EXTERNAL_OBJECT true
  - GENERATED true
  -)

10. ROS 2アプリケーションのビルドと実行

- DNNDK + ROS 2 Application → CNNの並列化で高スループットを実現
  - 1スレッド for ROS 2 Node
  - 4スレッド for CNN
  - 遠赤外線画像 on Memory



10. ROS 2アプリケーションのビルドと実行

FPGAモジュールの使用方法はシンプル

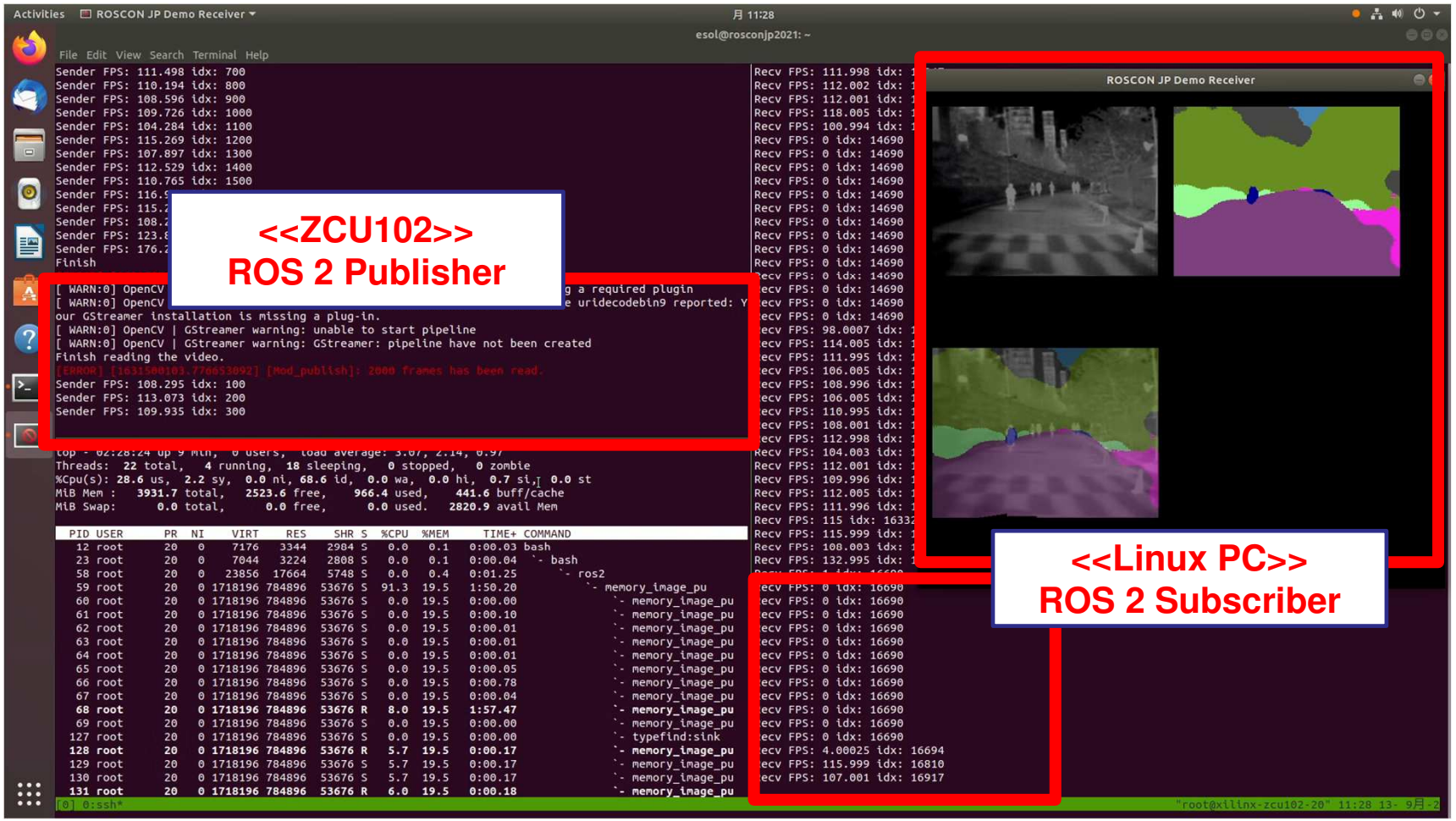
```
MyNode::MyNode()
{
  dpuOpen();                          // 初期化処理
                                      // FPGAモジュールのロード
  kern = dpuLoadKernel("dpu_segmentation_0");
  dpuCreateTask(kern, 0);          // タスク生成
}

MyNode::~MyNode()
{
  // 終了処理
  dpuDestroyTask(tsk);
  dpuDestroyKernel(kern);
  dpuClose();
}
```

```
MyNode::sub_image(img)
{
  // CNN Process
  dpuSetImputImage(img); // 遠赤外線画像を入力
  dpuRunTask(tsk);              // 領域分割処理@CNN
  res = dpuGetTensorAddress();      // 領域分割結果
  msg = toROSMsg(res);    // ROSメッセージを生成
  topic->publish(msg);         // Topicへ出力
}
```

注：購読した画像をDPU処理しトピックへ出力する概念コード

10. ROS 2アプリケーションのビルドと実行

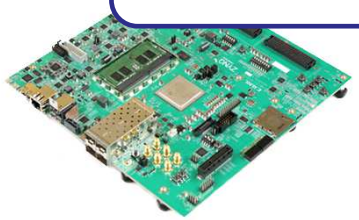


10. ROS 2アプリケーションのビルドと実行

Zynq® Ultr

ROS 2のノード通信においても性能発揮できることを確認

PC



PetaLinux 2020.2
ROS 2 foxy on Docker

Ubuntu 18.04.5 LTS
ROS 2 foxy on Docker

遠赤外線画像
セグメンテーション結果
移動可能方向

遠赤外線動画

110 Hz
<<ROS 2 Node>>
publisher

<<CNN>>
dpu_segmentation

130 Hz

110 Hz
<<ROS 2 Node>>
subscriber

Image Viewer

まとめ

- CNNのFPGA実装により実時間処理が可能であることの実証例
- Xilinx FPGA SoC環境における深層学習環境の実現
- 深層学習環境の構築におけるポイントの紹介

エッジデバイスにおいて実時間応答性が求められる計算量の多いシステムを
FPGA+CNN+ROS 2の組み合わせで解決できる可能性がある。

今回紹介したユースケース以外への活用を探っていきたい。