PerFC: An Efficient 2D and 3D Perception Software-Hardware Framework for Mobile Cobot

Tuan Dang, Khang Nguyen, Manfred Huber
University of Texas at Arlington
tuan.dang@uta.edu, khang.nguyen8@mavs.uta.edu, huber@cse.uta.edu

Abstract

In this work, we present an end-to-end software-hardware framework that supports both conventional hardware and software components and integrates machine learning object detectors without requiring an additional dedicated graphic processor unit (GPU). We design our framework to achieve real-time performance on the robot system, guarantee such performance on multiple computing devices, and concentrate on code reusability. We then utilize transfer learning strategies for 2D object detection and fuse them into depth images for 3D depth estimation. Lastly, we test the proposed framework on the Baxter robot with two 7-DOF arms and a four-wheel mobility base. The results show that the robot achieves real-time performance while executing other tasks (map building, localization, navigation, object detection, arm moving, and grasping) with available hardware like Intel onboard GPUs on distributed computers. Also, to comprehensively control, program, and monitor the robot system, we design and introduce an end-user application. The source code is available at https://github.com/tuantdang/perception_framework.

Introduction

Recent years have seen an increasing number of sensors with different modalities integrated into robots that significantly enhance robot perception, especially for autonomous service mobile cobots to perform map building, localization, navigation, object detection, and efficiently grasping of objects (Hsiao et al. 2009). This requires that besides support for conventional tasks in robot control and navigation, efficient techniques to deal with the high computational needs of 2D and 3D perception must also be deployed on the same system. Therefore an efficient software-hardware framework that enables sensors, communication, perception, navigation, and motion planning to operate seamlessly is a crucial part of incremental robotic development.

Previous works focus mainly on only one of the aspects (Hmedan et al. 2022; Vice et al. 2022), and flexible component integration is often omitted. Recently, Robot Operating System 2 (ROS 2) (Macenski et al. 2022) has improved security and reliability, which are critical in a commercial product; meanwhile, ROS 1 is still popular in research and industry. Nevertheless, the single failure point of the ROS master causes poor performance if multiple sensing modalities are initiated simultaneously, especially with high bandwidth data in LiDAR sensors and RGB-D cameras. Moreover, integrating state-of-the-art machine learning (ML) with optimal configurations into the ROS software stack can burden developers as no official framework can handle this task.

Recent works on deploying Deep Neural Networks (DNN) for safe and secure automation (Biondi et al. 2019; Nazarova et al. 2021) propose a visionary hypervisor-centric architecture. Yet, the integration of ML modules is meticulously tailored for specific applications. Also, their complexity worsens when they are deployed on different computing hardware. Thus, a framework is needed that eliminates repetitive tasks (training, testing, and detection) and is compatible with available hardware on robot systems.

To fill this gap, we build an efficient hardware-software framework (Fig. 1) that allows simple integration of various tasks and different hardware versions. More importantly, we introduce a design that can support multiple state-of-the-art object detection models and execute them on low-end commodity devices in real-time, enabling developers to manage computing tasks on available hardware in the system with high flexibility and minimum effort.

In this work, we make the following contributions: (1) building a complete software-hardware framework for a mobile cobot system that supports map building, localization, navigation, and motion planning, as well as 2D and 3D perception using state-of-the-art DNNs, (2) verifying the framework feasibility and performance on a real robot system, and (3) producing a fast method to train multiple object detection using transfer learning with open-source code.

Related Work

Architecture of Robot Software: Many software architectures have been designed for industrial robot applications (Rendiniello et al. 2020) to cope with robot-language dependency. While they address this, they limit users' ability to develop new functions. Specifically, adding sensing modalities or hardware becomes complicated since developers only access services from specific robot software toolkits. Accessing services at the OS layer or other libraries is restricted, preventing developers from accessing many open-source libraries. For those reasons, we develop a software framework that enables developers to flexibly access multiple system layers and open-source libraries while maintaining simple integration with most state-of-the-art ML models.

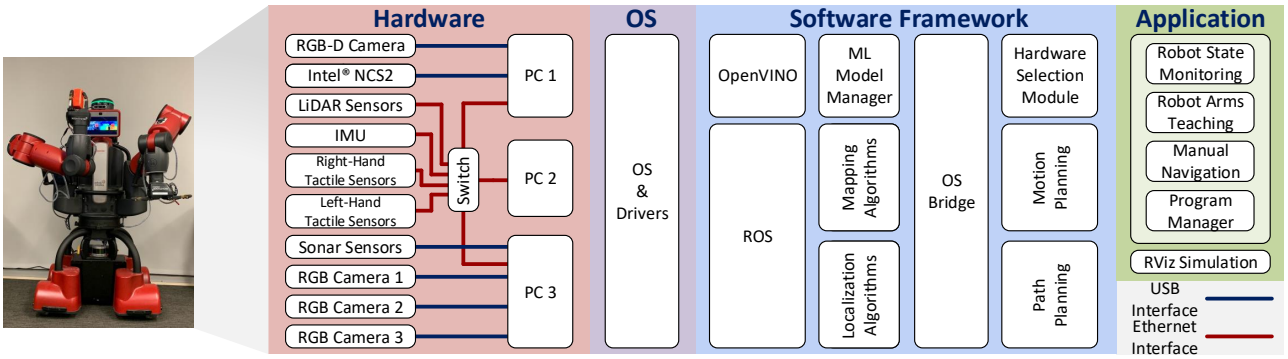

Figure 1: *PerFC* software-hardware framework includes hardware layer, OS layer, software framework, and application layer.

Robot Mapping, Localization, Motion, and Navigation: Previous works on the Baxter robot (Qureshi et al. 2019; Pinto et al. 2016) concentrate on pick-and-place and motion planning tasks. Thus, LiDAR sensors are crucial for its map building and navigation. Moreover, in the past autonomous mobile service robots with navigation and tracking modules (Veloso et al. 2015; Bellotto et al. 2008) have been embedded with lightweight, highly customized simultaneous localization and mapping (SLAM) algorithms due to their application simplicity (i.e., specific tasks and object tracking in a known environment). This specialization has often made these robots unexpandable for developers. Here we introduce an expandable framework and reuse motion planning supported by Moveit (Chitta et al. 2012) reflecting a well-studied research area (Ichnowski et al. 2020).

Robot Perception: 2D perception (Liu et al. 2016; Wang et al. 2022) is widely used in research and industrial applications, while 3D perception (Mao et al. 2022; Charles et al. 2017) is dominating in autonomous driving vehicles with LiDAR sensor support, but limited in everyday object perception in the robotic domain. One cause of the lack of robotic research in 3D perception is the absence of diverse labeled datasets since most of them are not specifically for robotic applications. In this work, we used a hybrid method of 2D state-of-the-art detection and 3D estimation methods.

Software-Hardware Framework

The design goals for our software-hardware framework are:

- Reusability and simple integration into systems with mixed versions of OS and middleware: Linux & ROS.
- State-of-the-art ML models deployed with optimal configurations of cameras and processing devices.

Hardware is often compatible with a specific Linux kernel, and a specific ROS distribution is only provided to a specific Linux version. For this reason, selecting hardware concurrently with selecting Linux kernels implies narrowing down options in choosing a ROS distribution for developers. Unfortunately, not all hardware works well with the same Linux kernel, leading to using various ROS versions in the same system. Therefore, calling the same APIs from different ROS distributions may cause backward incompatibility issues due to a slight change in the function prototype and the underlying implementation of that supported API. We adopt message conversion (Kim et al. 2013) between multiple communication protocols to implement the message translator between ROS versions.

The driver incompatibility problems can be solved by using suitable Linux kernels supporting these devices' drivers. However, it may raise backward incompatibility between a certain ROS distribution and APIs from other ROS distributions. Indeed, we encountered these compatibility issues with the Baxter robot (Ubuntu 14.04 and ROS Indigo) when executing motion APIs on Linux machines with Ubuntu versions other than Ubuntu 14.04. Therefore, an OS bridge between APIs from different ROS distributions is needed.

Framework Description

As illustrated in Fig. 1, we structure the software-hardware framework as four different layers: hardware layer, OS layer, software framework, and application layer.

1. **Hardware Layer** includes distributed computers, sensors (cameras, tactile sensors, LiDAR sensors), IMU, actuators, and ML accelerators and how they interface with each other through USB and Ethernet ports.
2. **OS Layer** contains an OS and device drivers that support connecting devices at the hardware layer.
3. **Software Framework** is the core contribution of this work. It bridges different Linux and ROS distributions, selects optimal configurations (Dang et al. 2022) of sensors to available hardware to operate in real-time, and supports basic functionalities of the robot system, including map building, localization, navigation, motion planning, and arm movement for grasping. Most of the components are implemented on top of ROS and OpenVINO.
4. **Application Layer** allows users to control and monitor the robot via a GUI, as shown in Fig. 2, which is written using the PyQt5 toolkit. Users can manually control the robot's arm and joint positions, navigate the mobility base, and program the robot using Python. Also, native simulations such as RViz can be used to monitor the robot.

Robot Perception

To perform real-time detection and recognition of a task relevant object set with limited richness compared with those in open datasets like ImageNet and MS COCO, we adopt two strategies: transfer learning (Zhuang et al. 2021) and single state detection (Liu et al. 2016). We first transfer knowledge from a rich feature domain into a sparse feature domain, which represents our dataset. In the following we mathematically model the overall concept of transfer learning and address the two questions: (i) what to be transferred between models and (ii) how to transfer that knowledge.

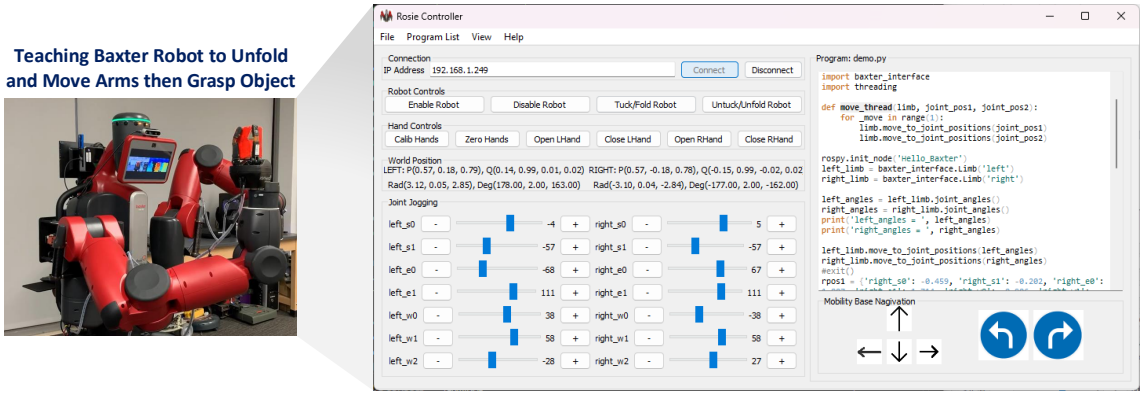

Figure 2: Graphic User Interface (GUI) provides robot teaching, monitoring, navigation, and programming.

Transfer Learning Definition: A domain is defined by $\mathcal{D} = \{\mathcal{X}, P(X)\}$ where $X = \{x_1, x_2, ..\} \in \mathcal{X}$ with $\mathcal{X}$ representing the feature space, and $P(X)$ its marginal distribution. Let $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$ be the learning task that learns from training pairs (x_i, y_i) with $y_i \in \mathcal{Y}$ in the label space. The objective of transfer learning is to improve the predictive function $P(Y_t|X_t)$ in target domain $\mathcal{D}_t = \{\mathcal{X}_t, P(X_t)\}$ using knowledge in the source domain $\mathcal{D}_s = \{\mathcal{X}_s, P(X_s)\}$ and source learning task $\mathcal{T}_s = \{\mathcal{Y}_s, P(Y_s|X_s)\}$. Let $P(Y|X) = f(X, \beta)$ where f is the task function. The minimizer for the trainable parameters, β, is written in terms of the loss function, $L(\cdot, \cdot)$, and the task function, f, as follows:

$$\text{argmin}_\beta \sum_X L\left[f\left(X, \beta\right), Y\right] \tag{1}$$

With respect to DL and computer vision concepts, we divide the task function into two components: feature extraction (backbone) and detection (head), such that $f(X, \beta) = (f^D \circ f^F)(X, \beta^D, \beta^F)$ where f^D and f^F are task function for detection and feature extraction, respectively, and β^D, β^F are parameters for detection and feature extraction, respectively. The analogous minimizer for β_t^F and β_t^D is:

$$\text{argmin}_{\{\beta_t^D, \beta_t^F\}} = \sum_{X_t} L\left[\left(f_t^D \circ f_t^F\right)\left(X_t, \beta_t^D, \beta_t^F\right), Y_t\right] \tag{2}$$

Since features in the source domain are more generalized and sufficiently cover our target domain, we assume that the feature spaces in the source and target domain are similar. However, our target labels are different ($\mathcal{Y}_s \neq \mathcal{Y}_t$) since we retrain the models with in-lab objects (cone, cube, and sphere). Here, we utilize two transfer learning strategies: (1) instance transfer, where the marginal distribution of source features is different from that of target features, and (2) feature representation transfer, where we fit the source feature domain into the target feature domain.

To implement instance transfer, we transfer $\left(\beta_s^D, \beta_s^F\right) \rightarrow \left(\beta_t^D, \beta_t^F\right)$, where β_s^D and β_s^F are resultants from source task functions, and fine-tune $\left(\beta_t^D, \beta_t^F\right)$ using Eq. 2. For feature representation transfer, we separate the source task into two components (backbone and head) and transfer the entire source task's backbone into the target task. Specifically, we transfer $\beta_s^F \rightarrow \beta_t^F$, and train β_t^D using Eq. 2. We also train with randomly initialized weights as a third strategy for accuracy comparisons in the Evaluation section.

Depth Estimation: We obtain depth images and RGB images simultaneously from an Intel RealSense D435i camera, which handles the depth image creation process, including camera calibration, image rectification, and disparity computation. As the whole predicted bounding box also covers non-detected objects, averaging the depth of the bounding box would incur estimation errors. We therefore down-scale the bounding box around its center and calculate the estimated depth of the object, D, by averaging depth values of each pixel in the scaled region as follows:

$$D = (w \times h)^{-1} \cdot \left[\sum_{i=x_0-w/2}^{x_0+w/2} \sum_{j=y_0-h/2}^{y_0+h/2} d(i,j)\right] \tag{3}$$

where $d(i, j)$ returns the depth value at pixel (i, j), w and h indicate width and height of the scaled region, respectively, and (x_0, y_0) are center coordinates of the bounding box.

Evaluation & Demonstration

We evaluate the add-on components for system completeness, such as 2D and 3D vision, hardware configurations, and their performance. Other components such as mapping, localization, navigation, and planning are well-supported ROS packages: 2D Navigation Stack and MoveIt!.

Data Preparation

To verify the correctness of our proposed method, we first collect data from three in-lab objects: cones, cubes, and spheres. We then label them with annotations in Pascal VOC format and split our custom dataset into three sets: training set (70%), validation set (20%), and test set (10%).

Evaluation Metrics

To evaluate how well the transfer learning strategies perform during training stages, we calculate the validation loss, average precision (AP), and mean average precision (mAP) for MobileNetv1 (Howard et al. 2017), MobileNetv2 (Sandler et al. 2018), SqueezeNet (Iandola et al. 2016), VGG-16 (Simonyan et al. 2014), and YOLOv7 (Wang et al. 2022). We train each model with three different strategies. Data augmentation is also used in a preprocessing procedure to enrich the training dataset, including rotation, cropping, and color distortion. We also evaluate the detection performance on test sets using AP and mAP calculated based on multiple intersections over union (IoU) thresholds. The IoU thresholds range from 0.01 to 1.00 with a step of 0.01. After evaluating detection proposals on all IoU thresholds, we calculate

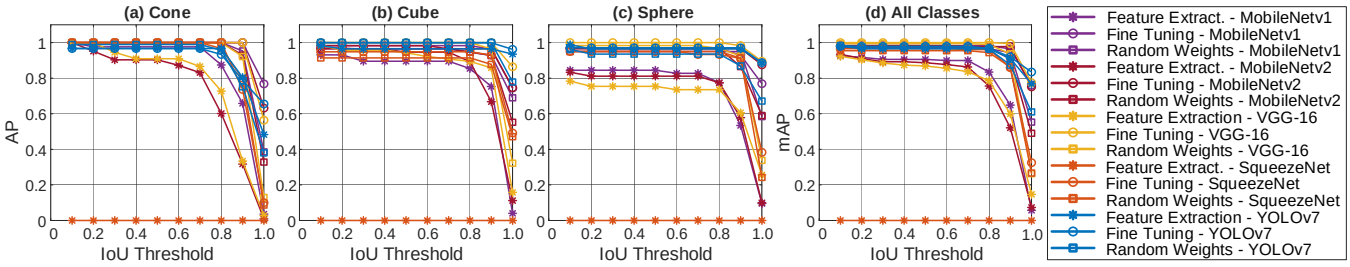

Figure 3: AP for each class and mAP among all classes on MobileNetv1, MobileNetv2 Lite, VGG-16, SqueezeNet, and YOLOv7-tiny models. All models are trained with three different strategies, as described in the Robot Perception section.

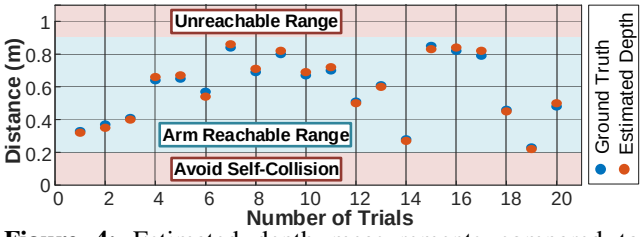

Figure 4: Estimated depth measurements compared to ground truth distances from kinematic transformation.

the AP and mAP for each model: $mAP = (\sum_{k=1}^{c} AP_k)/c$, where c is the number of classes, as shown in Fig. 3.

Results

Test Performance: We test detectors on commodity computers using Intel processors (i.e., Core i3-3217U and HD Graphics 4000). The fine-tuning strategy gives the highest accuracy, while the feature extraction strategy gives the best result except for YOLOv7 (Fig. 3). The feature extraction strategy performs better than the randomly initialized weights strategy regarding YOLOv7. When detecting a sphere, there is a slight difference in the precision between feature extraction transfer and fine-tuning transfer strategies. Fig. 3 also reveals that the source feature extraction in YOLOv7 works well with objects in our target domain, while other models fail to extract features from objects in our target domain. Lastly, YOLOv7 achieves the highest precision at the maximum IoU threshold, while the feature extraction transfer learning strategy does not work for SqueezeNet.

Network	#Params	CPU	Intel GPU	VPU
MobileNetv1	6,883,296	14.87 ± 0.12	19.37 ± 0.23	11.74 ± 0.07
MobileNetv2	3,087,328	17.35 ± 0.19	19.96 ± 0.22	10.15 ± 0.05
SqueezeNet	1,639,648	18.35 ± 0.22	22.53 ± 0.28	14.82 ± 0.11
VGG-16	24,013,744	2.49 ± 0.01	5.15 ± 0.02	2.22 ± 0.005
YOLOv7-tiny	6,652,669	12.59 ± 0.07	21.47 ± 0.22	13.67 ± 0.08

Table 1: Detection performance of models in frames per second (fps) on different hardware configurations (implemented using OpenVINO) with a confidence level of 95%.

Hardware Configuration: We run each detection model on CPU, GPU (Intel), and VPU (Intel NCS2) for $n = 300$ samples and calculate confidence intervals: $CI = \overline{fps} \pm z_{\alpha/2} \cdot (\sigma/\sqrt{n})$, where $\overline{fps}$ is mean frame rate (fps), σ is the standard deviation, and z is the confidence level value of $\alpha = 95\%$. The test is implemented using OpenVINO, which enables ML models to run on Intel onboard GPU. We also test on a VPU interfacing via USB. The onboard GPU outperforms other computing devices in terms of frame rate.

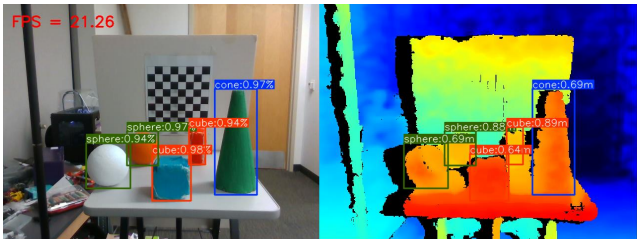

Figure 5: Simultaneous object detection using YOLOv7-tiny and depth estimation on Intel RS D435i camera.

MoblileNetv2 outperforms YOLOv7 when being tested on the CPU but underperforms YOLOv7 on the onboard GPU and VPU. Lastly, the VPU maintains the most stable performance as measured by the variance (Table 1).

Depth Estimation: We use techniques described in Eq. 3 to estimate depth of detected objects (Fig. 5). To generate ground truth distances, we teach the robot to grasp and hold an object in its gripper and then calculate the distance between the robot and that object using the kinematic transformation. Fig. 4 shows that the minimum error is 1.00 cm, the maximum error is 3.00 cm, and the mean error is 1.75 cm.

Demonstration

The demonstration video includes two scenarios: (1) the robot grasps an object while estimating the depth of the detected object, and (2) the robot performs SLAM while following a person using a face recognition module running on the Intel NCS2: `https://youtu.be/q4oz9Rixbzs`.

Conclusions & Future Works

This work proposes a software-hardware framework for mobile cobots focusing on building and optimizing 2D and 3D perception with different commodity hardware. We build the framework on top of multiple ROS distributions, Linux versions, and OpenVINO. For design purposes, the framework can support multiple hardware and find the optimal configurations for input devices/sensors and computing devices. To address task specific object sets, we introduce transfer learning strategies and evaluate them on different computing devices. We then tested our framework on a 7-DOF two-arm Baxter robot with 2D detection and 3D depth estimation. An end-user application is also introduced to facilitate software reusability. We reserve advanced techniques in robot 3D perception, such as segmentation, detection, and recognition, from a point cloud perspective for future work.

Acknowlegment

We would like to thank Christopher Collander for his initial support in this project.

References

Hsiao, Kaijen et al. (2009). "Reactive grasping using optical proximity sensors". In: *2009 IEEE International Conference on Robotics and Automation*. IEEE, pp. 2098–2105.

Hmedan, Belal et al. (2022). "Adapting Cobot Behavior to Human Task Ordering Variability for Assembly Tasks". In: *The International FLAIRS Conference Proceedings*. Vol. 35.

Vice, Jack et al. (2022). "Leveraging Evolutionary Algorithms for Feasible Hexapod Locomotion Across Uneven Terrain". In: *arXiv preprint arXiv:2203.15948*.

Macenski, Steven et al. (2022). "Robot Operating System 2: Design, architecture, and uses in the wild". In: *Science Robotics* 7.66, eabm6074.

Biondi, Alessandro et al. (2019). "A safe, secure, and predictable software architecture for deep learning in safety-critical systems". In: *IEEE Embedded Systems Letters* 12.3, pp. 78–82.

Nazarova, Elena et al. (2021). "CobotAR: interaction with robots using omnidirectionally projected image and DNN-based gesture recognition". In: *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, pp. 2590–2595.

Rendiniello, Angelo et al. (Sept. 2020). "A Flexible Software Architecture for Robotic Industrial Applications". In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. ISSN: 1946-0759, pp. 1273–1276. DOI: 10.1109/ETFA46521.2020.9212095.

Qureshi, Ahmed H et al. (2019). "Motion planning networks". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 2118–2124.

Pinto, Lerrel et al. (2016). "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours". In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 3406–3413.

Veloso, Manuela et al. (2015). "Cobots: Robust symbiotic autonomous mobile service robots". In: *Twenty-fourth international joint conference on artificial intelligence*.

Bellotto, Nicola et al. (2008). "Multisensor-based human detection and tracking for mobile service robots". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 39.1, pp. 167–181.

Chitta, Sachin et al. (2012). "Moveit![ros topics]". In: *IEEE Robotics & Automation Magazine* 19.1, pp. 18–19.

Ichnowski, Jeffrey et al. (2020). "GOMP: Grasp-optimized motion planning for bin picking". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 5270–5277.

Liu, Wei et al. (2016). "Ssd: Single shot multibox detector". In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer, pp. 21–37.

Wang, Chien-Yao et al. (July 2022). *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. en. arXiv:2207.02696 [cs]. URL: http://arxiv.org/abs/2207.02696 (visited on 02/06/2023).

Mao, Jiageng et al. (June 2022). *3D Object Detection for Autonomous Driving: A Review and New Outlooks*. arXiv:2206.09474 [cs]. URL: http://arxiv.org/abs/2206.09474 (visited on 02/03/2023).

Charles, R. Qi et al. (July 2017). "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". en. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, pp. 77–85. ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.16. URL: http://ieeexplore.ieee.org/document/8099499/ (visited on 02/01/2023).

Kim, Jin Ho et al. (2013). "Design of a seamless gateway for Mechatrolink?" In: *2013 IEEE International Conference on Industrial Technology (ICIT)*, pp. 1246–1251. DOI: 10.1109/ICIT.2013.6505852.

Dang, Tuan et al. (2022). "ioTree: a battery-free wearable system with biocompatible sensors for continuous tree health monitoring". In: *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, pp. 769–771.

Zhuang, Fuzhen et al. (Jan. 2021). "A Comprehensive Survey on Transfer Learning". In: *Proceedings of the IEEE* 109.1. Conference Name: Proceedings of the IEEE, pp. 43–76. ISSN: 1558-2256. DOI: 10.1109/JPROC.2020.3004555.

Howard, Andrew G. et al. (Apr. 2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. en. arXiv:1704.04861 [cs]. URL: http://arxiv.org/abs/1704.04861 (visited on 01/28/2023).

Sandler, Mark et al. (June 2018). "MobileNetV2: Inverted Residuals and Linear Bottlenecks". en. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Salt Lake City, UT: IEEE, pp. 4510–4520. ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00474. URL: https://ieeexplore.ieee.org/document/8578572/ (visited on 01/28/2023).

Iandola, Forrest N. et al. (Nov. 2016). *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡0.5MB model size*. arXiv:1602.07360 [cs]. URL: http://arxiv.org/abs/1602.07360 (visited on 02/08/2023).

Simonyan, Karen et al. (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556*.