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/tuandang/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.
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.
Transfer Learning Definition: A domain is defined by \( D = \{ X, P(X) \} \) where \( X = \{ x_1, x_2, \ldots \} \in \mathcal{X} \) with \( \mathcal{X} \) representing the feature space, and \( P(X) \) its marginal distribution. Let \( T = \{ Y, P(Y|X) \} \) 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 \( P(Y|X) \) in target domain \( D_t = \{ X_t, P(X_t) \} \) using knowledge in the source domain \( D_s = \{ X_s, P(X_s) \} \) and source learning task \( T_s = \{ Y_s, P(Y_s|X_s) \} \). Let \( P(Y|X) = f(X, \beta) \) where \( f \) is the task function. The minimizer for the trainable parameters, \( \beta \), is written in terms of the loss function, \( L(\cdot, \cdot) \), and the task function, \( f \), as follows:

\[
\argmin_{\beta} \sum_{X} L [ f(X, \beta), Y ]
\] (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 \( \beta^D \) and \( \beta^F \) are parameters for detection and feature extraction, respectively. The analogous minimizer for \( \beta^D \) and \( \beta^F \) is:

\[
\argmin_{\{ \beta^D, \beta^F \}} \sum_{X} L \left[ \left( f^D \circ f^F \right) (X_i, \beta^D, \beta^F), Y_i \right]
\] (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 (\( Y_s \neq Y_t \)) 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 \( (\beta^D, \beta^F) \to (\beta^D_s, \beta^F_s) \), where \( \beta^D_s \) and \( \beta^F_s \) are results from source task functions, and fine-tune \( (\beta^D_t, \beta^F_t) \) 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 \( \beta^F_s \to \beta^F_t \), and train \( \beta^D_t \) 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:

\[
D = (w \times h)^{-1} \left\{ \sum_{i=x_0-w/2}^{x_0+w/2} \sum_{j=y_0-h/2}^{y_0+h/2} d(i,j) \right\}
\] (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...
the AP and mAP for each model: $mAP = (\sum_k AP_k)/c$, 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. 4 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.

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

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

Hardware Configuration: We run each detection model on CPU, GPU (Intel), and VPU (Intel NCS2) for $n = 300$ samples and calculate confidence intervals: $CI = \bar{fps} \pm z_{\alpha/2} \cdot (\sigma/\sqrt{n})$, where $\bar{fps}$ is mean frame rate (fps), $\sigma$ 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.

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.

Acknowledgment

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


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


