ApproxRobotics: The Cost and Accuracy Tradeoff for Small Mobile AgRobots

Anonymous Author(s)
Affiliation
Address
email

Abstract: Autonomous robots are increasingly relying on high-dimensional visual information for perception, planning, and control. The deep neural network pipelines that perform inference tasks on these robots require significant computational resources, which increases the robot energy consumption and the cost. This has been a barrier in adopting learning based methods in cost-constrained domains, such as agriculture. In this paper we show that structured pruning on neural networks can enable agricultural robots to employ lower-cost computational hardware, without losing task robustness in visual navigation and visual phenotyping tasks. We expose key trade-offs between computational cost and prediction accuracy in the perception module of an autonomous navigation stack in a production agricultural robot. Our key finding is that, for closed-loop control systems used in robots, it is often possible to relax the accuracy of CNNs used for computer vision without significantly hurting the end-to-end task outcomes. We show that computational approximations enable us to deploy a state-of-the-art vision-based autonomous navigation pipeline and a real-time video analytics task on a single resource-constrained Raspberry Pi4. Our results show that it is possible to use learning-based control for small mobile robots using low-cost compute hardware.

1 Introduction

Autonomous robots are increasingly relying on visual information for perception, planning, and control [1, 2, 3, 4, 5, 6, 7]. Visual data is very high dimensional, yet, with advances in deep learning, many robotics applications extract actionable information through vision. However, a major challenge is that deep learning inference is highly computationally expensive, and can be energy intensive. Large robots such as autonomous cars can afford to have much larger computational payloads, since they are powered by carbon fuels or large batteries and have sophisticated cooling systems. In contrast, small battery-powered autonomous robots such as those used in agriculture, mining, or remote area exploration, have much tighter size, weight, power, and production cost constraints [8, 9]. For such robots, optimizing the computational requirements for deep learning powered visual inference tasks is crucial. A key open question, which we address in this paper, is what computational optimization can enable small battery-operated mobile robots to use low-cost and low-power computational hardware. In particular, recent advances in neural networks have led to a number of different neural network optimization strategies with different impacts on network accuracy, and we aim to understand the trade-offs between reduced computational costs of inference in neural networks and task robustness.

This paper demonstrates through extensive field experiments that computational approximations in convolutional neural network (CNN) models can enable the use of low-cost computational hardware in cost-constrained domains, such as agriculture. In particular, we present an empirical field study of tradeoffs between the end-to-end accuracy of computation and its cost (time and energy), in the context of mobile agricultural robots that perform closed-loop visual-guided navigation and vision-based online object detection and tracking tasks. The key insight emerging from our work is that, for closed-loop control systems used in robots, it is often possible to relax the accuracy of CNNs used for computer vision without significantly hurting the end-to-end task outcomes. In particular, we show that structured weight pruning (a technique that drops low-weight filters) of

Submitted to the 5th Conference on Robot Learning (CoRL 2021). Do not distribute.
neural network models [10, 11, 12] (used for visual perception) can be used to trade off accuracy for computation time improvements in closed-loop visual navigation tasks, even making it possible to run both navigation and a real-time object detection task concurrently (three CNNs in total) on a single Raspberry Pi 4, without significantly hurting system robustness.

We evaluate the impact of computational approximations in the context of a production agricultural robot (AgRobot), TerraSentia, obtained from EarthSense [13]. This commercial robot is used for autonomous navigation through corn fields for high-throughput phenotyping and a variety of production agriculture tasks. Our study focuses on the phenotyping task of stand counts for corn (Zea mays). Since each corn plant yields a corn ear of predictable size, corn stand count is highly correlated with yield, and as such this phenotype is commonly measured by farmers and crop-scientists.

This involves two visual tasks on the robot, 1) autonomous navigation using a state-of-the-art learning-based visual guidance system, CropFollow [14] with the front camera, and 2) counting corn stems using the side cameras with a detection and tracking algorithm similar to [15, 16]. Our work investigates how well approximation techniques, and in particular structured pruning, can be used to trade off model accuracy for computation time improvements (in particular, increased frames-per-second) in the neural network models used for heading and distance prediction (navigation) and object detection (stand counting). These computation time improvements facilitate using lower-cost compute hardware, performing additional computations on the same hardware, or both.

While many techniques for network pruning have recently emerged [12, 17, 18, 19, 20, 21, 22] (see Related Work in the supplementary materials for further details), they do not answer important application-level usability questions:

1) Does pruning of CNNs used as part of large real-world applications provide significant compute time improvements for the overall application?

2) Is it acceptable to relax some accuracy to improve compute performance while avoiding observable impact on the end-to-end quality of the task, e.g., without losing control robustness or sacrificing too much accuracy in an analytics task?

Our study answers both questions positively in the context of AgRobots. Our optimization objective is to maintain end-to-end task robustness (i.e., collision-free navigation), which is in contrast with most existing approaches to neural network pruning, which are aimed at retaining the computational accuracy of the individual CNN component. Those conservative approaches to pruning limit the achievable computational gains as they do not leverage the inherent robustness of real-world applications, which may have in-built compensation (e.g., closed-loop autonomous systems).

With our approach of trading off accuracy without losing robustness, we are able to drive far more aggressive computational performance improvements, ranging up to 15x (on CPU). These performance speedups allow us to perform the entire navigation pipeline and real-time video analytics, including two CNNs, Bayesian sensor fusion Model Predictive Control, and an object detection algorithm on a low-end $35 Raspberry Pi4 [23]. This compares favorably with the $876 Intel NUC [24] currently used on commercial TerraSentia robots, and with the $59 Jetson Nano, the cheapest device we found to deliver necessary performance without approximations. Moreover, the Raspberry Pi4 requires 30% lower peak power than the Jetson Nano (7W vs 10W). Alternatively, the Jetson Nano host CPU is identical to the Raspberry Pi4, so all of these computations for both tasks could be performed on the Jetson’s CPU, leaving the GPU free for other computationally intensive tasks.

Key Results: The key results of our work are as follows.

• We show that pruning the CNNs used in the visual perception system of an AgRobot helps achieve the minimal required FPS for collision-free navigation using only very low-cost compute hardware (a $35 Raspberry Pi4), making it a feasible choice for mobile robots.

• We find that the CNN-based autonomous navigation control in the evaluated AgRobot is robust to infrequent mispredictions. We also identify pruning settings that introduce large prediction errors that greatly impact the navigation quality, resulting in collisions.

• By relaxing CNN accuracy constraints for corn stand counting, again using network pruning, we show that multiple compute-intensive tasks including the navigation pipeline and vision-based corn stand counting can run concurrently on a single Raspberry Pi4.

Significance: Our paper presents a much-needed study of computational approximations on end-to-end visual navigation and phenotyping tasks for autonomous field robots. Such studies are sorely
missing in robotics, but are necessary because Moore’s law is ending at the same time that roboticists are increasingly using computationally expensive visual learning methods for autonomy [25].

As such, these results have major implications both for low-cost AgRobots, and for other robotics applications. In agriculture, low-cost autonomous robots can enable several applications that are currently not possible with larger tractors, such as scouting for disease and pests, herbicide-free mechanical weeding, and carbon sequestration through in-season cover-crop planting. Low cost (e.g., under $500 per robot) is essential because agriculture is an extremely cost-sensitive activity, and our paper establishes that a key component of cost (compute hardware) can be minimized dramatically.

The implications of our work extend also to other resource-constrained applications in robotics. Small, battery-powered mobile robots using visually-guided outdoor navigation systems have important uses in forestry, delivery, mining, search and rescue, scientific exploration, defense surveillance, and others. The successful compute optimizations and cost reductions demonstrated in our study would be directly applicable to any such applications, as well.

2 Background: Robot System Design

2.1 Hardware Setup

TerraSentia robot is a compact, light weight robot that maneuvers between crop rows for commodity crops such as corn and soybean. It hosts two 2D horizontal-scanning LIDARs which are not evaluated in this work, as vision-based navigation is shown to work better (fewer collisions) in [14]. For vision-based navigation, it uses a forward-facing camera with 720p resolution at 30 FPS for row following using the algorithm in [26]. It hosts two side cameras (left and right) used for high-throughput phenotyping. An embedded 6 DoF Inertial Measurement Unit (IMU) gathers angular velocity and acceleration measurements. TerraSentia uses an Intel NUC 10i7FNH as the primary computer, and a Raspberry Pi3 for the lower-level control logic (speed and the angular rate controller). The GPU on the Intel NUC is used for compute-intensive tasks, including the CNNs in the navigation pipeline [26] and any real-time data processing tasks. The Intel NUC 10i7FNH costs $876 MSRP [27]. In our work, we want to replace the Intel NUC with a single Raspberry Pi4 board that costs $35. Raspberry Pi4 includes a quad core cortex-A72 CPU at 1.5Ghz and 2 GB RAM.

2.2 Autonomous Navigation Software Setup

Figure 1 shows the workflow of the vision-based navigation pipeline [26].

Perception Module. We utilize the CropFollow visual navigation system developed in [14] as the primary perception module. The module takes as input $320 \times 240$ RGB images (resized from 720P) and uses two different CNNs (both ResNet-18) to estimate the robot heading $\theta$ and relative distance $d$, respectively. $d$ is the ratio of the left distance (from center of robot to left crop row) to the total distance between adjacent rows, i.e., $d = dL/(dL + dR)$.
**IMU Fusion with Extended Kalman Filter.** An Extended Kalman Filter (EKF) is used to fuse CNN predictions with inertial measurements from IMU. The Kalman filter reduces the effect of uncertainties from a single source (vision or IMU), thereby reducing the probability of abrupt control variations (e.g., a sudden large turn). The Kalman filter takes as input 1) CNN distance prediction, 2) CNN heading prediction, 3) robot’s linear speed from the wheel encoders, 4) angular speed from the IMU, and 5) the robot state at the previous time step $s_{k-1}$, to compute the current state $s_k$. This state includes a heading estimate and a distance estimate.

**Model Predictive Controller.** A non-linear Model Predictive Controller (MPC) receives the EKF estimates for heading and distance and solves a constrained optimization problem (with curvature radius as constraints) to compute angular velocity commands that keep the robot following the reference path. In “row following,” the reference path is a straight line through the center of the crop lane.

### 2.3 Real-time Stand Counting Task

Real-time analytics tasks in digital agriculture are relevant for crop breeders, and also farmers that need to closely monitor crop health and crop yield [28]. Performing these tasks in real-time has the advantage of 1) delivering results faster to users, 2) reducing data transfer and storage costs for large data recordings to be analyzed offline, and 3) reduced cloud/server costs for offline analysis [29]. One such task we evaluate as an example is corn stand counting: using a stream of side camera images to count corn stands in crop rows. This involves object detection, which is extremely expensive for resource constrained systems [30, 31] and challenging to execute in real-time.

The stand counting algorithm we evaluate, **standc-track**, uses an object detector for detecting individual corn stands in image frames, and a SORT object tracker [32] that keeps track of specific corn stand instances across frames to avoid counting duplicates. This approach is similar to other methods for object tracking [16, 33]. In Appendix §A.5, we compare **standc-track** with a less expensive but less accurate algorithm, **standc-stride**, which presents different tradeoffs.

### 3 Approximation: Model Pruning

In the literature, many generic and domain-specific approximation techniques have been proposed for optimizing neural network computations with (potentially) small loss in inference accuracy. Examples include weight pruning (dropping/skipping weights or filters) [34, 35, 10, 11], perforated convolutions [36, 37] (skipping and interpolating some output computations), integer quantization [35] (INT8, INT4) and others. While these approaches cannot give theoretical guarantees on the accuracy, they have shown promising empirical tradeoffs for neural networks examined in isolation.

In this work, we apply weight pruning to the convolutional neural networks used in the navigation pipeline and corn stand counting task, because weight pruning has proved especially effective in reducing computational cost with small impact on accuracy. Weight pruning can be categorized into two major types: 1) **unstructured pruning** [34, 35], which removes individual weights that are deemed less important to the overall computational accuracy (e.g, low-magnitude values), and 2) **structured pruning** [10, 38], which removes groups of contiguous weights, for instance, entire filters and channels from convolution layer weights. Unstructured pruning provides much higher reduction in model sizes (up to $13 \times$ in prior work [35]) than structured pruning (up to $4.5 \times$ in prior work [38]) since it allows removing weights at a finer granularity. However, prior work [39] has shown that unstructured pruning introduces unpredictable sparsity in the underlying tensor computations (e.g., convolutions, matrix multiplications), which can lead to slowdown on highly parallel architectures such as GPUs and CPU vector units (Cortex-A72 vector units in Pi4 [40]) not well suited to irregular computational patterns. In contrast, structured pruning preserves dense computation patterns, which better utilizes parallel hardware and provides real speedups on GPUs and CPUs [41, 42, 43, 44].

In this work, we adopt the iterative structured pruning algorithm proposed by Renda et al. [11], shown in Algorithm 1. We prune convolution filters with lowest L1-norm values; approach proposed by Li et al. [10]. The **PruneModel** routine takes as input a trained model, training data (for retraining), number of epochs to retrain, prune fraction (percentage of filters to prune), and the total number of pruning iterations (numPruneLevels). Each pruning iteration (outer most loop) 1) removes a pruneFraction fraction of filters with lowest L1-norm from each convolution
Algorithm 1: Structured Pruning Algorithm

Inputs:
- model: target model
- trainData: training data
- numEpochs: epochs to use for training in each iteration
- pruneFraction: percentage of filters to remove in each iteration
- numPruneLevels: number of pruning iterations

Output: prunedModels: List of pruned models

Function PruneModel(model, trainData, numEpochs, pruneFraction, numPruneLevels)

Set prunedModels;
for i = 1 to numPruneLevels do
  foreach convLayer ∈ model do
    foreach convFilter ∈ convLayer do
      filterLNorm = computeLNorm(convFilter);
      filterList = (convFilter, filterLNorm);
      end
      minWeightFilters = getLowestFilters(filterList, pruneFraction);
      remainingFilters = filterList \ minWeightFilters;
      newConvLayer = setLayerFilters(convLayer, remainingFilters);
      model = updateModelLayer(model, newConvLayer);
      prunedModels = model;
  end
end
return prunedModels;

Datasets and Model Training. Both ResNet-18 and SqueezeNet-v1.1 for heading and distance prediction are pretrained on ImageNet and fine-tuned on 25K corn images with heading and distance ratio labels generated from manually annotated vanishing lines [14]. 20K images are used for training and 5K images for validation. The object detector SSD-MobileNet-V2 is pretrained on the COCO [49] dataset and fine-tuned on 1K corn-only images with manually labelled bounding boxes. With a 4-to-1 split, 800 images are used for training and 200 for validation. In the evaluation of stand counting algorithm (which invokes object detection), we use a dataset of 60 videos, each labelled with the total stand count (a number). Appendix §A.1.1 includes more details on model training.

Model Pruning. We apply the pruning technique in Algorithm 1 with 20 iterations each removing 20% filters. We refer to the output model of each iteration as a prune level and label these from 1 to 20 inclusive (0 is the unpruned baseline). We use the same prune level for both heading and distance models when evaluating the navigation quality, because both models run concurrently with their collective output fed into the EKF, and running one model faster than the other gives no benefit.

Hardware Setup. We run the models on a Raspberry Pi4 to evaluate whether pruning enables the use of low-cost hardware. To understand the impact of FPS (image frames processed per second) changes, we need a device that provides high-enough FPS, for which we manually reduce the FPS by adding time delays using sleep() calls. For these experiments, we use an NVIDIA Jetson Nano (4GB memory) [50], which provides approximately 25 FPS on the baseline ResNet-18 models for heading and distance prediction simultaneously.

Field Setup. We perform our row-following experiments on production corn fields with each navigation run covering 360 meters. A human observer measures the number of collisions with the corn row boundaries that need manual intervention – there are no collisions with other robots. We evaluate 5 navigation speeds: 0.5 m/s, 0.6 m/s, 0.8 m/s, 1.0 m/s, and 1.2 m/s.

Frameworks. All models are trained and pruned in PyTorch. For CPU execution on Raspberry Pi4, we use the ONNX runtime [51], and for GPU execution on Jetson Nano, we use TensorRT [52].

5 Evaluation

We empirically evaluate the following research questions: RQ1: Does pruning help increase the FPS (frames processed per second) of the vision models, what is the impact on model accuracy, and how
does this tradeoff differ between ResNet-18 and SqueezeNet? **RQ2:** What are the minimum FPS requirements from the vision models for collision-free navigation, and does pruning help achieve the minimum FPS on the Raspberry Pi4? **RQ3:** How do ResNet-18 and SqueezeNet pruned models compare in terms of delivering collision-free navigation, given their different tradeoffs? **RQ4:** Can pruning object detection models used in stand counting facilitate real-time execution, concurrently with navigation? **RQ5:** How does prediction error and FPS impact navigation quality? Before addressing these questions, we first establish the minimal FPS required for collision-free navigation.

### 5.1 Establishing Minimal FPS Requirements

For a vision model processing \( f \) frames per second and providing collision-free navigation, we declare \( f \) to be the *minimal FPS* if all lower FPS values lead to collisions. In practice, we check 0.5 FPS below \( f \) to confirm \( f \) as the minimal FPS and consider this to be sufficient precision. At navigation speed 0.5 m/s, 2 FPS is sufficient for collision-free navigation. For speeds 0.6 m/s, 0.8 m/s, 1 m/s, and 1.2 m/s, the minimal FPS are 2.5, 3, 4, and 5, respectively. Overall, minimal FPS increases with navigation speed. The rationale is that higher speeds require faster control decisions from the MPC, which requires faster heading and distance estimates from the EKF, subsequently related to the FPS of the CNN predictions (as shown by the navigation workflow in Figure 1).

Figure 2 shows how increasing FPS improves the navigation quality. We perform this experiment on the Jetson Nano with the baseline ResNet-18 models and navigation speed set to 1.0 m/s. At 2 FPS (lower than minimal FPS), the robot has 12 collisions over the 360-meter run. At 3 FPS, the number of collisions reduces to 1. With FPS \( \geq 4 \), the navigation stabilizes and results in no collisions.

### 5.2 Evaluating Prune Levels on Validation Set – RQ1

Figure 3 presents the tradeoff space of prune levels (level 0 for baseline) for the heading prediction model; distance prediction models show a similar trend (§A.3). The tradeoff is between performance measured in FPS and the 95th-percentile loss (defined in Appendix §A.1.2) on validation set. Losses for the heading model are measured in degrees. Therefore, points to the upper left are better tradeoff points. We measured the compute performance by running one neural network alone on the Raspberry Pi4 in isolation (no other task running) – the results with running the complete CropFollow algorithm (including both CNN models, EKF and MPC) are shown later.

SqueezeNet is faster but significantly less accurate than ResNet-18, thus covering a different part of the tradeoff space. The baseline version of heading prediction with SqueezeNet has 95th-percentile loss 14% higher than its ResNet-18 counterpart. Similarly, baseline distance prediction SqueezeNet (Appendix §A.3) has 11% higher loss than its ResNet-18 counterpart. On the other hand, baseline SqueezeNet is 3.3× faster than baseline ResNet-18 on Pi4 (5.5 FPS v.s. 1.7).

Figure 3 shows that pruning provides significant speedups in FPS, at some cost in accuracy. With each higher prune level (see point labels), both the loss and FPS increase. At the highest prune level (20), ResNet-18 provides 43 FPS (25× increase over baseline) while SqueezeNet provides 132 FPS. When considering accuracy loss, ResNet-18 provides a better overall tradeoff. For instance, ResNet-18 heading model at prune level 20 has higher FPS (43 FPS) and also (slightly) lower loss (7.2 degrees) than SqueezeNet prune level 6 (36 FPS and 7.4 degrees). This shows, somewhat surprisingly, that pruning the larger, more accurate (but initially more expensive) model can achieve better frame rates and better accuracy than pruning the smaller model.

### 5.3 Using Pruned Models to Achieve Minimal FPS on Pi4 – RQ2

With the full navigation pipeline running, without pruning, the baseline ResNet-18 models deliver 0.9 FPS, which is significantly less than the minimal FPS requirement of 2 FPS at 0.5 m/s (§5.1).

Here, we address two open questions: 1) are the computational time reductions from pruning shown in Section 5.2 sufficient to achieve the minimal FPS for collision-free navigation using Pi4, and 2) does the additional accuracy loss affect navigation quality.

Figure 4 shows how varying the prune levels impacts the FPS and the number of collisions at 1.2 m/s, using ResNet-18. The minimal required FPS is 5 FPS at 1.2 m/s (purple line). The red line (FPS at different prune levels: right axis) shows that the FPS increases with increasing prune levels while the blue line (number of collisions: left axis) steadily decreases. At prune level 9, the FPS surpasses
5.4 Comparing ResNet and SqueezeNet Pruned Models – RQ3

We evaluate how different prune levels of ResNet-18 and SqueezeNet compare in terms of navigation quality. This is important because SqueezeNet is much smaller and faster than ResNet-18, but also less accurate, and model pruning directly affects this tradeoff. Figure 4 shows for ResNet-18 that the navigation quality increases as the FPS surpasses the minimum, and does not decrease at higher prune levels (13 and 20), even though the prediction error increases (Figure 3). In comparison, the baseline SqueezeNet (Figure 5) provides an FPS of 2.0, which is insufficient for stable navigation and results in 11 collisions. Then, SqueezeNet provides 5.9 FPS at prune level 4 and 10.1 FPS at prune level 6, both surpassing the minimum and resulting in no collisions. However, unlike ResNet-18, prune level 9 that delivers sufficient FPS (18.5 FPS) still results in entirely unstable navigation with 9 collisions. This is because the higher prune level results in much higher errors (see Figure 3), causing the increased collisions. This result shows the importance of tuning the prune levels to find a suitable tradeoff between compute time and prediction accuracy. In Section A.2, we further discuss how high prediction errors in the vision models can lead to poor quality control decisions.

5.5 Navigating with Real-time Stand Counting – RQ4

Since model pruning reduces the compute requirements for navigation, we examine whether pruning also facilitates running an instance of stand counting concurrently with the navigation pipeline on Pi4. We set the navigation speed to 0.5 m/s, as the evaluation dataset for stand counting is recorded at 0.5 m/s. The accuracy of stand counting is captured in percentage error (PE) of corn counts over this dataset, $PE(pr, gt) := \frac{1}{N-1} \sum_{i=0}^{N-1} \frac{|pr_i - gt_i|}{gt_i} \times 100\%$; where $N$ is the number of videos, $pr$ is the predicted stand count, and $gt$ is the ground-truth stand count. In this experiment, we use 10% error as an example threshold for accuracy constraint. Higher thresholds may allow increasing flexibility, and Appendix §A.5 shows how relaxing the acceptable error to 12% opens up further

Figure 2: Impact of increasing FPS on navigation stability at 1.0 m/s measured by collisions, using baseline ResNet-18 heading and distance models.

Figure 3: 95th percentile loss (in degree) and FPS of ResNet-18 and SqueezeNet heading models at different prune levels (annotated on each point).

Figure 4: ResNet-18 performance (FPS) and navigation quality (collisions) across different prune levels.

Figure 5: SqueezeNet performance (FPS) and navigation quality (collisions) across different prune levels.

Figure 6: Percentage error and FPS of stand counting across different prune levels.

Figure 7: Prune levels of vision model (ResNet) in navigation and detector stand–track. “Feasible” means <10% stand count error and collision-free navigation.
optimization opportunities. We find that standc-track requires a minimal FPS of 16 (unrelated to FPS constraints for heading and distance models), below which the stand counting accuracy is irrecoverable. At a speed of 0.5 m/s, FPS < 16 means that consecutive image frames move more than the distance between the detected bounding boxes (a detected box around each corn stand), making it difficult for the SORT object tracker to correctly track corn stands across frames.

Figures 6 presents the percentage error and FPS of the standc-track algorithm with object detection model of different prune levels. Generally, both FPS and error increase with higher prune levels. However, due to the randomness in model retraining, error is not monotonically increasing, e.g., level 16 has lower error than level 15. Prune level 16 runs at 16.1 FPS while giving 8.94% percentage error, which satisfies both the FPS and the accuracy requirement.

Tradeoff Space of Stand Counting and Navigation. Figure 7 illustrates the overall tradeoff space generated by prune level options in navigation and stand counting. The figure focuses on ResNet-18; SqueezeNet shows a similar trend (§A.4). Each point represents a pair of choices for the two prune levels, and leads to one of three outcomes: 1) Low FPS, when either navigation or stand counting doesn’t meet its FPS requirement; 2) Low Accuracy, when either component doesn’t meet its accuracy requirement; 3) Feasible, when there is no violation in accuracy or FPS. FPS is measured with standc-track and navigation pipeline running together. The accuracy of stand counting is shown above in this subsection, while for the navigation accuracy we use results from Section 5.3.

Given the heavily compute-constrained hardware, the feasible region in the figure is far away from the origin (baseline). With navigation and stand counting running on the same hardware, navigation FPS reaches the threshold at prune level 13 which is 10.2× faster than the baseline. These results show that aggressive approximations are needed to enable both components to run concurrently on low-end hardware. The significance is that such approximations enable a much lower cost and lower energy design point for mobile robot applications.

5.6 Analyzing Navigation Quality Across Pruning Settings – RQ5

Our evaluations show that both low prediction accuracy and low FPS negatively impact navigation quality. In Appendix §A.2 we include a detailed analysis that shows how high prediction errors and insufficient FPS from vision models impact control decisions that lead to robot collisions.

6 Discussion and Conclusions

Alternate Choices for Compute Hardware. For vision-based navigation, the $59 Jetson Nano (2GB) and the $75 Intel Myriad are potential alternatives to Pi4. For vision models, Jetson Nano can provide 25 FPS (on GPU) and Intel Myriad 10 FPS. Intel Myriad needs a host device which adds to the overall compute cost, making it an expensive choice. In contrast, Jetson Nano is a standalone device with on-board CPU and GPU. Since it provides sufficient FPS on the baseline models, pruning the models is not strictly necessary for running the navigation pipeline. However, pruning models running on the Nano is still valuable since it allows the models to run on its CPU alone, freeing up the far-more-powerful GPU for heavy-weight workloads. Without such workloads, the Raspberry Pi4 is our hardware of choice because it has significantly lower cost ($35) and power consumption (7W) compared to Nano (10W) [53].

Implications for Future Research. Our study shows that exploiting accuracy-performance trade-offs offers significant opportunities for improving the cost and energy efficiency of compute hardware in autonomous robot navigation systems and their application workloads. We believe these and similar techniques based on relaxing accuracy constraints can also be valuable in related application domains, such as autonomous drones, autonomous vehicles, and other similar mobile robots.

References


