IntPred: Flexible, Fast, and Accurate Object Detection for Autonomous Driving Systems

Hamid Tabani
Barcelona Supercomputing Center
Barcelona, Spain
hamid.tabani@bsc.es

Matteo Fusi
Barcelona Supercomputing Center
Barcelona, Spain
matteo.fusi@bsc.es

Leonidas Kosmidis
Barcelona Supercomputing Center
Barcelona, Spain
leonidas.kosmidis@bsc.es

Jaume Abella
Barcelona Supercomputing Center
Barcelona, Spain
jaume.abella@bsc.es

Francisco J. Cazorla
Barcelona Supercomputing Center
Barcelona, Spain
francisco.cazorla@bsc.es

ABSTRACT

Deep Neural-Network (DNN) based Object Detection is one of the most important and time-consuming stages of Autonomous Driving software in cars. In non-critical domains, the performance and energy requirements of object detection can be reduced at the cost of accuracy in the detected objects. This is not the case in a critical domain like automotive, for which a delicate balance between performance/energy overheads and accuracy of object detection must be found. We propose IntPred to achieve such a balance by leveraging on the fact that, with high frame rates, objects do not move significantly across frames. IntPred tailors object interpolation for the case of object detection in autonomous driving frameworks, in line with approaches devised for other domains, thus heavily reducing the performance requirements of full-fledged DNN-based object prediction. IntPred results in comparable accuracy to the original object detection, while saving more than 70% of the computations. The latter allows using lower-performance and cheaper platforms resulting in saving energy and reducing heat dissipation: for instance, in an NVIDIA Jetson TX2 platform, specific for autonomous driving systems, our technique increases the frame processing rate by 4.6x. IntPred also allows consolidating additional applications onto the same platform.

KEYWORDS

Autonomous Driving Systems, Object Detection, Safety-critical Systems

1 INTRODUCTION

Investment on Autonomous Driving (AD) systems for the automotive domain has grown significantly in the last years, and the trend is expected to hold in the foreseeable future [9, 19]. Interestingly, as the driving automation level increases, so does AD system's criticality, which translates into specific requirements on the technology that can be deployed in AD systems. For instance, AD must work timely, i.e. in a real-time fashion in its interaction with the environment; and second, the quality of the functionality provided cannot drop below certain thresholds. Failing to accomplish with these requirements may cause the AD system to misbehave, potentially causing serious damage to the environment or casualties.

AD implements advanced functionalities to identify vehicle surroundings, locate its position with high precision, define a route to follow, and generate control commands (e.g., accelerating, braking, and steering) to drive the vehicle through a specified route to get to its destination. Those AD functionalities are provided by software applications that implement sophisticated and complex algorithms. Furthermore, applications process huge amounts of data coming from sensors (e.g. video cameras, short- and long-range radars and laser) in order to be able to make accurate decisions in all driving conditions [5, 7].

The impressive recent improvements in deep learning technology have made it be the first choice in a variety of areas, and AD is not an exception to that [10, 13]. Improved results offered by deep-learning based algorithms come, however, at the cost of increased computing performance demands. The latter are covered, under the stringent time constraints of the automotive domain, by executing AD software on powerful hardware [6, 7, 11, 18], e.g. high-end GPUs despite the existing challenges [8, 16, 24]. Naturally, these hardware devices consume significant amounts of energy, which recent studies show can reduce the driving range (i.e. autonomy of cars) more than 10% [15]. This calls for hardware and software solutions to reduce the performance and energy requirements of AD software, without reducing the accuracy of the AD system due to its high criticality.

From all AD software functionalities, our results on several widely-used state-of-the-art AD systems confirm the results of previous studies [15] showing that *object detection* is one of the most compute and energy-intensive modules. In this work, we focus on the camera-based object detection module of industrial autonomous driving frameworks such as Apollo [5] and Autoware [1], which operates on multiple cameras at a high frame rate (i.e. >25 frames per second). Apollo object detection (Apollo-OD) is a modified version of YOLO (You Only Look Once) [20, 22], a state-of-the-art unified model and deep learning-based approach widely used in many areas. YOLO stages, that operate in a pipelined fashion, include *fetching* the frame data, the DNN-based object *prediction* stage, and *displaying* the outcome of the processing.

Contribution. We contribute with a proposal to reduce object detection computing demands builds on the observation that, with a camera capturing 20+ frames per second, objects move slowly across consecutive frames. This is in contrast to video frames in other domains such as movies, in which by the change of one scene to another, adjacent frames can be totally different. Furthermore,

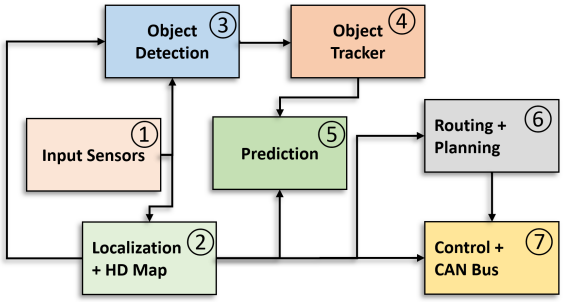

Figure 1: A state-of-the-art AD system software pipeline.

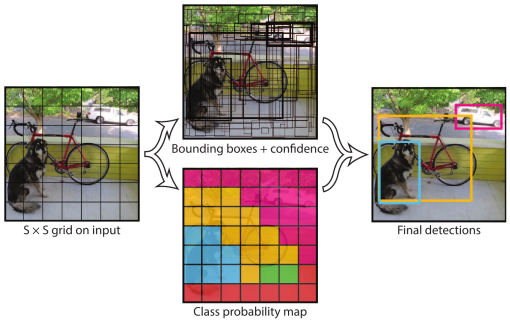

Figure 2: YOLO model [22].

the direction of objects' movement is quite predictable from one frame to the next one.

We start by making a performance analysis when running Apollo-OD on a high-end system with a powerful NVIDIA 1080 Ti GPU [3] and on embedded low-power NVIDIA Jetson TX2 and Xavier platforms [4, 11] specially meant for AD systems in automotive. In all the mentioned platforms, we show that the predict stage consumes significantly more computing power than the other stages.

We propose implementing frame interpolation, already devised for other domains [14, 26], to achieve accurate prediction-based object detection in AD systems significantly reducing the amount of computation required. In particular, due to the similarity of consecutive frames and the longer duration of the predict stage, our approach uses base frames to locate and detect the object inside adjacent frames with an interpolation prediction (IntPred) algorithm rather than applying the compute-intensive DNN-based prediction stage.

Our results show that IntPred reduces up to 70% of the total amount of computation required and allows to increase the frame rate up to 4.6x. Although we designed IntPred in such a way that it has negligible impact on accuracy, it allows to gradually trade off performance demand (and so power) and accuracy. This allows the designer to find the desired balance between both metrics.

There have been various proposals to improve the performance of object detection using different schemes, including reducing the DNN model parameters, training smaller DNN models, reducing the resolution of the video, etc. Our work is orthogonal to these other solutions and it can be used together with them. In particular, we leverage interpolation-based solutions devised for other domains with similarities to object detection in AD systems in the input data to be processed [14, 26]. Furthermore, our work can be applied independently from the implementation methods (e.g,. based on Tensorflow, Caffe, etc.).

The rest of the paper is organized as follows: Section 2 provides background on a state-of-the-art object detection algorithm. Section 3 presents our analysis of the baseline object detection module in an AD system and our proposed scheme. Section 4 evaluates our proposal. Section 5 summarizes the related work and Section 6 concludes the paper.

2 OBJECT DETECTION

The architecture of AD software comprises several modules as shown in Figure 1:

(1) *Input Sensors* including different types of cameras, LiDAR, Radar, and GPS sensors.
(2) *Localization* leverages information received from different sensors to estimate the location of the vehicle precisely.
(3) *Object Detection* identifies the location and the boundaries of objects within an image that belong to a given set of classes with a probability for each object.
(4) *Object Tracking* is responsible to track those objects identified by the *Object Detection*.
(5) *Prediction* module anticipates the future motion trajectories of perceived obstacles/objects.
(6) *Routing* module tells the vehicle how to reach the specified destination. *Planning* plans the spatiotemporal trajectory for the vehicle to take. Based on the outcome of these modules,
(7) *Control* generates control commands such as accelerating, braking, and steering, and *CAN Bus* passes a set of control commands to the vehicle.

Object detection has significantly improved with the advance of deep learning techniques. Our baseline object detection, which is widely used in several areas including state-of-the-art autonomous driving systems [1, 5], leverages a You Only Look Once (YOLO) [20, 22] approach. Unlike previous object detection algorithms, the whole image (frame) is passed as an input to the Deep Neural Network (DNN) of Apollo-OD just once, see Figure 2. The DNN inference provides the bounding boxes as well as the classified objects. To that end, it first divides the (frame) image into an $N \times N$ grid. Each cell inside the grid is responsible to predict and generate K bounding boxes, where each bounding box describes the rectangle that encloses an object. Apollo-OD, similarly to YOLO, also outputs a confidence score that tells how certain it is that the predicted bounding box actually encloses an object. For each bounding box, the cell also predicts an object class [22]. This works just like a classifier by giving a probability distribution over all the possible classes.

Apollo-OD comprises three stages. After the initialization completes, these stages repeat and work on a per-frame basis:

- *Fetch* (*F*) loads the frame data.
- *Predict* (*P*) evaluates the input data with the DNN. The latest and the most accurate trained DNN is composed of more than 100 layers [21].
- *Display* (*D*) provides the outcome of the processing.

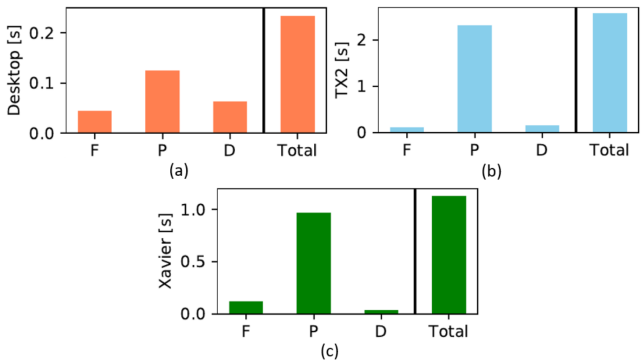

Figure 3: Execution time breakdown of baseline Apollo-OD Stages in our reference platforms. The execution time for each stage is measured for processing three frames.

Table 1: Hardware setup and parameters.

Platform	Desktop System	Jetson TX2	Jetson AGX Xavier
CPU	AMD Ryzen 7 1800x	1) ARM Cortex A-57 Quad-core 2) Denver Dual-core	1) Carmel ARMv8.2 Octa-core
Memory	64 GB DDR4	8 GB DDR3	16 GB DDR4
NVIDIA GPU	GTX 1080 Ti	Pascal, 256 cores	Volta, 512 cores

As we show in the next section, P stage is the most time consuming one, while F and D take shorter to execute, see Figure 3(a), 3(b), and 3(c). Within the Apollo-OD software pipeline, F, P, and D work in parallel on frames f_i, f_{i-1}, and f_{i-2} respectively.

3 INTPRED: ANALYSIS AND PROPOSAL

We use three different platforms with configurations described in Table 1. We use a high-end desktop GPU system with a powerful NVIDIA GPU (*Desktop* setup) and two embedded automotive platforms: the NVIDIA Jetson *TX2* and *AGX Xavier*.

3.1 Performance analysis of Apollo-OD stages

Even sophisticated high-end multicore CPUs cannot satisfy the requirement of processing tens of frames per second in an autonomous driving system [15]. Therefore, such systems highly rely on the use of accelerators, e.g., high-end GPUs for such tasks.

We first profile the execution time of the different stages of the object detection pipeline. Figures 3(a), 3(b), and 3(c) show the average execution time for each stage in the baseline Apollo-OD when processing our video tests on the Desktop and both Jetson platforms respectively. In all three experiments, we observe that P is the stage taking the longest to execute, especially in the case of the Jetson TX2 and Xavier platforms. This occurs because the GPU in the Jetson series are designed with reduced computing capabilities in comparison with high-end Desktop GPUs such as the GTX 1080 Ti. In particular, in the Jetson GPUs, the *Prediction* stage requires more than 85% of the total execution time to evaluate the DNN of Apollo-OD in the GPU. For instance, in the case of the TX2, *Prediction* stage requires 89% of the execution time, and *Display* and *Fetch* stages require only 6% and 5% respectively.

3.2 Introduction to IntPred

IntPred builds on the fact that frames captured in a small timespan are similar. From this observation, IntPred detects objects in the f_i frame, does not perform the DNN-based predict stage for w frames, i.e. in frames f_{i+1}, f_{i+2},... f_{i+w}, and resumes it for frame f_{i+w+1}. In those frames for which P is skipped, IntPred performs the detection using interpolation (I), similarly to solutions used in other domains [14, 26].

Algorithm 1 shows the pseudo-code for the interpolation step. f_i is a predicted image, j is the index of the image for which we want to produce the detections by interpolation and f_{i+w+1} is another image that we have already predicted. We assume that $f_i < f_j < f_{i+w+1}$ which means f_j comes after f_i and before f_{i+w+1}. p_i and p_{i+w+1} represent the array of the detections associated to the frames f_i and f_{i+w+1} respectively.

Algorithm 1 Interpolation of the detection

Require: $f_i < f_j < f_{i+w+1}$
Ensure: Estimate of the detection for the jth frame
1: **procedure** INTERPOLATE($w, f_i, f_j, f_{i+w+1}, p_i, p_{i+w+1}$)
2: $r \leftarrow w + 1$
3: $d \leftarrow j - i$ ▷ d represents how much f_j is far from f_i
4: $p_j \leftarrow \frac{p_i \cdot (r-d) + p_{i+w+1} \cdot d}{r}$ ▷ Do the weighted average over r
5: **return** p_j
6: **end procedure**

3.3 IntPred Software Pipeline

Figure 4(a) shows the pipeline of the baseline Apollo-OD for predicting objects in video frames. First, frame f_1 is fetched and after, the prediction $P1$ stage for frame f_1 starts. In parallel with $P1$, the next frame is fetched ($F2$). As discussed, the P stage takes more time than other stages. Therefore, after it finishes, it is displayed ($D1$) while the prediction stage has another frame for processing and the fetch stage can fetch the new frame ($P2$ and $F3$ respectively). This loop can continue as long as there are new frames to process.

Figure 4(b) shows an example of how IntPred pipeline stages work for $w = 2$ and how it compares to Apollo-OD default pipeline (Figure 4(a)). Without loss of generality, in this example, we have assumed that all stages have constant latency in processing frames. While this does not hold in reality, this just affects the improvements achieved by IntPred but not its functioning. We also assume that the F,D, and I stages take similar time, while P takes longer based on the results presented in previous Section.

Frames f_1, f_4, f_7,... are processed as in Apollo-OD, i.e. with the default F-P-D pipeline. After fetching f_1, $w = 2 + 1$ additional frames are fetched ($F2$, $F3$, $F4$). After the prediction of the first frame ($P1$), we predict the fourth ($P4$). This creates the boundaries to use interpolation (I) instead of prediction (P) for frames 2 and 3, i.e. $I2$ and $I3$. In the first hyperperiod, we can see that $I2$ and $I3$ end their execution after $P2$ and $P3$ in the default Apollo-OD, which makes that frames 2 and 3 end up being processed slower than in Apollo-OD. However, this only happens for the first hyper-period, i.e. first few frames. In the following periods, we can see how all frames (i.e. from f_4 to f_7) finish their execution faster under IntPred than under Apollo-OD.

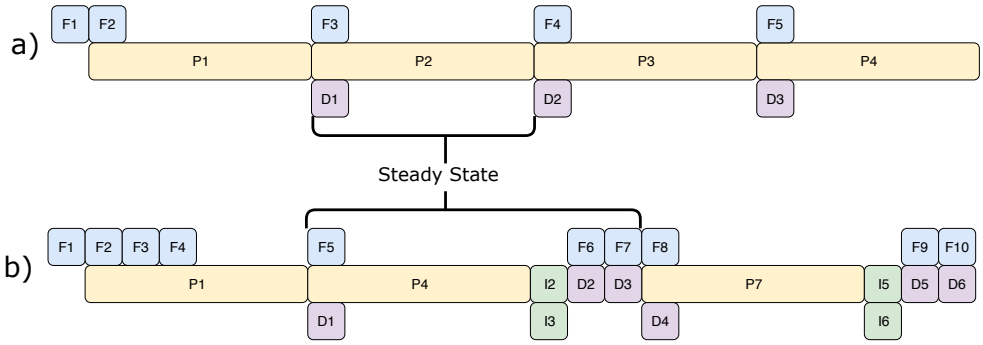

Figure 4: YOLO and IntPred timeflow. YOLO stages are F,P, and D. IntPred uses I instead of P. The number indicates the frame number being processed. The interpolation width, w is 2.

Algorithm 2 Modified Workflow of Apollo-OD

```
1:  procedure EXECUTE_YOLO(w, net)
2:      i = 0                                              ▷ Setup Phase
3:      F(i)
4:      parallel p_i ←P(i) & ∀j ∈ [i + 1, i + (w + 1)] F(j)
5:      repeat                                             ▷ Main Loop
6:          parallel p_{i+w+1} ←P(i + w + 1), D(i, p_i) & F(i + w + 2)
7:          ∀j ∈ [i + 1, i + w] p_j =I(w, i, j, i + w + 1, p_i, p_{i+w+1})
8:          for all j in [i + 1, i + w] do
9:              parallel D(j, p_j) & F(j + w + 1)
10:         end for
11:         i ← i + w + 1
12:     until There are processable frames
13: end procedure
```

Note that cameras are able to provide frames at a very high rate (e.g., 60 FPS). Therefore, due to the massive computation required for processing frames at such a high rate and in real-time, computation stages are the limiting factor. Moreover, as discussed earlier, autonomous cars use multiple cameras to detect the objects surrounding the car (for instance, Tesla uses eight cameras in various types [25]), thus further increasing the need for frame processing computation.

3.4 IntPred Workflow

Algorithm 2 shows how our modified workflow is executed. An example of the workflow with $w = 2$ is shown in Figure 4. The setup phase sets the index of the actual frame at f_0, performs the fetch for the first $w + 3$ frames, and performs the prediction for the first frame.

As far as there are frames to be processed, the main loop iterates, and it operates as follows: while the prediction is running on the GPU, the system displays the previously predicted frame and fetches a new one. Then, the predictions for the frames f_0 and f_{w+1} are available and the interpolation for the frames between f_0 and f_{w+1} takes place. The next step draws the interpolated frames in order and, at the end, sets the frame index to the last predicted frame ($i \leftarrow w + 1$). In the next iteration, the predictions associated to this frame will be reused in the new interpolation step. This allows to perform just one prediction per iteration and w interpolations. This workflow can be implemented using a circular buffer with at least $w + 3$ slots.

3.5 Discussion

Increasing the interpolation window size, w, may cause a small delay, in order of milliseconds, in detecting objects since the fetching process that takes place during the prediction may be longer than the prediction itself. This is the case when the baseline system is able to process the input frames at the same rate as they are produced and in real-time. Therefore, IntPred can cause few milliseconds (ms) of delay in generating the results for any given frame. For instance, to process frames at 30 fps, each frame should be processed in up to 33 ms. As Figure 4 shows, using IntPred can result in a small delay (few milliseconds) in generating the results due to the I stage.

Autonomous cars need to frequently update their status to keep up with the continuously changing real-time traffic conditions. To this end, the AD system should be able to process at a sufficiently high frame rate so that, in case of drastic traffic condition changes between neighboring frames, it reacts quickly. Moreover, such reaction time must be lower than the human reaction time, which requires a frequency of at least one reaction every 100 ms. According to the collision prevention assistant systems built by Mobileye [2], this also aligns with its frame rate. Although the delay produced by IntPred is negligible in comparison with the human's reaction time and autonomous systems requirements, we keep this delay as small as possible. As shown later in Section 4, due to the impact of large interpolation window sizes, w, on the accuracy of prediction and also the possible impact on delay, we only use a small window size.

4 EXPERIMENTAL SETUP AND RESULTS

We use a recorded video collected by AD cars and select several frame sequences, with 10,000 frames in total, covering most of the important scenarios and objects of interest for an autonomous car: crossroads, pedestrians on the side-walk and crossing the road,

Table 2: FPS for the baseline and our technique w/ variable w.

	Desktop		Jetson		Xavier	
Interpolation Width	Mean	Std	Mean	Std	Mean	Std
Default	22.3	0.7	1.5	0.1	2.7	0.1
$w = 1$	29.3	1.5	2.7	0.1	4.5	0.1
$w = 2$	34.5	1.7	3.9	0.1	6.6	0.4
$w = 3$	38.2	2.0	4.8	0.2	7.1	0.4
$w = 4$	40.5	2.2	5.7	0.3	7.8	0.3
$w = 5$	41.8	2.4	6.4	0.4	8.5	0.3
$w = 6$	42.8	2.3	7.0	0.5	9.8	0.4

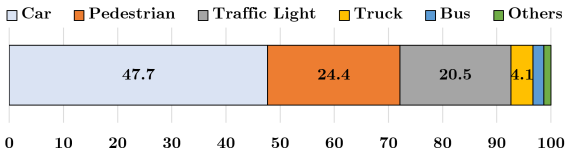

Figure 5: Distribution of each object class among the total detected objects.

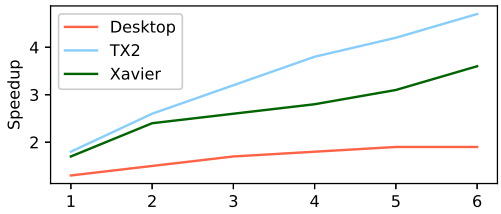

Figure 6: Speedup introduced by the interpolation step. The values of x-axis show different window sizes, w.

tunnels, stops with stop signs, traffic lights, cars in the street and parked at the side.

4.1 Object Classes

According to our analysis from real autonomous cars OD, such as Apollo, and also according to the available annotations in datasets collected by MIT [17], we only focus on five different classes of objects since they are the most important and critical objects of interest for any autonomous car. Figure 5 shows these classes. This figure also shows the percentage of each class of objects detected in the default version of Apollo-OD (2% Bus objects and 1.3% Other objects).

4.2 Performance

Table 2 shows the performance improvements when our interpolation technique is introduced. The *Default* configuration in the *interpolation width* column corresponds to results obtained by running the default version of Apollo-OD, while the numbers are associated to different interpolation widths, i.e. we tested interpolation widths from 1 to 6. We provide the results for both high-end and also Jetson platforms with the configuration described in Table 1. The results for the Desktop platform show a considerable increase in the frame rate from 22.3 FPS up to 42.8 FPS, which results in a speedup of 1.62x.

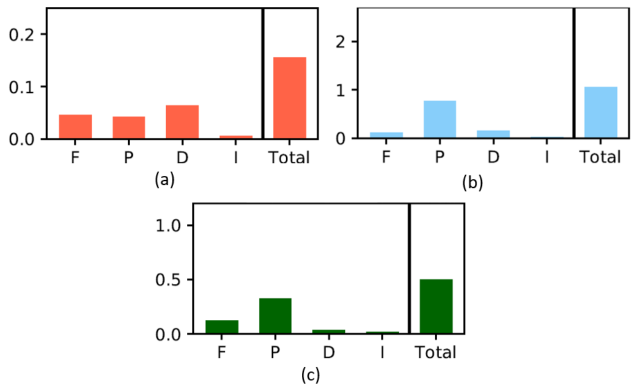

Figure 7: Execution time breakdown of IntPred Stages in our reference platforms. The execution time for each stage is measured for processing three frames.

The frame rate in the *Jetson* configurations is very low, since the *Prediction* stage that evaluates the DNN requires significant computational power. The introduced interpolation allows to reach 7.0 FPS with an interpolation width of 6 in the case of *TX2*, while with the *Xavier* platform can achieve 9.8 FPS. This translates into a speedup of 4.6x and 3.6x respectively for the Jetson platforms as it is shown in Figure 6.

In Figures 7(a), 7(b), and 7(c), we can see the requirements of the interpolation algorithm with respect to the original DNN-based prediction and also the *Display* and *Fetch* stages. As shown, in the default Apollo-OD, the *P* stage is the most time consuming stage of the application as discussed before. On the contrary, by using IntPred, we significantly reduce the time spent in the *P* stage in comparison with Figure 3.

As the results show, IntPred provides significant speedups for platforms such as Jetson where the computation capabilities are much lower than those of high-end and powerful GPUs. The speedup in the *Desktop* setup saturates when the interpolation width becomes large enough to make the display and fetch steps (line 9 of Algorithm 2) take the same amount of time as the prediction step. In the *Desktop* platform, this occurs with a small interpolation width since the setup uses a high-end GPU and so, the relative duration of the prediction stage is reduced with respect to the other steps. The lower performance of the GPU in Jetson platforms leaves more room for optimization for the new workflow, thus allowing to fetch more frames without introducing additional penalties.

4.3 Accuracy Metric

The main issue with increasing the interpolation width is the loss of accuracy. In order to estimate the accuracy of IntPred, for detection in each frame of the baseline, we tried to match it with a detection of the IntPred version by looking at their position within the frame[1]. From this set of pairs, for every frame and for each class of objects we selected the ones that have the maximum difference in probability between the two members of the pair. At the end of this process, for every frame and for every class, the maximum

[1]Pairing detections using the location is the only way for matching elements since it is not possible to uniquely identify detections among different versions.

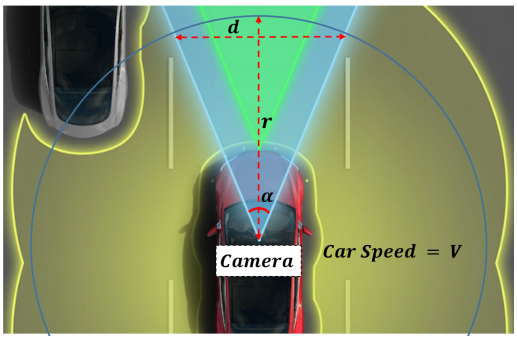

Figure 8: Field of view for the front camera in an AD car with the specified parameters.

difference in probability between Apollo-OD and IntPred is available. We used the dataset described in Section 4 for the analysis of accuracy, which we applied to each individual frame.

We compare the number of objects detected in the baseline Apollo-OD and in IntPred and we take this into account in measuring the prediction accuracy. It may happen that an object is not detected in the baseline whereas it appears in the IntPred due to an effect from other frames. On the contrary, an object may not appear in the IntPred whereas it is detected in the baseline. Note that since the baseline system is not a perfect predictor, the mispredictions in the baseline and in the IntPred compensate part of the accuracy loss in IntPred. However, the baseline system (with no interpolation) must be the reference against which to compare IntPred, since it is the best – yet realistic – object detection scheme.

4.4 Reducing the Accuracy Loss

IntPred builds on two main observations to reduce the accuracy loss:

(1) An object/obstacle cannot appear and cross the front camera in less than a specific period of time.
(2) Autonomous driving systems consider a sequence of few consecutive images to make proper decisions about an object/obstacle.

Regarding observation (1), we use formulation proving that an object/obstacle needs to move with an unrealistic speed in order not to be captured in few consecutive frames by the camera of an AD car. Note that this correlates with the fact that even the best human drivers need more than 200 ms to react properly in different situations.

We need to know whether it is possible that an object or obstacle enters the field of view of the camera but it is not detected in a series of frames. We formulated this problem to show that, if we consider a window of consecutive frames, an object will always appear in at least one of the frames of the window unless it moves faster than a given speed. Note that a failure of the object detection system to detect an already existing object in several consecutive frames (in the baseline scheme without interpolation or with reasonable interpolation) is very rare and, therefore, we do not take this rare case into account in our formulation.

Figure 8 shows the field of view of the front camera in an AD car as well as different parameters including distance, velocity of the car, etc. We draw a circumference centered at front camera's position with radius r around the AD car, which moves at a speed v_y^{car}. We assume that this – r – is the maximum distance at which the camera can detect an object. We also assume that the front camera has an angle of vision of α.

If an object is positioned at distance r from the car, entering the camera's field of view from either left or right, in a trajectory perpendicular to the car, the object needs to have at least the speed of v_x^{obj} in order to cross the field of view of the camera in w frames in time t.

If the car is static, the distance to cross would be $d = 2r\sin(\frac{\alpha}{2})$. However, since the car is moving perpendicular to the object, this distance is shortened over time. The distance of the object to the moving car is shortened as $d_f - d_o = -2v_y^{car}t\tan(\frac{\alpha}{2})$. Now, we can add this term to show the speed of the object in terms of the distance when the vehicle is moving, as Equation (1) shows.

$$v_x^{obj} = 2\frac{r\sin(\frac{\alpha}{2}) - v_{obj}^{car}t\tan(\frac{\alpha}{2})}{t} \quad (1)$$

Example. In order to illustrate this discussion, let us assume that we have a frame size of $w = 3$ and the camera captures at 25 FPS, i.e. capturing a frame every 40 ms. We assume that the field of view of the camera (α) is 60 degrees, $v_{obj}^{car} = 30m/s$ (which is equal to 108 Km/h), $t = 0.12$ and $r = 100m$.

$$v_x^{obj} = 2\frac{100\sin(60) - 30\cdot 0.12\tan(60)}{0.12} = 1,339.45\text{m/s} \quad (2)$$

As shown in this example with realistic parameters, an object needs to move at an unfeasible speed in order not to be captured in any of the frames of the window. From another point of view, the example clearly illustrates why objects have gradual (and tiny) movements with respect to high frame rates of the camera.

Regarding observation (2), as mentioned earlier, AD systems consider a window of consecutive frames for making more precise decisions. For instance, Apollo perception module uses a window of frames for tracking objects and make decision on perceived objects. Wider sizes of the window help to increase the reliability of the perception.

We use this feature of the AD systems to reduce the accuracy loss in the interpolation stage of the IntPred. As an example, Figure 9 shows four consecutive frames in which $w = 2$ and also detected objects in each frame. As the Figure shows, we use P_1 and P_2 as reference frames to do an interpolation for frames I_1 and I_2. According to observation (1), objects that are detected in both P_1 and P_2 frames should also be detected in frames I_1 and I_2. In case there are some objects that are in P_1 and not in P_2 (or vice versa), then IntPred considers the closest processed frame as the reference for the interpolated one, so frame P_1 for I_1 and P_2 for I_2 in the example. As an example, in Figure 9, O_3 in frames P_1 and I_1 and O_4 in frames P_2 and I_2 shows this case. In case of having equidistant processed frames (i.e. central interpolated frame when w is odd), then we select one of the processed frames (the newest one).

Our experiments show that this approach can recover most of the accuracy loss caused by IntPred without this feature. Note that the most accurate object detectors are still not perfect. Therefore, part of accuracy loss is related to the inaccuracy in the baseline predictor. For instance, if O_3 in P_1 is an incorrect detection, this is

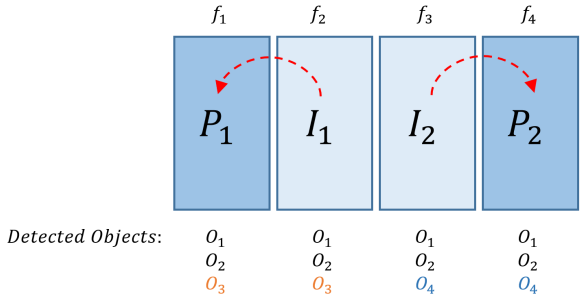

Figure 9: Four consecutive frames in the IntPred with $w = 2$ and the detected objects in each frame.

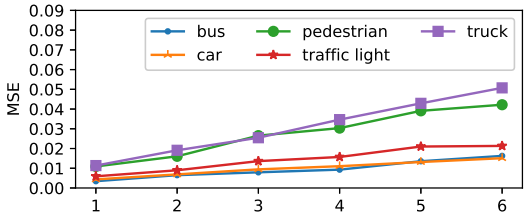

Figure 10: MSE by changing the interpolation width.

propagated to I_1 as well. On the contrary, if an object is not detected in P_1, it will also be omitted in I_1. As shown next in our experiments, however, these corner cases have minimal impact on the accuracy of the IntPred.

4.5 Accuracy Results

Figure 10 shows the Mean Squared Error (MSE) estimates between the probability of the used dataset with different interpolation widths. As shown, all the MSEs are less than 0.05, which means that the error is contained in general. The highest value is obtained with the *truck* class when the interpolation width becomes large since the number of samples is smaller in comparison with the others. This makes the MSE related to such a class less closer to the reference value with a given w. Note that, more than 92% of the detected objects belong to *Pedestrian*, *Car* and *Traffic light* classes, whose joint MSE for $w = 4$ is below 0.015.

Apollo-OD considers only the detected objects with a probability higher than a certain threshold. Therefore, only few objects with a high probability will always remain in every frame. Our studies show that the average probability of detected objects for *Car*, *Pedestrian*, *Truck*, *Traffic Light*, and *Bus* are 86.4%, 81.9%, 77.3, 71.9%, and 95.8% respectively. Although the probabilities using IntPred may slightly vary due to interpolation, however, such a small variability in the probabilities does not affect the outcome of the object detection in general.

Figure 11 shows the histogram of maximum prediction probability differences for the objects and with different sizes of interpolation width, w. Each category shows the number of objects within a specific error range. As shown, for most of the objects and the predictions, the differences are relatively small. However, for a small percentage of the detections, there are higher differences. For instance, for *Bus* category, most of detections have a difference in

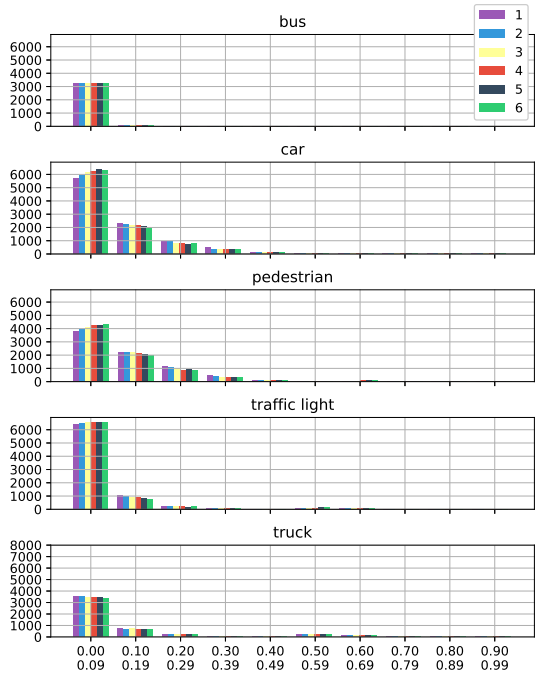

Figure 11: Histogram plot for the prediction error of the IntPred in comparison with the Apollo-OD.

the probabilities between 0.0 and 0.1. However, a small percentage of objects has higher differences in the probability of detection, which are shown in the corresponding range in each plot. Note that, according to our analysis, most of the detections have a very high confidence score (probability) that is higher than the other scores by far. We observed that for more than 98% of the cases, the differences in the scores does not change the order of the top identified objects in each frame. This results in the same output for the AD system in comparison with the Apollo-OD. For instance, for pedestrian object class and different probability ranges we observe that for Apollo-OD and IntPred, more than 95% of the frames have the same top objects. This means that regardless of the small difference in the confidence score of the detected objects, IntPred provides the same object as a result in comparison with the baseline Apollo-AD.

5 RELATED WORK

Several works in the context of AD try to improve object detection speed using various methodologies. [23] proposes using 2.8x fewer parameters of YOLO DNN, which affects the accuracy of YOLO. Authors also propose to use a motion-adaptive inference method to reduce the frequency of the inference. However, this works only when the technique does not detect a significant difference in the classes of objects. Other works, [27–29], propose how to compute higher quality features faster by using temporal information, which are specific to different object detection architectures.

Software optimizations such as using highly optimized libraries (e.g., cuDNN, OpenBLAS, etc.), and model compression, provide significant performance and energy improvements. Hardware accelerators provide orders of magnitude in performance improvements and energy savings. However, long design process and lack of

flexibility are their disadvantages, especially for rapidly-changing applications such as object detection.

Many works vary some critical parameters such as the number of DNN parameters, precision of the data, and image resolution, which mainly sacrifice accuracy to achieve higher detection speed. A summary of such works can be found in [12].

Beyond AD, frame interpolation has already been studied for other problems, including object detection [14, 26]. Those works already support the concept of frame interpolation, which we apply to a different domain – automotive – with its own constraints and platform characteristics.

Our work is different in several aspects. First, our baseline system is implemented using highly optimized libraries. In addition, we propose a technique that is very efficient for object detection in autonomous driving systems, in which the frames change smoothly, similarly to [14, 26]. Furthermore, we do not sacrifice the accuracy of the system by reducing the network parameters or varying other parameters such as the frame resolution. Finally, our approach is orthogonal to the aforementioned ones and can be combined easily for increased computation and energy savings.

6 CONCLUSIONS

We proposed IntPred, a scalable and fast object detection scheme for autonomous driving systems. Unlike the non-critical domains where performance and energy efficiency of object detection systems can be improved at the cost of reducing accuracy, in critical domains such as autonomous driving vehicles the accuracy of the system is the most important factor. We observed that in captured video from an autonomous vehicle camera, consecutive frames do not tend to change significantly. Based on this observation, IntPred applies a light-weight interpolation of objects in certain frames, which results in a significant reduction in the DNN computation phase of object detection. IntPred provides very close accuracy to the baseline object detection while providing 4.6x speedup in an NVIDIA Jetson TX2 platform by saving more than 70% of the computations.

ACKNOWLEDGEMENTS

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness (MINECO) under grant TIN2015-65316-P, the SuPerCom European Research Council (ERC) project under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 772773), and the HiPEAC Network of Excellence. MINECO partially supported Jaume Abella under Ramon y Cajal postdoctoral fellowship (RYC-2013-14717) and Leonidas Kosmidis under Juan de la Cierva-Formación postdoctoral fellowship (FJCI-2017-34095).

REFERENCES

[1] 2016. Autoware. An open autonomous driving platform. https://github.com/CPFL/Autoware/.
[2] 2017. Mobileye, Mobileye C2-270 Essentials.
[3] 2017. NVIDIA GeForce GTX 1080 Ti. https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080-ti/.
[4] 2017. NVIDIA Jetson TX2. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/.
[5] 2018. Apollo, an open autonomous driving platform. http://apollo.auto/.
[6] 2018. Apollo Hardware Configurations. http://apollo.auto/platform/hardware.html.
[7] 2018. NVIDIA drive platforms. https://www.nvidia.com/en-us/self-driving-cars/drive-platform/.
[8] Sergi Alcaide, Leonidas Kosmidis, Hamid Tabani, Carles Hernandez, Jaume Abella, and Francisco J Cazorla. 2018. Safety-Related Challenges and Opportunities for GPUs in the Automotive Domain. *IEEE Micro* 38, 6 (2018), 46–55.
[9] ARM. 2015. ARM Expects Vehicle Compute Performance to Increase 100x in Next Decade. https://www.arm.com/about/newsroom/arm-expects-vehicle-compute-performance-to-increase-100x-in-next-decade.php.
[10] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. 2015. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*. 2722–2730.
[11] Mike Demler. June 2017. Xavier Simplifies Self-Driving Cars.. In *Microprocessors Report, The Linly Group*.
[12] J. Huang et al. 2017. Speed/accuracy trade-offs for modern convolutional object detectors. In *CVPR*.
[13] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, et al. 2015. An empirical evaluation of deep learning on highway driving. *arXiv preprint arXiv:1504.01716* (2015).
[14] R. Jain et al. 2011. Interpolation Based Tracking for Fast Object Detection in Videos. In *CVPR*.
[15] Shih-Chieh Lin et al. 2018. The Architectural Implications of Autonomous Driving: Constraints and Acceleration. In *ASPLOS*.
[16] Fabio Mazzocchetti, Pedro Benedicte, Hamid Tabani, Leonidas Kosmidis, Jaume Abella, and Francisco J Cazorla. 2019. Performance Analysis and Optimization of Automotive GPUs. In *Proceedings of the 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD) 2019*. IEEE.
[17] Massachusetts Institute of Technology-MIT. [n. d.]. Annotated Driving Dataset. https://github.com/udacity/self-driving-car/tree/master/annotations.
[18] Roger Pujol, Hamid Tabani, Leonidas Kosmidis, Enrico Mezzetti, Jaume Abella, and Francisco J Cazorla. 2019. Generating and Exploiting Deep Learning Variants to Increase Heterogeneous Resource Utilization in the NVIDIA Xavier. In *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
[19] A. Lewis R. Haslehurst. 2019. Mapping the Road to Autonomous Vehicles. In *Insights Work*, Vol. XIX.
[20] Joseph Redmon and Ali Farhadi. 2017. YOLO9000: better, faster, stronger. *arXiv preprint* (2017).
[21] Joseph Redmon and Ali Farhadi. 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767* (2018).
[22] J. Redmon et al. 2016. You only look once: Unified, real-time object detection. In *CVPR*.
[23] M. Shafiee et al. 2017. Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video. *arXiv preprint arXiv:1709.05943* (2017).
[24] Hamid Tabani, Leonidas Kosmidis, Jaume Abella, Francisco J Cazorla, and Guillem Bernat. 2019. Assessing the Adherence of an Industrial Autonomous Driving Framework to ISO 26262 Software Guidelines. In *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 9.
[25] TESLA. [n. d.]. Full Self-Driving Hardware on All Cars. https://www.tesla.com/autopilot.
[26] T. Ujiie et al. 2018. Interpolation-Based Object Detection Using Motion Vectors for Embedded Real-time Tracking Systems. In *CVPR Workshops*.
[27] X. Zhu et al. 2017. Deep feature flow for video recognition. In *CVPR*.
[28] X. Zhu et al. 2017. Flow-guided feature aggregation for video object detection. In *CVPR*.
[29] X. Zhu et al. 2018. Towards High Performance Video Object Detection. In *CVPR*.