Light-weight Object Detection and Decision Making via Approximate Computing in Resource-constrained Mobile Robots

Parul Pandey[1], Qifan He[2], Dario Pompili[1], and Roberto Tron[2]

***Abstract*— Most of the current solutions for autonomous flights in indoor environments rely on purely geometric maps (e.g., point clouds). There has been, however, a growing interest in supplementing such maps with semantic information (e.g., object detections) using computer vision algorithms. Unfortunately, there is a disconnect between the relatively heavy computational requirements of these computer vision solutions, and the limited computation capacity available on mobile autonomous platforms. In this paper, we propose to bridge this gap with a novel Markov Decision Process framework that adapts the parameters of the vision algorithms to the incoming video data rather than fixing them a priori. As a concrete example, we test our framework on a object detection and tracking task, showing significant benefits in terms of energy consumption without considerable loss in accuracy, using a combination of publicly available and novel datasets.**

I. Introduction

Autonomous navigation in indoor environments (e.g., with robotic drones) can be achieved through the use of mapping, planning, and control solutions that rely purely on *geometric* information (such as occupancy grids or point clouds). There exist an opportunity, however, to improve these techniques for robots that need to work in close proximity to humans, e.g., in controlled environments such as warehouses or in search and rescue missions. There has been a growing interest in supplementing geometric maps with *semantic* information, by using vision information to reach some understanding of the environment [1]. As a motivating application, we posit that the autonomous robots could use the same existing infrastructure developed for humans, such as traffic signs, to navigate. In other words, signs that have intuitive meaning for humans can be used to influence the behavior of robots; for instance, if a robot detects a "Stop" sign, it might pause its motion until some environment-specific condition is met.

To realize these functions, it is necessary to run object detection algorithms on the incoming data (video stream) in *near real-time*. However, running computationally-intensive vision algorithms using on-board processors can cause delay in detection and significant battery consumption leading, to reduction in the mission time. In robotics, offloading operations to clouds has been proposed as a way to handle the constraints on the robots, targeting collaborative applications such as localization [2], [3] and object recognition [4], [5]. However, such approach faces challenges such as a variable network connectivity or a high latency, leading to performance bottlenecks, especially for real-time/interactive applications that require low response times.

[1]Parul Pandey and Dario Pompili are with the Department of Electrical and Computer Engineering, Rutgers University–New Brunswick, NJ {parul_pandey, pompili}@cac.rutgers.edu

[2]Qifan He and Roberto Tron are with the Department of Mechanical Engineering, Boston University, MA {heqf, tron}@bu.edu

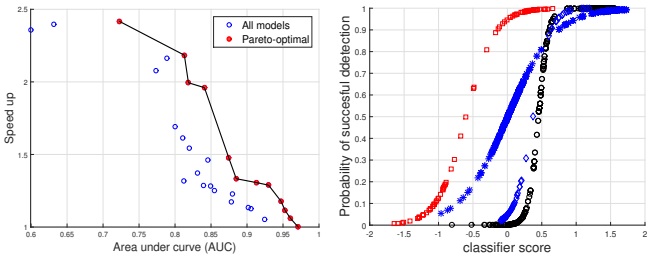
Proposed Approach: In recent years there has been tremendous improvement in the performance of detection algorithms; however, these algorithms have been mostly designed for accuracy, and are too slow for resource-constrained autonomous systems (since they typically rely on deep neural architectures or model ensembles). We argue that the selection of the computer vision algorithm, along with its parameters, should depend on the input data, resources available with the device (battery capacity, and CPU frequency), and the accuracy of the results that is acceptable to the user or application. To this end, we exploit the paradigm of *approximate computing* that reduces the amount of computation that an application is expected to perform, resulting in a reduction of the application execution time, subject to a potential but acceptable loss in accuracy.

Specifically, we propose a decision framework that selects the parameters of computationally intensive algorithm that are "good-enough", i.e., parameters that give acceptable accuracy in each frame of the video with significant savings in time and energy. Our framework exploits the temporal correlation between the continuous stream of frames obtained from the camera sensors, learns the good-enough parameters from the first few frames, and applies these parameters to the subsequent frames. We cast the problem of selecting the algorithm and input parameters for a video as a Markov Decision Process (MDP). MDPs provides a mathematical framework for modeling decision making in a stochastic environment. We design a reward function for the MDP that achieves acceptable accuracy for the application with minimum cost (in terms of time). While in this paper we focus on a specific case study, the selection of good-enough parameters via MDP is a generic technique that may be applied to a wide range of computationally-intensive algorithms. Our proposed framework works well in variety of challenging conditions, such as those that exists in object detection application (e.g., background clutter or poor illumination).

Previous Work: The problem of object detection has been considered exhaustively in the computer-vision literature. However, our focus is on reducing the computational requirement of state-of-the art detector to better fit real-time applications on resource-constrained platforms. Likewise, MDPs have also been used in the computer vision community for the problem of object detection [6], [7], multi-object tracking [8], human path predictions [9], and videogame play [10]. However, they have not been considered with the goal of achieving good-enough performance for video data on resource-constrained devices.

Paper Contributions: We make these contributions:

- We use object detection and tracking as a case study to motivate that traditional "one-size-fits-all" approaches are energy and time inefficient for real-time applications, while we show via simulations and experiments that our solution shows improved performance.



(a) (b)

Fig. 1: (a) Representing the tradeoff between accuracy in terms of area under the curve (AUC) and speed up in execution time per frame for different values of parameter tuple of the object detection pipeline, namely, number of detection proposals and cell size in object detection pipeline; (b) Illustration of variation of probability of successful detection with classifier score for various road sign in KUL traffic sign dataset [11]. The SVM scores have been converted to probabilities using Platt Scaling [12] and we determine the values of c_{opt}, c_{ge}, and c_{ua} for the reward function.

- We identify, via offline experiments, the algorithm and parameters of an object detection application that can be approximated, showing 42% increases in speed while maintaining 80% accuracy.
- We validate the proposed solution on publicly available datasets, and on a new dataset of videos collected with a Bebop drone; we achieve up to $30\,\text{fps}$ for 480×270 video frames with an accuracy drop of less than 2% with respect to a fixed choice of object detection parameters.

Paper Organization: In §II we introduce the entities of our decision framework, cast the problem of selecting the parameter-algorithm combination as an MDP, and present its application to a popular object detection algorithm. In §III, we provide details of our experimental setup and study the performance of our solution under various conditions. Finally, in §IV, we conclude our work.

II. Proposed Work

In this section we give the details of our proposed MDP-based decision framework for selecting good-enough parameters for an object detection and tracking. In MDPs, an agent takes *actions* based on its *state*, and receives a corresponding *reward*. The goal of the MDP is to maximize the sum of rewards. In our case, the rewards are designed such that the system achieves acceptable accuracy with minimum cost (in terms of execution time). To this end, a *policy* is learned for an MDP, which gives the optimal action that should be taken from each state of the MDP to maximize its sum of rewards achieved from that state. In our scenario the agent is represented by the vision-based navigation system in the drone, actions corresponds to choices of the parameter of the object detection workflow that will be used on the received frame, the current state is represented by whether the object is likely to have been successfully detected in the current frame, and the reward is given by a combination of execution time and confidence levels returned by the object detector and tracker. We first detail our decision framework, and then show its application to a popular object detection algorithm.

Decision Framework: As the video is captured by the drone's camera, the frames are given to the MDP, which estimates the good-enough parameters of an object detection algorithm from the first few frames, and then reuses them in the subsequent frames. The system continues to use the good-enough parameters as long as the classifier score is above a threshold, otherwise the MDP is triggered again to re-estimate the good-enough parameters. If, for a given frame, the MDP cannot find parameters that lead to a sufficient score after a set number of attempts, we consider this as a sign of an unlikely detection, and drop the frame. Our framework is composed of the following entities:

States: Each state $s \in \mathcal{S}$ of the MDP is represented by a tuple given as $\mathtt{s} \doteq \{\mathtt{c}, \mathtt{params}\}$, where $\mathtt{c}$ is the classifier score and $\mathtt{params}$ includes all tunable parameters of an object detection algorithm. We first identify all combinations of tunable parameters in $\mathtt{params}$, thus generating different instances. Next, only those instances of $\mathtt{params}$ that are on the Pareto-optimal front with respect to the accuracy/execution time tradeoff are considered in $\mathcal{S}$. See Fig. 1(a) for an example.

Actions: The action space of the MDP consists of choosing good-enough parameters ($\mathtt{params}$) for object detection, and applying them to the incoming frame.

Reward Function: The reward is the way of communicating to the MDP the goal we want to achieve, that is, the agent should obtain a high positive reward when it achieves the desired goal, and a lower reward otherwise. Our goal is to achieve acceptable accuracy with minimum execution time. We use the classifier score as a proxy for detection accuracy; while the latter, naturally, cannot be computed at runtime due to the absence of ground truth annotations, the classifier score correlates well with the quality of the prediction (e.g., see Fig. 1(b)), with higher scores indicating better predictions.

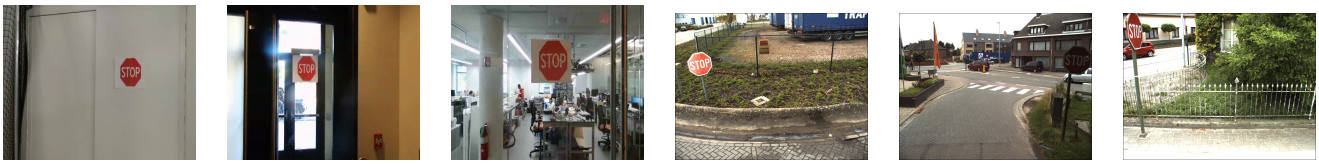
In order to define our reward function, we first delineate three categories for the classifier score c that we obtain after running the object detection algorithm: optimal score (c_{opt}), good-enough score (c_{ge}), and unacceptable score (c_{ua}) with $c_{opt} > c_{ge} > c_{ua}$, and with associated weights w_{opt}, w_{ge}, w_{ua}, respectively,. The category of weights are obtained from offline training of an algorithm, as explained later for a specific object detection algorithm.

When an action is executed (i.e., the object detection algorithm is run with parameters corresponding to a state s'), the MDP receives a classifier score, and thus a weight $w \in \{w_{opt}, w_{ge}, w_{ua}\}$. Our proposed reward function is:

$$R(s', w) = \frac{w}{r(s')^{\alpha}}, \tag{1}$$

where $r(s') = \frac{T_{s'}}{T_{min}}$, gives the factor of increase in execution time $T_{s'}$ when choosing the tuple for state s' with respect to the parameter tuple which takes minimum time T_{min}, and α is a tuning parameter. In order to encourage the MDP to select parameters with acceptable (good-enough or better) accuracy and minimum execution cost we let $w_{ge} \gg w_{opt}$.

Application to Object Detection: In this work we use object detection and tracking as a representative application. We employ two well known computer-vision algorithms, namely, Histogram of Gradients (HoG) [13] and Edge-boxes [14], although our framework could be applied to any set of algorithms that offer tuning parameters. HoG features represent the shape of the object to be detected by computing a dense grid of local gradients (edge directions) in an image. The input image is divided into cells and an orientation based histogram of gradients with bins of a given size is computed for each cell using gradient orientation at each of its pixel. The cell histograms are normalized with respect to neighboring cells in a block of a given size. In this specific case, the parameter tuple is given by $\mathtt{params} \doteq \{\mathtt{BB}, \mathtt{bsize}, \mathtt{csize}, \mathtt{nhist}\}$, that is: number of bounding boxes or



(a) (b) (c) (d) (e) (f)

Fig. 2: Motivation figures to illustrate that parameters of the object detection algorithms can be adapted to input data to achieve savings in time and energy. Figures (a,b,c) show variation in the amount of background clutter; (d and e) show variation in illumination, and (f) shows variation in camera viewpoint. These variation lead to differences in the minimum number of bounding boxes parameter (detection proposals) required for object detection. Lower detection proposals translate to lower execution time.

detection proposals, block size, cell size, and number of bins in the histogram, where detection proposals is a parameter of Edgeboxes, and the others belong to HoG.

Fig. 2 visually shows an example to make the case that the parameters of the object detection algorithm should not be fixed, but should be adapted to the scene. For instance, in Fig. 2(a,b,c) the minimum number of bounding box required per image varies as the background clutter (quantified using [15]) varies from 1 for image 2(a) to 100 for image 2(b). Similarly, due to variation in illumination, image 2(d) requires less detection proposals than 2(e) and for image 2(f) the detection of the road sign (on the bottom-right corner) is not possible because of the camera's viewpoint. Fig 1(a) shows the tradeoff between accuracy in terms of area under the curve (AUC) and speed up in execution time. Each marker in the figure is one instance of `params`, with red markers highlighting the Pareto-optimal instances. We see that we can achieve a reduction in execution time by 50% for a 10% loss in accuracy.

Thresholds: We use a Support Vector Machine (SVM) classifier [16] for which the score c is calculated as $c = w^T x + b$, where w is a weight vector, b is the bias term, and x is the HoG feature extracted from a detection proposal. Features from images of both positive (containing road sign) and negative (with no road sign) training sets are used to train the SVM classifier (i.e., find the optimal values for w and b). Cross-validation is used to determine the threshold score T_c for each road sign over which the probability of correct detection is high. In Fig. 1(b) we give an example of how categories for a classifier score are selected. Here, the SVM scores have been converted to probabilities using Platt Scaling [12]. Using this graph, we set the threshold classifier scores for the three categories as $c_{opt} = 0.9$, $c_{ge} = 0$, and $c_{ua} = -0.5$. We set the value of classifier threshold T_c to be equal to c_{ge}. The parameter α is set in the interval $[0, 1]$.

Run-time Decision Framework: When a new frame enters the navigation system, the MDP framework identifies the good-enough parameters of the object detection algorithm that give classifier score above the threshold T_c with minimum execution time. If after running the object detection application on the frame with good-enough parameters we get score greater than T_c, we initialize a Kanade-Lucas-Tomasi (KLT) object tracker [17] with the object location in that frame. The good-enough parameters given by the MDP are used in the subsequent frames with the combination of the results from the KLT tracker on those frames.

Object Tracker: The KLT tracker produces a bounding box B_t for the current frame propagated from the previous frame, with the results given by the detection algorithm, which gives another bounding box B_d from the current frame. We calculate the overlap between the bounding boxes from the good-enough object detector B_d and the KLT tracker B_t using the *Intersection over Union* (IoU) metric:

$$\theta(B_d, B_t) = \frac{B_d \cap B_t}{B_d \cup B_t}. \quad (2)$$

The MDP is re-triggered when either the classifier score is below the threshold T_c, or when the overlap of the bounding boxes $\theta(B_d, B_t)$ is lower than a pre-defined threshold (θ_{th}).

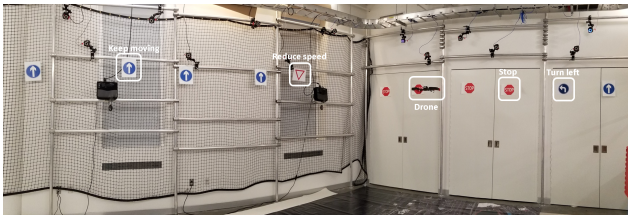
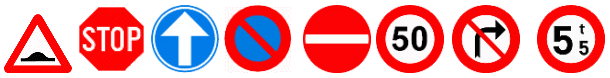


Fig. 3: One of the areas used in the Robotics Lab, Boston University for collecting our dataset. The arena simulates a warehouse where human-interpretable signs (road signs) were used as cues to help the drone navigate the warehouse. Each road sign is associated with a specific control command.



A14 B5 D1a E1 C1 C43 C31R C21

Fig. 4: Traffic signs from the KUL dataset [11] (with the codes assigned to them in the dataset) that we have considered for performance evaluation of our proposed solution.

III. Performance Evaluation

This section focuses on how our proposed MDP-based decision framework can reduce the time required for running object detection algorithms on resource-constrained robots. We evaluate our solution on publicly available datasets and data collected by our team via a drone platform.

Experimental Setup: As mentioned in the introduction, we envision aiding the navigation of robots in a warehouse by using the same visual cues as humans to execute an associated control task (e.g., reducing speed, changing direction). Currently, there is no public dataset for this specific application; instead, as a proof of concept, we use the KUL dataset, a traffic road sign dataset [11]. Videos taken from a moving camera containing traffic signs while traveling along a road is very similar to those collected by a drone in a warehouse where road signs can be used to guide its motion.

In addition, to test the performance of our system in more realistic conditions, we created the *BU-RU dataset*, based on a setup similar to a warehouse in the Robotics Lab at Boston University; see Fig. 3 for an example. The dataset contains videos of a drone (Parrot Bebop) that simulate a warehouse where human-interpretable signs (road signs) were used as cues to help the drone navigate. Each road sign is associated with a specific control command. We collect videos in three

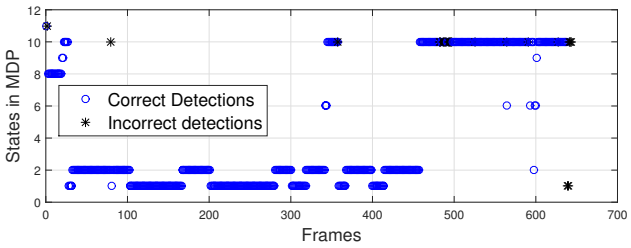
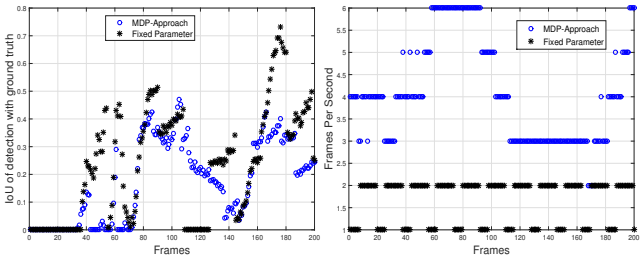


Fig. 5: Illustration of the MDP state space for the object detection algorithm that the navigation system accesses for an example video sequence from the BU-RU dataset.



(a) (b)

Fig. 6: Comparison of performance for an example video sequence in terms of (a) IoU of ground truth with detected locations; (b) Frames per second achieved for an object detection algorithm by parameter selection versus fixed parameter approach.

different locations in the lab, namely, an hallway, a dedicated flying arena, and an office area. In each case, the drone is tele-operated with a laptop through a WiFi connection. The resulting dataset contains scenarios with different conditions in terms of clutter and illumination (see Figs. 2)

Data Preprocessing: We give now a few more details about the two datasets used in this work.

KUL Dataset: In Fig. 4 we show the different road signs used in our evaluation. This set of signs was selected to span sufficient shape variability while appearing often enough in the dataset (i.e., in a sufficiently high number of video frames) so to have statistically relevant results. We extract the portion of the video frames from the dataset which continuously include the selected signs. Typically only one road sign per video frame is present, with few exceptions. These extracted videos are, in average, 40 frames long. Since the original dataset does not provide the ground truth location of the objects in all the video frames, we manually annotated this information in our smaller videos. Unfortunately, given the limits of the instances in which a sign appears continuously in the KUL videos, this dataset is not sufficient to fully test our MDP-based approach. Thus, we have collected a new dataset with longer sequences.

BU-RU Dataset: The video streams for this dataset were shot by a 14 mega-pixel 180° fisheye camera on a Parrot Bebop drone. The raw video stream is cropped and rectified to a plain video with about 80° (horizontal) by 50° (vertical) field of view. The quality of the video stream is limited to 480×270 px at 30 fps. We collected 10 videos each for different values of background clutter (low, medium, high). Each video is about 10 s long (i.e., around 300 frames). We also collected 6 videos in poorly illuminated locations, where each video is also 10 s long. We also collected 200 negative training images with no road signs in them. Our framework is run on a laptop with an Intel i7 CPU 3.4 GHz processor.

MDP States for an Example Video Sequence: We now present an example video sequence to show the advantage of selecting different parameters for road sign detection for HoG object detection algorithm. We choose a sequence from the BU-RU dataset. When the first frame arrives, the MDP decision framework is triggered and good-enough parameters {BB, bsize, csize, nhist} are selected by the MDP. If after executing the object detection algorithm with these parameters the classifier score is greater than the threshold T_c, then the navigation system uses the same set of parameters for the next frame otherwise the MDP is triggered again. In Fig. 5 we can see that due to the temporal correlation between frames, the same parameter values (i.e., state) are reused for multiple video frames.

Note that, while the drone is processing a particular frame, all the other incoming frames are dropped until the drone finishes its processing. Our decision framework focuses on reducing the time taken to process each frame so that the number of frames processed is closer to the incoming frame rate. We now compare the performance of our MDP-based framework against a fixed choice of parameters; for the latter, we use the tuple {1000,2,8,9} as in [18]. The deviation of the detected road sign from the ground truth is shown in Fig. 6(a). We see that our framework achieves up to 6 fps, while the fixed parameter approach tops out at 2 fps, as shown in Fig. 6(b). This is because our framework adapts the parameters using the first few frames, lowering the execution time for the other frames.

Achieving Real-time Performance: Even with the incorporation of our approach, the processing rate achievable through pure object detection is far below the frame rate of the drone platform (30 fps). To improve our real-time performance and also improve the IoU between detected object and ground truth, we can choose to not trigger the MDP for every new frame. After the MDP finishes adapting from the first few frames, we can check the classifier score achieved on the successive frame: if this score is greater than the predefined threshold T_c, the KLT tracker is initialized with the detected object. Here, instead of discarding the remaining frames during processing, as we did earlier, we run the KLT tracker on these frames in parallel.

We use as template of the road sign an average of several images obtained from the training data. We continue to track until the normalized distance between the histograms of the template and detected region by the tracker is greater than a predefined threshold T_m (equal to 0.5 in our experiments). In Fig. 7 we see that this leads to a significant improvement, with performance in the 10 to 30 fps range; this is due to the fact that the KLT tracker is computationally much cheaper than the object detector alone. Fig. 7(b) also shows that the MDP (shown by black dots) is triggered when the normalized pairwise-distance between the histograms of the template and detected region by the tracker is below T_m.

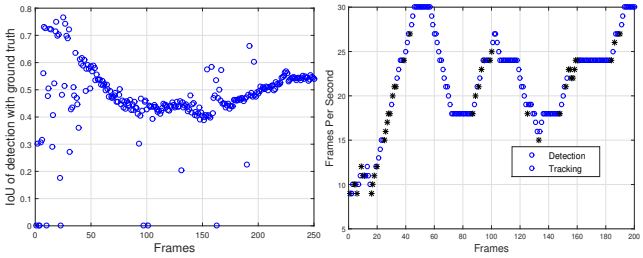
In Fig. 7(a) we see that this approach leads to an improvement in performance and we get up to 30 fps for some frames and the lower bounds on FPS achieved for this sequence is 10 fps with IoU for majority of frames. Also, Fig. 7(b) shows that the MDP (shown by black dots) is triggered when the pairwise distance between the histograms of the template and detected region by the tracker is below T_m. KLT tracker is computationally cheaper and gives us higher FPS performance than using only an object detector. However, object trackers accumulate error during runtime (drift) and typically fail if the object disappears from the camera view. Hence, a combination of detector and tracking

Thresholds	Value
Re-trigger MDP	100 frames
Re-initilize KLT	10 frames
Score thresh T_c (KUL)	0
Score thresh T_c (BU-RU)	-1
Access MDP per image	5
KLT thresh θ_{th}	0.5
Template matching T_m	0.5
Max. object proposals	2000
IoU for evaluation	0.5

TABLE I: Various thresholds considered in the performance evaluation of our solution.

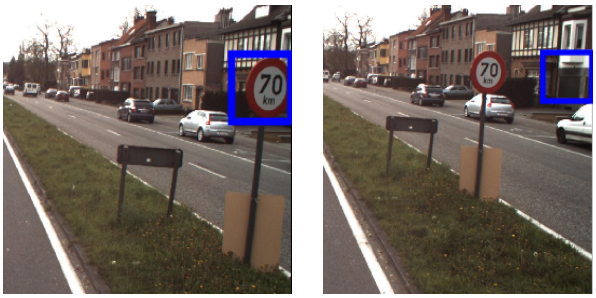
Codes	Average		Good Enough		Speed-up [%]	Accuracy Loss [%]
	P	E [s]	P	E [s]		
A14	0.85	1.85 ± 0.3	0.71	1.59 ± 0.3	14.05	16.40
B5	0.57	2.21 ± 0.11	0.46	2.6 ± 0.51	-	
D1a	1	2.1 ± 0.13	0.78	1.10 ± 0.02	40.1	22.0
E1	0.99	2.1 ± 0.28	0.95	2.14 ± 0.06	45.7	4.0
C1	0.75	2.06 ± 0.16	0.61	1.59 ± 0.1	22.8	6.7
C43	0.82	2.09 ± 0.05	0.60	1.95 ± 0.01	6.7	26.8
C31R	0.85	2.04 ± 0.16	0.80	1.89 ± 0.7	7.4	5.0
C21	0.50	1.88 ± 0.1	0.50	1.56 ± 0.1	17.0	0

TABLE II: **KUL dataset:** Comparison of precision and execution time of road signs (codes) in KUL dataset for fixed parameters [18] and good-enough parameters given by MDP approach for object detection algorithm.



(a) (b)

Fig. 7: Real-time performance of up to 30 fps for some frames and the lower bound of 10 fps for frames in a sequence when MDP based parameter selection is implemented along with tracker. The IoU is greater than 0.4 for majority of frames.



(a) (b)

Fig. 8: Video frames from one of the video sequences in the KUL dataset. The tracker was initialized using the location from given by the detector in (a) i frame. The tracker has drifted away from the object location in the (b) $i+5$ frame and should be re-initialized via location generated by object detection algorithm.

helps in achieving high performance with low execution time. Our solution brings further benefits in reducing time by choosing parameters of the object detection algorithm adaptively. Figure 8 gives an example of how we cannot solely rely on the tracker and so we use a combination of detector and tracker.

Although the KLM tracker is computationally cheaper, it is prone to accumulating errors during runtime (drift), and typically fail if the object disappears from the camera view (see Figure 8 for an example). Hence, a combination of detector and tracking helps in achieving high performance with low execution time. Our solution brings further benefits by choosing parameters of the detection algorithm adaptively:

Performance on the KUL Dataset: We measure quantitatively the performance benefits in terms of execution time and accuracy of our proposed MDP-based approach for selecting good-enough parameters. The different thresholds considered in our performance evaluation are shown in Table I. We quantify the benefits of approximation in Table II. Here the execution time corresponds to the time taken to run the object detection workflow per frame. The results show that approximation at the parameter level can indeed bring benefits in terms of execution time for the KUL dataset. The first column of the table indicates the codes of these roads signs used in the dataset as shown in Fig. 4; for most of the traffic signs we get a gain in execution time for a small penalty of accuracy. For instance, for road signs E1, C1, and C31R, and C21 we are able to get a speed up (in percentage) of 45.7, 22.8, and 7.4, and 17.0 and accuracy loss of 4.0, 6.7, 5.0, and 0.0, respectively.

Performance with Varying Clutter and Illumination: Here we test the performance of our solution under different values of background clutter and illumination on our BU-RU dataset. In Fig. 9(a,b,c) we show the performance of the MDP when the background clutter is low (e.g., as in Fig. 2(a)) and when the background clutter is high (e.g., as in Fig. 2(c)). We quantify the background clutter using [15] as 4.05 ± 0.53 for high clutter, and 2.0 ± 0.63 for low clutter.

We compare the following scenarios: (i) Detection via fixed parameters (we run the object detection algorithm on each frame with the same parameters); (ii) Random parameter selection with tracker (detection is done using a random parameter tuple); (iii) MDP-based detection (we adapt the object detection parameters using our MDP, using the same parameters as long as the IoU of location detected by tracker and object detector is higher than a pre-defined threshold θ_{th}); (iv) Fixed parameter detection with tracking (fixed parameters are used for object detection, but the object detector is only run when the overlap of location detected by object tracker and template is less than a pre-defined threshold T_m); and finally (v) MDP-based detection with tracking (as the previous one, but the parameters of the object detector are identified by the MDP).

Background Clutter: Figure 9(a) shows results for low background clutter condition. The fixed parameter approach (Scenario i) of [18] has the best performance but it also takes the highest execution time. The MDP approach (Scenario iii) brings 39% gain in time 4% loss in accuracy. In the case

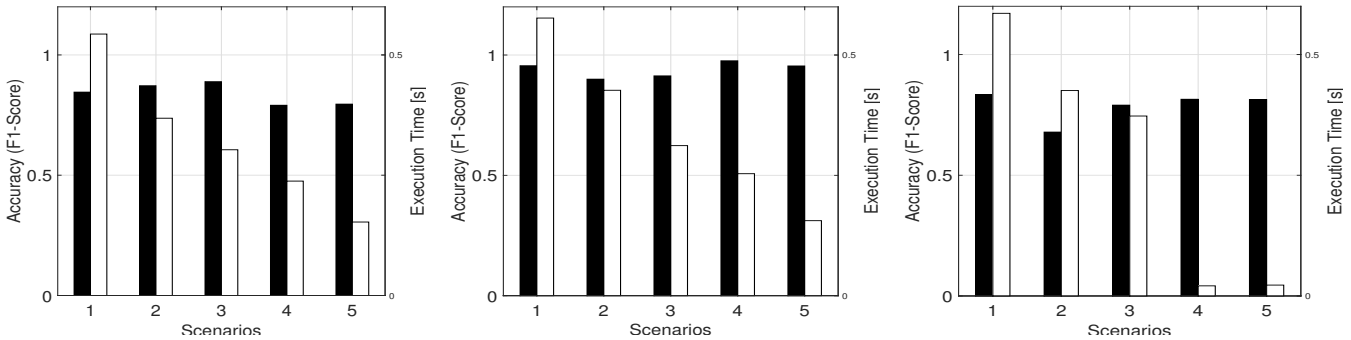


Fig. 9: **BU-RU dataset:** Performance of object detection in a video under different scenarios, (i) Fixed parameter, (ii) Random parameter, (iii) MDP-based parameter, (iv) Fixed parameter and tracking, and (v) MDP-based approach and tracking for different conditions (a) low and (b) high background clutter; (c) poor illumination.

where tracking is used (scenarios (i) and (iv)), we see that MDP based brings further gains in execution time of 35%. Triggering the MDP only at specific intervals (Scenario (v)) when the performance of tracker fails gives us a 50% better performance than triggering the MD at every frame (Scenario (iii)). The benefits of the MDP approach are visible also in the case of high background clutter, as shown in Fig. 9(b). In particular, we see that MDP-based approach with tracking (Scenario (v)) brings a 38% gain in execution time at a cost of 2% loss in accuracy.

Poor Illumination: In Fig. 9(c) we show the performance of the MDP when the illumination is low (e.g., as in Fig. 2(e)). We see that MDP approach (Scenario (iii)) again achieves a gain of 35% in comparison to fixed parameter approach (Scenario (i)). However, along with the tracker both fixed parameter and MDP-based approach give similar results because detector is not triggered frequently and is able to achieve acceptable performance (overlap of tracker and template above T_m) with the tracker. As, a result the execution time of this two scenarios is significantly lower than other results (since the tracker is computationally much cheaper than the object detector).

IV. Conclusions

We presented a new method to handle the problem of resource constraints in mobile robots. Specifically, we targeted the object detection problem and presented a solution based on Markov Decision Processes to select the parameters of computationally-intensive computer vision algorithms and bring benefits in terms of time and energy for long-term object-detection in videos. We showed the benefit of our formulation in a robot navigation setting with experiments on a public dataset and a new dedicated dataset via experiments. We showed a decrease in execution time of object detection and tracking application by $20 - 70\%$ with an accuracy of $98 - 100\%$ with respect to the fixed parameters approach used in previous works.

Acknowledgment: This work was supported by the NSF NRI Award No. IIS-1734362.

References

[1] A. Paolillo, A. Faragasso, G. Oriolo, and M. Vendittelli, "Vision-based maze navigation for humanoid robots," *Autonomous Robots*, vol. 41, no. 2, pp. 293–309, 2017.

[2] B. D. Gouveia, D. Portugal, D. C. Silva, and L. Marques, "Computation sharing in distributed robotic systems: A case study on SLAM," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 410–422, 2015.

[3] G. Mohanarajah, V. Usenko, M. Singh, R. D'Andrea, and M. Waibel, "Cloud-based collaborative 3D mapping in real-time with low-cost robots," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 423–431, 2015.

[4] D. Hunziker, M. Gajamohan, M. Waibel, and R. D'Andrea, "Rapyuta: The Roboearth Cloud Engine," in *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 2013.

[5] O. Zweigle, R. van de Molengraft, R. d'Andrea, and K. Häussermann, "RoboEarth: Connecting Robots Worldwide," in *Proc. of ACM International Conference on Interaction Sciences: Information Technology, Culture and Human*, Seoul, Korea, 2009.

[6] L. Paletta, G. Fritz, and C. Seifert, "Q-learning of sequential attention for visual object recognition from informative local descriptors," in *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005, pp. 649–656.

[7] S. Karayev, M. Fritz, and T. Darrell, "Anytime recognition of objects and scenes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 572–579.

[8] Y. Xiang, A. Alahi, and S. Savarese, "Learning to track: Online multi-object tracking by decision making," in *IEEE International Conference on Computer Vision*, 2015, pp. 4705–4713.

[9] K. M. Kitani, B. D. Ziebart, J. A. Bagnell, and M. Hebert, "Activity forecasting," in *European Conference on Computer Vision*. Springer, 2012, pp. 201–214.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[11] M. Mathias, R. Timofte, R. Benenson, and L. Van Gool, "Traffic sign recognition-How far are we from the solution?" in *Proc. of IEEE International Conference on Neural Networks (IJCNN)*, Dallas, TX, USA, 2013.

[12] J. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Advances in Large Margin Classifiers*, vol. 10, no. 3, pp. 61–74, 1999.

[13] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[14] C. L. Zitnick and P. Dollár, "Edge boxes: Locating object proposals from edges," in *Proc. of European Conference on Computer Vision (ECCV)*, Zurich, Switzerland, 2014.

[15] R. Rosenholtz, Y. Li, J. Mansfield, and Z. Jin, "Feature Congestion: A Measure of Display Clutter," in *Proc. of the ACM SIGCHI conference on Human Factors in Computing Systems*, Portland, Oregon, USA, 2005.

[16] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[17] C. Tomasi and T. Kanade, "Detection and tracking of point features," *International Journal of Computer Vision*, vol. 9, no. 3, pp. 137–154, 1991.

[18] B. Alexe, T. Deselaers, and V. Ferrari, "Measuring the objectness of image windows," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2189–2202, 2012.