Hardware optimizations for anytime perception and control

Nischal K.N., Paritosh Kelkar, Dhruva Kumar, Yash Vardhan Pant, Houssam Abbas
Joseph Devietti, Rahul Mangharam

I. Motivation

Autonomous vehicles promise significant benefits to society, from reduced accident rates to greater mobility for the elderly. The biggest challenge in the design of autonomous vehicles comes from the uncertainty of the environment in which they will operate. Their control algorithms must be able to cope with driving events that occur on widely ranging time scales. For example, relaxed rural driving can accommodate planning actions every few seconds, while imminent collision avoidance requires planning and actuation on the order of a few milliseconds. Thus 'real-time' performance will imply different things depending on the context.

A second, related, challenge is that the perception algorithms on-board these vehicles (like object detection based on video feed) must handle a very large amount of data, leading to increased power consumption. This is especially true for autonomous vehicles (AVs) since they carry multiple sensors (cameras, LIDAR, radars, ultrasound radars, etc) whose data must be processed in real-time to avoid accidents.

As a result of the variability in the environment, the control and perception systems are over-engineered to operate as if the worst-case conditions always hold. This results in unnecessarily high power consumption from the computation platform. It is worth noting that over the past few decades, the power consumption of processors has increased by more than double, while battery energy density has only improved by about a quarter [1].

In the current work, we explore the idea of trading-off computation time and power for quality of output, and how this trade-off affects *control performance*. We start from the observation that the best quality output from the perception algorithm is not always required for the system to achieve the desired control performance. For example, the control objective might be to follow the center of a driving lane, and control performance is measured by the deviation from that center. At slow speeds, poor quality of position estimate may be tolerated since it won't lead to excessive deviations from the center. Therefore we focus on *anytime perception algorithms* that have a pre-defined set of interruption times. In general, the earlier the algorithm is interrupted, the worse is the quality of its output. On the other hand, that quality may be sufficient for the control algorithm to achieve its goal.

*This work was supported by STARnet a Semiconductor Research Corporation program sponsored by MARCO and DARPA, NSF MRI-0923518 and the US Department of Transportation University Transportation Center Program

The Departments of Electrical and Systems Engineering and Computer and Information Sciences, University of Pennsylvania, Philadelphia, U.S.A. {nischal,paritosh,dhruvak,yashpant,habbas,rahulm}@seas.upenn.edu, devietti@cis.upenn.edu

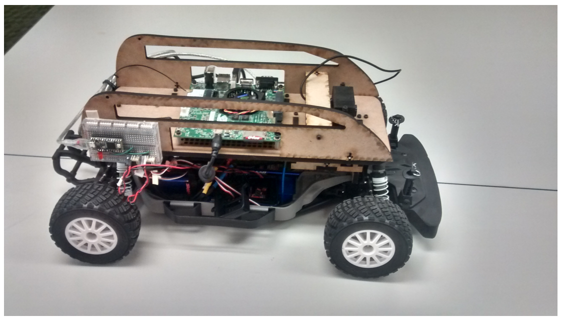

Fig. 1. Traxxas autonomous car with camera.

In [2] we proposed a way in which a standard perception algorithm can be turned into an anytime algorithm via off-line profiling, and thus can offer a time/power/quality trade-off. We also designed a model predictive controller than can make use of the trade-off offered by the anytime perception algorithm. To achieve the time/power/quality trade-off, we produced multiple versions of the perception algorithm. Broadly speaking, a version that ran for longer produced a higher quality output.

In this work, we turn our attention to achieving the time/power/quality trade-off using *platform-level* optimizations. Specifically, we work with an autonomous car $1/10^{th}$ the size of a regular car (Fig. 1). It is equipped with a front facing monocular camera and runs the Vanishing Point perception algorithm [3]. The on-board computation platform has both a CPU and GPU. More details of the experimental setup are in a later section. The platform-level optimization divides the Vanishing Point algorithm into components, and decides whether to run each component on the CPU or GPU, and at what frequency. The assignment to CPU or GPU is not static: every time the algorithm is executed, a different assignment may result, at a different frequency. The assignment is dictated by the controller: based on off-line profiling of each processor's performance on each component, the controller decides what quality is currently acceptable while minimizing power consumption. The profiling is described in more detail in a later section. In the present paper, we present initial results of partitioning the Vanishing Point algorithm and running different components on CPU and GPU, and at different frequencies. We also provide the related power numbers, demonstrating that a meaningful difference in runtime and power consumption exists depending on the assignment and frequency. In future work, we design a controller to make use of this and other optimizations, and relate the current results to the control performance.

II. Two stage approach for hardware optimizations

A. *Offline profiling of performance and power consumption of perception algorithm*

B. *Feedback driven online scheduling and hardware mode selection*

From the profiling stage of the perception algorithm **??**[section], we know that for a particular scheduling of the components of the algorithm on the CPU or the GPU, we have varying throughput and power consumption profiles with changing CPU-GPU frequencies. With this profile, and the fact that less delay results in better closed loop control performance of a system, we can take a step towards formulating a scheduling (tasks on CPU or GPU) and hardware mode (CPU-GPU frequencies) selection problem to be solved at run-time. Consider maximizing at each time step t, a cost function of the form:

$$\max_{\sigma,F_c,F_g} \alpha(t)\bar{\mathbf{T}}(\sigma,F_c,F_g) + \frac{1-\alpha(t)}{\mathbf{E}[\bar{\mathbf{P}}](\sigma,F_c,F_g)} \tag{1}$$

Here, σ denotes a schedule for tasks on the CPU or the GPU (see Fig. **??**), F_c and F_g denote the CPU and GPU frequencies respectively. Remember, from the profiling in Sec. **??**[section], that $\bar{\mathbf{T}}(\sigma,F_c,F_g)$ is the throughput of the vanishing point algorithm under schedule σ and with the CPU and GPU at frequencies F_c and F_g respectively. Similarly, $\mathbf{E}[\bar{\mathbf{P}}](\sigma,F_c,F_g)$ is the mean power consumed by the computation system at schedule σ and with the CPU and GPU at frequencies F_c and F_g respectively. $\alpha(t) \in [0,1]$ is a time-varying weight that decides how much to weight throughput versus performance at time step t. Note, higher $\alpha(t)$ is, more throughput is weighed. This varying $\alpha(t)$ results in different schedules and CPU-GPU frequencies for different times. It is worth noting that for a particular value of $\alpha(t)$, the optimal solution to Eq. 1 can be computed by composing the cost function from the profiled data and doing a search across parameters. In order to tie this to control performance, we can make $\alpha(t)$ a function of some feedback that reflects control performance, e.g. vanishing point abscissa or middle point abscissa or both. The function $\alpha(t)$ can now be composed such that as the vanishing point/middle point abscissas take on high values, $\alpha(t)$ also increases, resulting in a lower delay at the cost of more computation power. On the other hand, when the closed loop system is near or at steady state (vanishing point/middle point abscissas are small), we can trade-off computation delay (higher) in order to lower the computation power consumption.

III. Simulations for closed loop control and hardware optimization

After profiling the performance and energy consumption of the Vanishing Point based perception algorithm, the next step is to leverage this information at run-time and close the loop with control based on the perception while being efficient with respect to computation energy.

A. *Robot dynamics and control*

In order to simulate the closed loop behaviour of the robot, we use a unicycle model for the dynamic and a non-linear feedback controller as explained in [4]. The unicycle dynamics are:

$$\begin{aligned} \dot{x} &= v\sin\theta \\ \dot{y} &= v\cos\theta \\ \dot{\theta} &= \omega \end{aligned} \tag{2a}$$

Here, v is the velocity of the robot, which we treat as a constant parameter for our setup. The co-ordinate frame is defined such that x is the distance of the robot from the middle of the corridor. Fig. **??**[figureofunicycle] shows this. The goal of the closed loop controller is to navigate this robot along the middle of the corridor. Note, the controlled variable is ω, the desired angular velocity of the robot.

With the vanishing point based perception algorithm, the measurements from this system are the vanishing point and middle point abscissas as measured from processing the images from a front facing camera mounted on the robot. Using the geometry of the image frame as explained in [4], these measurements are

$$\begin{aligned} x_v &= k_1 \tan\theta \\ x_m &= k_2 \frac{x}{\cos\theta} + k_3\tan\theta \end{aligned} \tag{3a}$$

Note, Eq. 3 shows that the vanishing point depends only on the orientation of the robot and the middle point depends on both orientation and position. The objective of our robot is to traverse a corridor while being as close to the middle of the corridor. In order to realize this, we need to bring both x_v and x_m to converge to zero. A non-linear controller based on the non-linear dynamics of the robot (Eq. 2 and the measurements of x_v and x_m to achieve this is (from [4])

$$\omega = \frac{k_1}{k_1k_3 + x_m x_v}\left(-\frac{k_2 v}{k_1}x_v - k_p x_m\right) \tag{4}$$

Here, k_p is a positive proportional constant. To this controller, the time taken for computation of the vanishing point and middle point is a delay. In general, less delay means better control performance but as seen from the profiling of the vanishing point algorithm, this also means more computation power. Our goal is to achieve a trade-off where the control performance is acceptable (robot converges to the middle of the corridor) while the computation energy is minimized.

B. Online scheduling and mode selection

With the feedback in form of the vanishing point abscissa (x_v) and middle point abscissa (x_m), we can experiment with different functions to map the feedback to the weight $\alpha(t)$ in order to maximize the cost function of Eq.1 and hence choose a schedule for the perception algorithm tasks and the CPU-GPU frequencies (see Sec. II-B). Since we know that higher values of x_v and x_m imply more work has to be done to align to the corridor and/or get to the center of the corridor, we can have functions for $\alpha(t)$ that are monotonically increasing with $x_v(t)$ and $x_m(t)$. Note, for our simulations we scale x_v and x_m to be between -1 and 1. For the simulations, we try out

$$\alpha(t) = f_1(x_v(t)) \text{ s.t.,} \begin{cases} \alpha(t) = 0, & \text{if } x_v(t) \in [-d, d] \\ \alpha(t) = x_v(t) + d, & \text{if } x_v < -d \\ \alpha(t) = x_v(t) - d, & \text{if } x_v > d \end{cases} \tag{5}$$

This function depends only on the vanishing point abscissa x_v. The dead-zone magnitude, d is chosen to be 0.15 for our setup. Alternatively, we can try a similar function that depends only on the middle point abscissa.

$$\alpha(t) = f_2(x_m(t)) \text{ s.t.,} \begin{cases} \alpha(t) = 0, & \text{if } x_m(t) \in [-d, d] \\ \alpha(t) = x_m(t) + d, & \text{if } x_m < -d \\ \alpha(t) = x_m(t) - d, & \text{if } x_m > d \end{cases} \tag{6}$$

IV. Simulation results with profiled data

Acknowledgments

We would like to thank Kuk Jang for his help in creating diagrams in this paper.

References

[1] K. Lahiri, A. Raghunathan, S. Dey, and D. Panigrahi, "Battery-driven system design: A new frontier in low power design," *VLSI Design*, 2002.

[2] Y. V. Pant, H. Abbas, K. Mohta, T. X. Nghiem, J. Devietti, and R. Mangharam, "Co-design of Anytime Computation and Robust Control," *To be published, Proc. of Real Time Systems Symposium*, 2015.

[3] R. F. Vasallo, H. J. Schneebeli, and J. Santos-Victor, "Visual navigation: Combining visual servoing and appearance based methods," *6th Intl. Symp. on Intelligent Robotic Systems*, 1998.

[4] A. Faragasso, G. Oriolo, A. Paollilo, and M. Vendittelli, "Vision-based corridor navigation for humanoid robots," *IEEE International Conference on Robotics and Automation*, 2013.