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

Real-time control of autonomous vehicles requires the processing of a large amount of sensor data, which is used by the vehicle to determine its position in the world and to calculate its next move. Examples include data from cameras, LIDAR, radars and ultrasound radars, and possibly information communicated by other vehicles or the road infrastructure. The perception algorithms that process this data must do so fast enough for the control algorithm to compute the next input within the time constraints imposed by the environment. These time constraints are variable: 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. As a result of the variability in the environment, and to guarantee adequate performance in the most demanding conditions, the control and perception systems are over-engineered to operate as if the worst-case conditions always hold. Namely, the hardware processors always execute the perception code at a frequency that guarantees the fastest completion time. This results in unnecessarily high power consumption from the computation platform, as computation power increases with frequency. Note 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].

The usage of *anytime perception algorithms* allows us to perform a trade-off between the computation time of the algorithms, their power consumption, and the quality of their output. An anytime algorithm has a pre-defined set of interruption times. The earlier the algorithm is interrupted, the less power it consumes, but the worse is the quality of its output in general. On the other hand, that quality may be sufficient for the control algorithm to achieve its goal *in the current circumstances*. For example, in this paper, the control objective is 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, the perception algorithm might be interrupted early thus saving on computation power, *provided it gives a good enough estimate of position.*

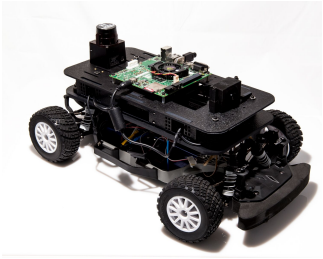

Fig. 1. Traxxas autonomous car with camera.

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 the time/power trade-off *for a fixed quality of output* and how it can be achieved using *platform-level* optimizations. Even when the output quality is fixed, the computation delay (equivalently, throughput) is known to affect control performance. Thus in this paper, we study how platform-level optimizations affect the computation throughout and power, and how to use this trade-off to save computation power without overly degrading throughput.

Note that the study of how computation delay affects control performance, and the design of anytime perception and control algorithms for power saving, are not specific to autonomous vehicles. Other control systems can benefit from these trade-offs, especially power-limited consumer robots. In this paper, we illustrate our approach on an autonomous car $1/10^{th}$ the size of a regular car (Fig. 1), which uses Vanishing Point navigation [3]. The setup, including the navigation algorithm, are presented in Section ?? Section ?? describes the offline profiling of Vanishing Point, which gives us Throughput vs Power curves for various processor frequencies and various scheduling of the navigation code on CPU and GPU. In Section ?? we combine power and throughput into one objective function, and design a supervisor what will determine the frequency and CPU/GPU allocation to maximize the objective. Section ?? presents

*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

experimental results that demonstrate the effect of the trade-off on control performance.

II. Problem Setup

In this work, we focus on the particular case of autonomous corridor navigation. For this purpose we use the vanishing point algorithm [3], [4] and feedback control to maintain heading parallel to the corridor and stay in the middle of it. We implement our algorithm on a $1/10^{th}$ scale car we developed.

A. The hardware

For our experiments, we converted a Radio controlled Traxxas Rally car into an autonomous robot shown in Fig. **??**[carFig], similar to the ones used in [5]. The computation platform is a NVIDIA Jetson TK1. The Jetson has a quad-core ARM Cortex-A15 CPU and a NVIDIA Kepler GPU. This setup allows us to schedule tasks on the CPU or GPU while observing the effect of this on power consumption and timing of the algorithm. The Jetson runs Ubuntu for Tegra as the operating system, and the algorithms are implemented using ROS [6]. The control signals for the drive and steer motor on the platform are generated by a Teensy 3.1 microcontroller running a ROS node which converts the continuous output of the control software to a PWM signal that acts as an input to the motor controller on the Traxxas. Finally, the vanishing point algorithm implemented in OpenCV [7] gets images from a front facing Point Grey Firefly MV camera capable of recording color images at upto a resolution of 752x480 pixels and upto a frame rate of 60 FPS.

B. Vanishing point-based corridor navigation

The Vanishing point algorithm [3] has been used extensively in indoor settings for navigating corridors autonomously [4], [8] and for outdoor lane detection [9]. The algorithm outputs the horizontal distance of the frame's vanishing point and middle point from the center of the frame. These two measurement signals are used as feedback by the controller to center and align the robot along the corridor. Figure 2 shows a visualization of Vanishing point operating on a frame collected from our robot navigating in a corridor.

We focus on the major computational tasks that comprise the vanishing point algorithm, which are briefly explained below:

- Blur: A Gaussian blur is applied on the image for denoising.
- Edge detection: We use the Canny Edge detector to find edges in the image.
- Hough Transform: used to detect straight lines in the image.
- RANSAC: used to select the parallel straight lines that best describe the sides of the corridor. These lines intersect in the image plane at the Vanishing Point.

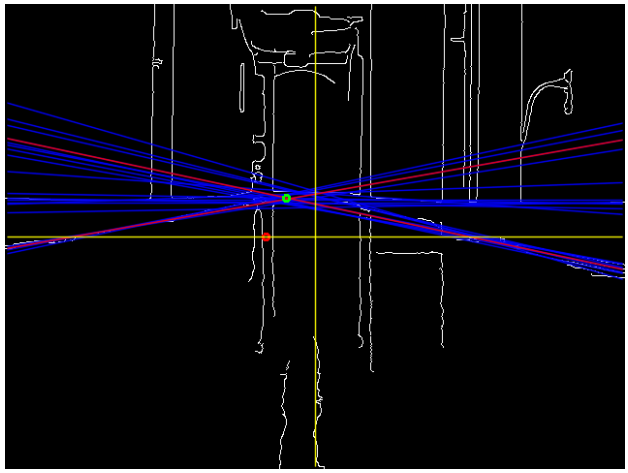

Fig. 2. Visualization of the Vanishing point algorithm. The green dot shows the vanishing point while the red dot shows the middle point.

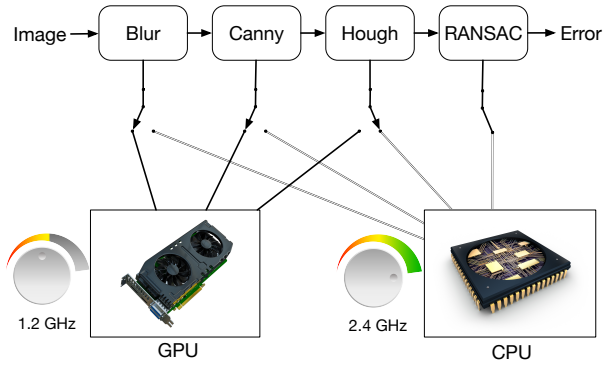

Fig. 3. The vanishing point algorithm with components running on either CPU or GPU at various frequencies, resulting in different power consumptions and execution times.

C. Problem statement

Note, on the Jetson, we can schedule any of these tasks to be run on either the CPU or the GPU as shown in Fig. 3 In addition, we can also change the CPU and GPU frequencies during run-time, resulting in different execution times for the vanishing point algorithm and different power consumption for the Jetson. With this in mind, the problem we want to solve is that of picking the best operating mode for the perception algorithm, the vanishing point algorithm in our case, in order to minimize computation energy without overly affecting the closed loop control performance of the system. For this, we propose a two step solution and evaluate it for out problem setup. This is explained in more detail in the following sections.

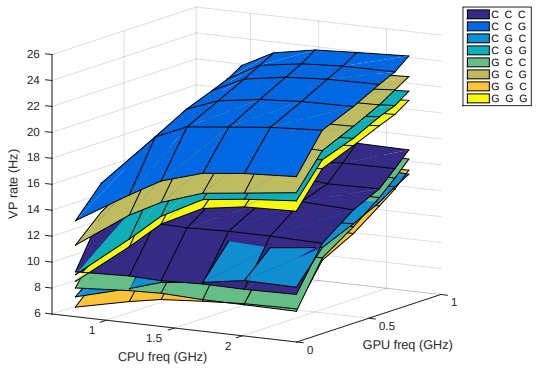

Fig. 4. Vanishing point algorithm update rate for different CPU-GPU assignments at varying frequencies (color in online version).

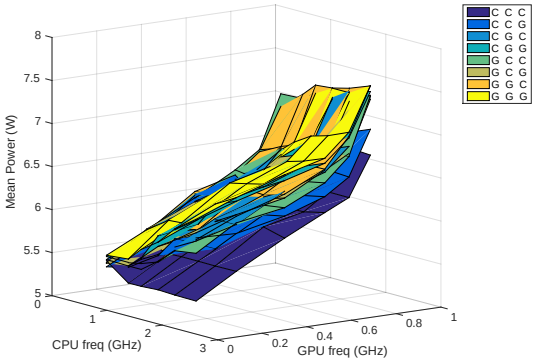

Fig. 5. Mean power consumed by the Jetson for different CPU-GPU assignments at varying frequencies. (Color in online version)

III. Two stage approach for hardware optimizations

A. Offline profiling of performance and power consumption of the 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 normalized 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 normalized 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.

IV. 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

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

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])

$$\omega = \frac{k_1}{k_1 k_3 + x_m x_v}\left(-\frac{k_2 v}{k_1} x_v - k_p x_m\right) \tag{4}$$

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. III-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

$$\alpha(t) = f_1(x_v(t)) \text{ s.t., } \begin{cases} \alpha(t) = 0.001, & \text{if } x_v(t) \in [-d, d] \\ \alpha(t) = x_v(t) + d, & \text{if } x_v < -d \\ \alpha(t) = x_v(t) - d, & \text{if } x_v > d \end{cases} \tag{5}$$

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.

$$\alpha(t) = f_2(x_m(t)) \text{ s.t., } \begin{cases} \alpha(t) = 0.001, & \text{if } x_m(t) \in [-d, d] \\ \alpha(t) = x_m(t) + d, & \text{if } x_m < -d \\ \alpha(t) = x_m(t) - d, & \text{if } x_m > d \end{cases} \tag{6}$$

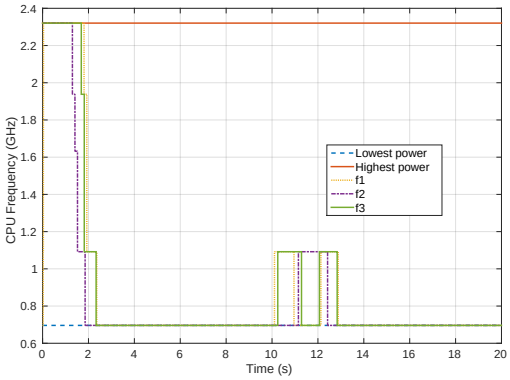

Fig. 6. CPU Frequency selected versus time.

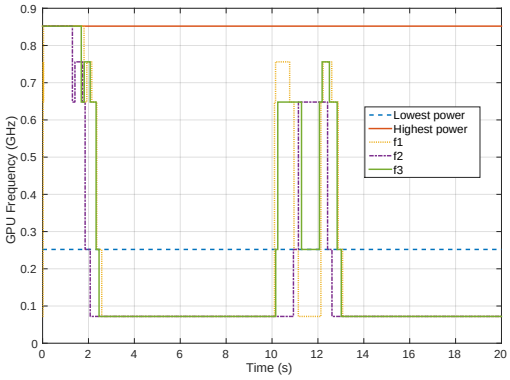

Fig. 7. GPU Frequency selected versus time.

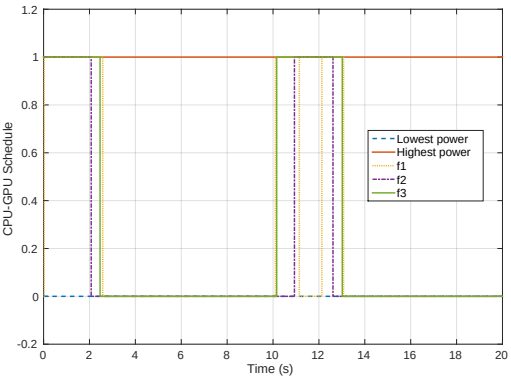

Fig. 8. Resource allocation schedule for the vanishing point algorithm.

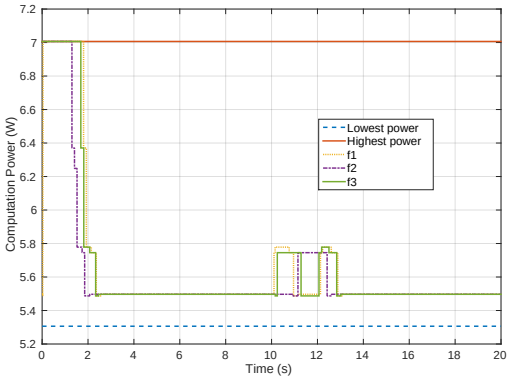

Fig. 9. Expected computation power for running the vanishing point on the Jetson.

V. Simulation results with profiled data

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.

[5] MIT IAP class, "RACECAR," 2015. [Online]. Available: http://sertac.scripts.mit.edu/racecar

[6] The Open Source Robotics Foundation, "ROS," 2015. [Online]. Available: www.ros.org

[7] Itseez, "OpenCV," 2015. [Online]. Available: www.opencv.org

[8] J. M. Toibero, C. M. Soria, F. Roberti, R. Carelli, and P. Fiorini, "Switching Visual Servoing Approach for Stable Corridor Navigation," *IEEE International Conference on Advanced Robotics*, 2009.

[9] A. C. Gallagher, "A ground truth based vanishing point detection algorithm," *Pattern Recognition*, vol. 35, no. 7, pp. 1527–1543, 2002.