Power-efficient algorithms for autonomous navigation

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

***Abstract*— Real-time navigation of autonomous vehicles requires the processing of a large amount of sensor data by the perception algorithms onboard the vehicle, like object detection and localization. To meet the driving performance and safety requirements, these algorithms require the hardware to be over-engineered to always operate for the worst-case. This leads to excessive power consumption by the computation platform. In this paper, we study how platform-level optimizations affect the computation throughput and power, and how to use this trade-off to save computation power without overly degrading throughput and control performance. The approach uses an offline profiling stage of the perception algorithm, which gives us Throughput versus Power curves for various processor frequencies and various scheduling of the perception code on CPU and GPU. At runtime, 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. We illustrate our approach on a scaled-down autonomous car which uses Vanishing Point navigation. Experimental results demonstrate that we can achieve an energy savings of upto 20% while degrading control performance by less than 1%.**

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. Google's autonomous vehicles generate over 750MB/s of sensor data [1] which must be processed by the perception pipeline fast enough with run-to-completion algorithms. To guarantee safety and meet the driving performance requirements, such run-to-completion algorithms require the hardware to be over-engineered for the worst-case: i.e., it always executes the software as if the worst-case conditions hold. This leads to a requirement of over 4KW in computational power. This is a significant drain on the vehicle's lithium-ion battery capacity, which is 24kWh in a Nissan Leaf for example, and a significant percentage of the average power drawn by the drive motors, which is 30kW in a small electric vehicle modeled in ADVISOR [2] going through the Urban Dynamometer Drive Cycle.

*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. {yashpant,habbas,nischal,paritosh,dhruvak,rahulm}@seas.upenn.edu, devietti@cis.upenn.edu

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.*

In [3] 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. 2), which uses Vanishing Point navigation [4]. The setup, including the navigation algorithm, are presented in Section II. Section III describes the offline profiling of Vanishing Point, which gives us Throughput versus Power curves for various processor frequencies and various scheduling of the navigation code on CPU and GPU. In Section III-C we combine power and throughput into one objective function, and design a

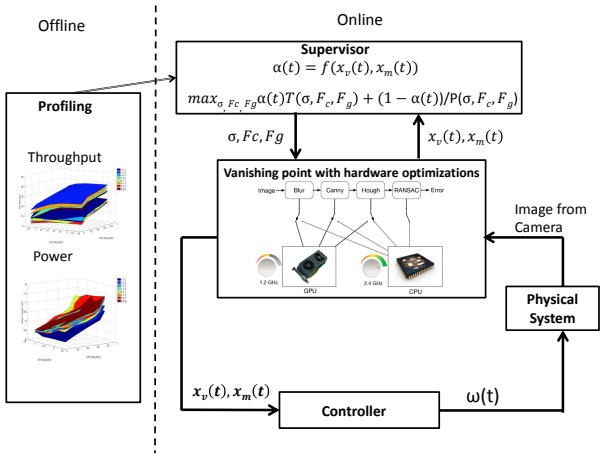

Fig. 1. Two stage approach.

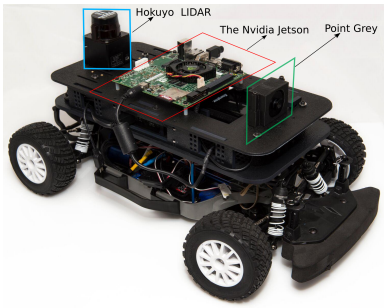

Fig. 2. Autonomous car ($1/10^{th}$ scale) developed using a Traxxas chassis. The car is capable of speeds of up to 40 miles per hour.

supervisor what will determine the frequency and CPU/GPU allocation to maximize the objective. Section IV presents experimental results that demonstrate the effect of the trade-off on control performance.

II. Problem Setup

In this section, we present the setup in which we study the throughput/power trade-off and its effect on control performance. Specifically, we developed a $1/10^{th}$ scale autonomous car. The car runs the Vanishing Point algorithm [4], [5] and a feedback controller to navigate a corridor and stay in its middle. The computation platform on board the car is a Nvidia Jetson, which has a quad-core ARM CPU and a 192 core Nvidia Tegra GPU.

A. Vanishing point for corridor navigation

The Vanishing Point algorithm (VP) [4] has been used extensively in indoor settings for navigating corridors autonomously [5], [6] and for outdoor lane detection [7]. For each image frame, the algorithm outputs the horizontal distances x_v of the vanishing point and x_m of the middle point from the center of the frame. The vanishing point is the intersection point of any two parallel lines in the environment, and the middle point indicates the center of the corridor. See Fig. 3. These two measurements are used by the feedback controller to center the robot in the corridor and align it with the walls. It does so by driving the abscissa

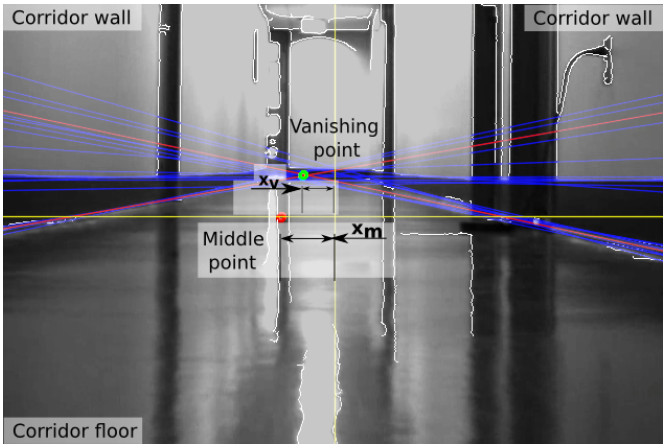

Fig. 3. Vanishing point algorithm output overlaid on the corridor image. The green (upper) dot shows the vanishing point while the red (lower) dot shows the middle point. These are computed from the intersection of the detected corridor guidelines. (Color in online version)

x_v and x_m to zero using the following feedback law for the steering rate ω [5]:

$$\omega = \frac{k_1}{k_1 k_3 + x_m x_v}\left(-\frac{k_2 v}{k_1} x_v - k_p x_m\right) \tag{1}$$

where k_p, k_1, k_2, k_3 are parameters, and the car dynamics are modeled as follows:

$$\dot{x} = v \sin\theta,\ \dot{y} = v\cos\theta,\ \dot{\theta} = \omega \tag{2}$$

where x is the car's horizontal position (referenced to an origin in the middle of the corridor), y is its vertical position, and θ is the steering angle. The car's velocity v is fixed. The VP computation time appears as a delay to this controller. In general, less delay means better control performance but larger computation power. Our goal is to achieve a trade-off where the control performance is acceptable (robot drives along the middle of the corridor) while the computation energy is minimized.

We define *throughput* as the update rate of VP, which is the inverse of its execution time. The faster VP executes, the better the control performance of the closed loop system since the controller sees a small delay. Hence, throughput acts as a proxy for control performance. In most implementations, the perception algorithm is always run at its maximum possible throughput to subject the controller to the smallest possible delays. This neglects the power consumed by the computation platform. In many autonomous systems, power draw from the computation platform is a significant concern; e.g., in our robot, the Jetson and drive motors are powered by separate energy sources. So while we would want to subject the controller to a small delay (operate the perception algorithm at a high throughput), we would also like to minimize the power draw from the computation platform in order to maximize the operating time of the system.

B. Exploiting hardware knobs to trade-off power and throughput

In order to trade-off computation power and throughput of the perception algorithm, we rely on the insight that the GPU can execute some tasks faster than a CPU, albeit at a greater power cost. Also, executing a task at a higher frequency (on either CPU or GPU) increases throughput, but again at a greater power cost. Thus, in our setup, we found that running VP on the CPU alone resulted in a low throughput (about 8Hz) and low power consumption (about 5W). On the other hand running it on the GPU allowed us to get a throughput in excess of 20Hz, but resulted in a power consumption of over 7W. In this section, we describe how to divide VP into tasks and profile VP's performance as we vary the execution frequencies of these tasks and their scheduling on either CPU or GPU. These tasks are (see Fig. 4):

- Blur: A Gaussian blur is applied on the image for de-noising.
- 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.

We can schedule any of these tasks to be run on either the CPU or the GPU as shown in Fig. 4. Let $\sigma \in \Sigma$ denote a given schedule, where

$$\Sigma = \{\text{CCC, CCG, CGC, CGG, GCC, GCG, GGC, GGG}\}$$

For example, schedule $\sigma =$ CCG means that the Blur task is done on the CPU, the Edge detection is done on the CPU and the Hough Transform is done the GPU, and so on. Since RANSAC took a negligible amount of time compared to the other tasks, we always execute it on the CPU. In addition, we can change the CPU and GPU frequencies during run-time, resulting in different execution times and power consumption for the Jetson. Let F_c and F_g be the frequencies of the CPU and the GPU respectively.

The hardware level knobs to trade-off throughput and computation power for an execution of the vanishing point algorithm are now σ, F_c and F_g. The throughput and computation power, functions of all three knobs, are denoted by $T(\sigma, F_c, F_g)$ and $P(\sigma, F_c, F_g)$ respectively.

C. The Mode Selection problem

The problem we solve is that of picking the best operating mode (σ, F_c and F_g) for the perception algorithm VP in order to minimize computation power $P(\sigma, F_c, F_g)$ without overly affecting the closed loop control performance of the system, quantified by the throughput $T(\sigma, F_c, F_g)$.

III. Two stage optimization

To solve the Mode Selection problem (Section II-C), we propose a two stage solution. The first stage is offline (Fig. 1 left): in it, we thoroughly profile the perception algorithm's throughput $T(\sigma, F_c, F_g)$ and power consumption $P(\sigma, F_c, F_g)$ for various values of the hardware knobs σ, F_c and F_g. The second stage is at runtime (Fig. 1 right): based on the current control error, a supervisor chooses a weighting of throughput and power; intuitively, the larger the error, the more weight throughput receives, and vice-versa. The supervisor then picks the task schedule σ and CPU-GPU frequencies F_c, F_g that best maximize the weighted combination of throughput and power. A high-level view of this approach is shown in Fig. 1. We next detail each stage.

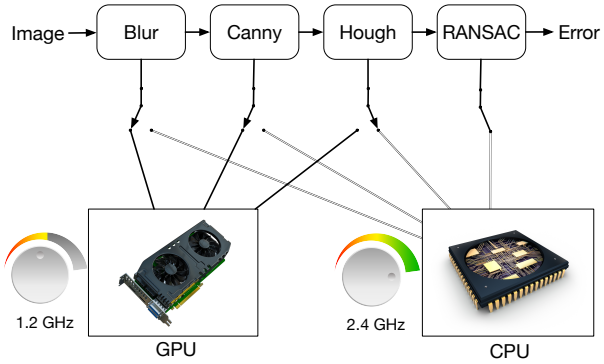

Fig. 4. The vanishing point algorithm with components running on either CPU or GPU at various frequencies, resulting in different power consumptions and execution times.

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

For the Vanishing Point (VP) algorithm, the first stage of our method is profiling the timing and power consumption of the computation. Namely, we vary the schedule and frequency knobs described in Section II-B. For each value of the knobs, we run VP on a video sequence previously acquired by the robot while navigating a corridor, and log power and execution time. Since for an algorithm like VP there is no well-defined notion of output quality, we use the update rate as a performance measure, since faster updates mean that the car controller has less delay, resulting in better control performance.

Figure 5 shows the profiling results for the throughput of VP at different CPU-GPU allocations of the 3 tasks and different frequencies of the CPU and the GPU. Note, the CPU can be clocked upto 2.32 GHz (on all 4 cores), while the GPU can be clocked upto 0.852 GHz. We select 6 operating frequencies evenly spaced between the minimum and maximum CPU and GPU frequencies.

Figure 6 shows the profiling of average power consumed by VP over all frames in the video for the same combinations of the knobs.

B. Feedback driven online scheduling and mode selection

The profiling results indicate that the knobs σ, F_c and F_g allow us to trade-off throughput for power, as expected. At runtime, we must decide which knob setting to choose at every time step. This is done by maximizing the following objective function at every time step t:

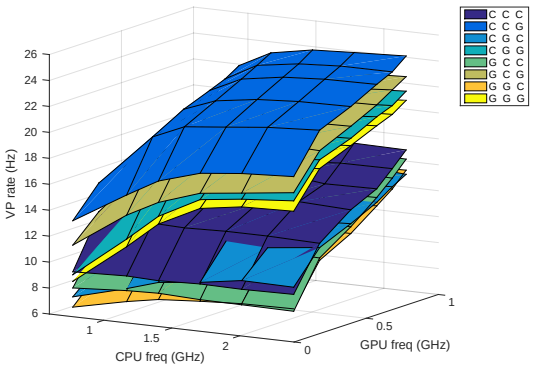

Fig. 5. VP throughput. Each surface corresponds to a particular schedule (see legend). For a given schedule, different CPU and GPU frequencies yields a different throughput. (Color in online version).

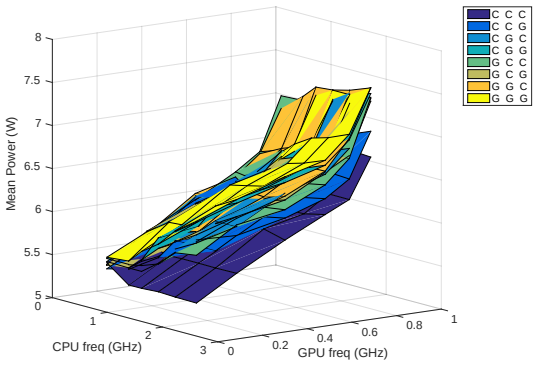

Fig. 6. Mean power consumed by the Jetson. Each surface corresponds to a particular schedule (see legend). For a given schedule, different CPU and GPU frequencies yields a different power. (Color in online version).

$$\max_{\sigma,F_c,F_g} \alpha(x_m,x_v)\bar{\mathbf{T}}(\sigma,F_c,F_g) + \frac{1-\alpha(x_m,x_v)}{\mathbf{E}[\bar{\mathbf{P}}](\sigma,F_c,F_g)} \tag{3}$$

Recall that $\bar{\mathbf{T}}(\sigma,F_c,F_g)$ is the normalized throughput of VP, and $\mathbf{E}[\bar{\mathbf{P}}](\sigma,F_c,F_g)$ is the mean normalized power consumed by the computation platform. The parameter $\alpha(x_m,x_v) \in [0,1]$ determines how much to weigh throughput versus performance at every time step. It is computed as a function of the abscissa x_m, x_v: recall that non-zero values of either indicates deviation from the center line of the corridor (Section II-A). Since x_m, x_v are time-varying, α is also time-varying. As x_m or x_v deviates further away from 0, α increases to more heavily weigh throughput, thus skewing the optimization towards larger throughput and better control performance. This translates as different CPU-GPU schedules and different frequencies. In this paper, we use three different functions for α, given $d > 0$:

$$\alpha = f_1(x_v(t)) = \begin{cases} 0.001, & \text{if } x_v(t) \in [-d,d] \\ x_v(t)+d, & \text{if } x_v(t) < -d \\ x_v(t)-d, & \text{if } x_v(t) > d \end{cases} \tag{4}$$

$$\alpha = f_2(x_m(t)) = \begin{cases} 0.001, & \text{if } x_m(t) \in [-d,d] \\ x_m(t)+d, & \text{if } x_m(t) < -d \\ x_m(t)-d, & \text{if } x_m(t) > d \end{cases} \tag{5}$$

$$\alpha = f_3(x_m(t),x_v(t)) \begin{cases} 0.001, \text{if } |x_m(t)|+|x_v(t)| < d \\ |x_m(t)|+|x_v(t)|-d, \text{otherwise} \end{cases} \tag{6}$$

C. Feedback control of vehicle

The online control procedure performs the following calculations at ever time step t (see Fig. 1):

1) Obtain x_m, x_v from VP, and provide it to both Supervisor and Controller.
2) The Supervisor:
   a) Computes $\alpha(x_m,x_v)$
   b) For each value of (σ,F_c,F_g), computes the objective value (3). $\bar{P}$ and $\mathbf{E}[\bar{\mathbf{P}}]$ are obtained from the offline profiling stage.
   c) Selects the value of (σ^*,F_c^*,F_g^*) that maximizes the objective. This is provided to VP.
3) The Controller computes the input value as described in Section II-A.
4) VP executes with the CPU/GPU schedule σ^* and frequencies F_c^*, F_g^*. Goto 1.

IV. Experiments

We simulate the system shown in Fig. 1, where the car is modeled by the dynamics of Eq. (2), the controller is given by Eq. (1), and the supervisor optimizes Eq. 3 as explained in Section III-C. The controller experiences a delay when receiving update values of x_m and x_v. This is the computation delay computed offline during the profiling stage. The simulation is repeated three times: once for each choice of α from Eqs. (4), (5) and (6). We also simulate two more scenarios: one where the schedule and frequencies are fixed at the highest throughput (highest power) values, and one where they are fixed at the lowest throughput (lowest power) values.

We initialize the robot aligned to the corridor $\theta = 0$ but off from the center by 0.25m, i.e. $x = 0.25$. The controller attempts to bring the robot to the middle of the corridor and align it with the corridor while the supervisor decides the hardware operating mode (σ,F_c,F_g) as per Sec. III-B. At 10 seconds, we apply a steering disturbance to the system, which is a pulse of magnitude 3 degrees per second and a duration of 2 seconds. The controller again tries to recover from this disturbance while the supervisor picks the best operating mode for the perception algorithm.

Figs. 8 and 9 show the selected CPU and GPU frequency versus time respectively. Fig. 11 show the computation power vs time, while Fig. 10 shows the schedule σ of CPU-GPU allocation for tasks. Fig. 7 shows the trajectory of x versus time. Note that only the trajectory of x for the lowest power, or highest delay mode differs enough from the others so as to be different visually in this plot.

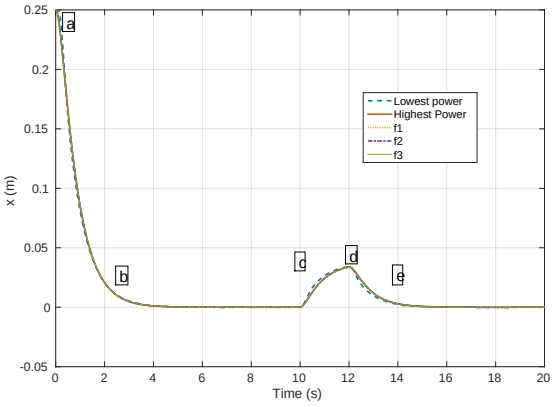

Fig. 7. x position of the robot. Note, for modes other than the lowest power (most delay), the trajectory of x is very similar.

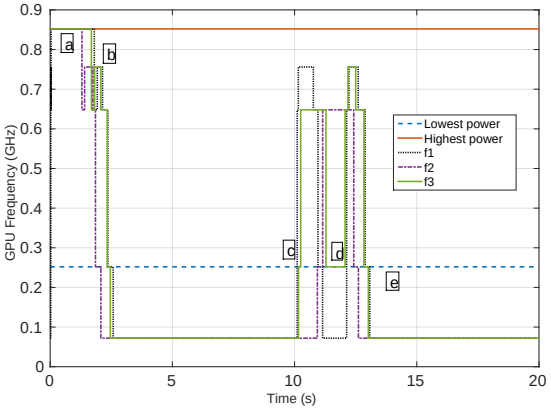

Fig. 9. GPU Frequency selected versus time. Note, for the lowest power mode, the GPU is at its second lowest frequency and not the lowest (while the schedule is $\sigma = CCC$) because of noise in power measurements.

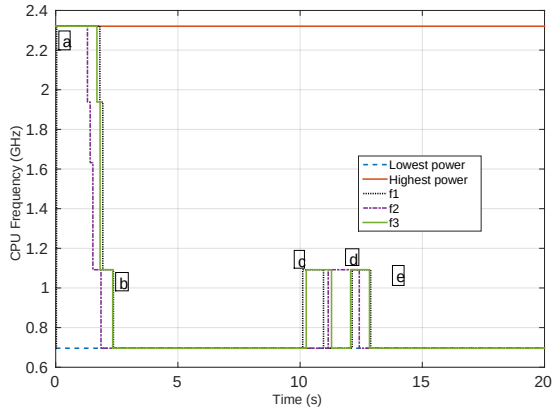

Fig. 8. CPU Frequency selected versus time.

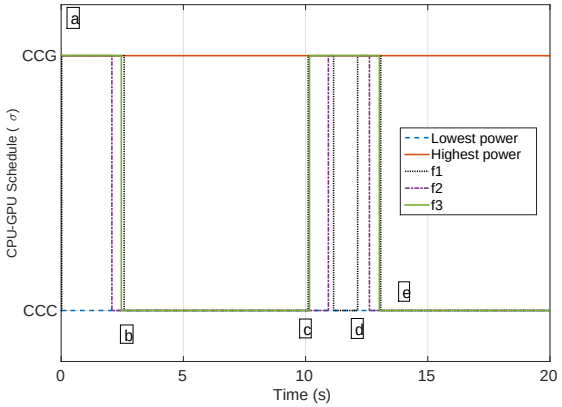

Fig. 10. CPU/GPU schedule for VP. Note that the schedule switches between only two allocations: CCG which has the best throughput with a relatively low power consumption, and CCC has the lowest power consumption (See Section III-A)

To better understand these figures, let us go through 5 checkpoints (**a**, **b**,**c**, **d** and **e**) in time common to Figs. 7, 8, 9, 10 ,11. At checkpoint **a** the controller starts to stabilize x initial displacement of 0.25m. At this time, both x_v and x_m have high magnitudes, implying α takes a high value (close to 1) for all $f_i,\ i = 1, 2, 3$. Due to this, Fig. 10 shows that σ becomes CCG and Figs. 8 and 9 show that CPU and GPU frequencies are high, this implies the vanishing point algorithm is near its highest throughput. Correspondingly, Fig. 11 shows that the computation power is also high.

At checkpoint **b** x has a small magnitude and is near settling the middle of the corridor. Because of this, both x_v and x_m have smaller magnitudes, and so α is small and the requested CPU and GPU frequencies start to decrease. σ is still CCG for all supervisory functions except f_2, but settles to CCC for all 3 as x settles to zero shortly after checkpoint **b**.

Checkpoints **c**, **d**, and **e** show how the system responds to a pulse-like disturbance in the steering (lasting from 10s to 12s). It is interesting to see how the different supervisory functions f_i result in different switching between schedules CCC and CCG and different CPU and GPU frequencies.

The closed loop control performance can be measured over a simulation time of T seconds as $L = \int_0^T |x(t)|dt$, and the expected energy consumed is calculated as the sum of powers over T seconds. A summary of the control performance and computation energy consumed for these five cases is shown in table I. It is clear that operating with the highest power/lowest delay mode for the vanishing point results in the best control performance, but the computation energy is high. On the other hand, lowest power/highest delay mode results predictably in low power consumption and the worst control performance. With out two stage approach, control performance is very similar (less than 1% degradation) to the highest power mode, while the computation energy is significantly (about 20%) lower. This clearly shows the benefit of our approach.

V. Related Work

The focus of perception based algorithm for autonomous systems has been on computation speed and performance

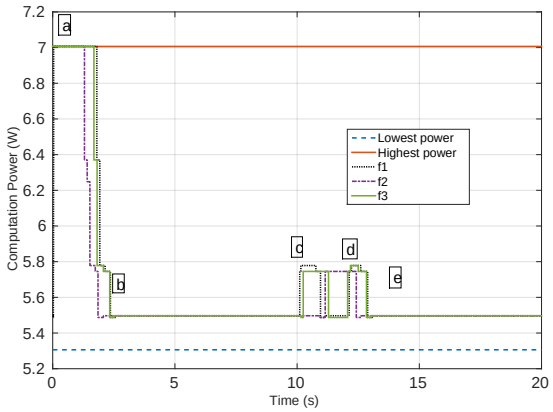

Fig. 11. Average computation power for running VP.

TABLE I
CONTROL PERFORMANCE AND COMPUTATION ENERGY

Supervisor	**Control perf.**(L)	**Energy**(J)
Lowest power (mode fixed)	0.3245	106.12
Highest power (mode fixed)	0.3010	140.13
$f_1(x_v)$	0.3025	113.16
$f_2(x_m)$	0.3023	112.39
$f_3(x_v, x_m)$	0.3018	113.07

with very little regard for power consumption, e.g. [8], [9]. In particular, computationally powerful but expensive GPUs are now becoming popular for implementing perception algorithms for autonomous navigation [9]. In this paper, our work focuses on trade-offs on performance of the perception algorithm to minimize computation power based on the feedback used for closed-loop control of the system. Unlike most implementations that rely on the GPU, we do not always schedule tasks to run on the GPU and vary GPU and CPU frequency at run-time in order to be power efficient without overly affecting control performance.

The effect of increasing computation time of a task on performance has been explored in [10] by using a resource allocation algorithm similar to QRAM [11]. Our work differs from this as we vary the resource allocation and execution time for the tasks that compose the perception algorithm at run-time while considering control performance and we also do not drop any tasks in order to meet the control requirements.

In the field of computer architecture approximate computing approaches [12], [13], [14] have been studied, seeking time or energy savings by performing a computation approximately instead of precisely. While our approach and approximate computing share a high-level goal, approximate computing lacks a feedback mechanism to balance computation and resources dynamically. Additionally the time and energy scale that our approach works at is much higher than what approximate computing looks at.

VI. CONCLUSIONS

In this paper, we introduce a two stage approach for hardware level optimization in order to trade-off throughput and power consumption of a perception algorithm. We use the vanishing point algorithm based corridor navigation as a running case study throughout the paper and simulate the closed loop performance based on the experimentally profiled vanishing point performance from a $1/10^{th}$ scale autonomous car. Through the simulations, we show that our run-time optimization for the trade-off results in negligible degradation of control performance ($\leq 1\%$) while significantly reducing computation energy (around 20%). In this work, one drawback is that the control algorithm itself is unaware of the trade-offs, unlike our work in [3]. Future work, in addition to developing a control algorithm that is aware of the trade-offs, will also focus on experimentally verifying the performance our method by closing the loop on the $1/10^{th}$ scale autonomous car and moving beyond simulations.

REFERENCES

[1] P. Diamandis and S. Kotler, *Bold: How to Go Big, Create Wealth and Impact the World*. Simon & Schuster, 2015.

[2] K. Wipke, M. Cuddy, D. Bharathan, S. Burch, V. Johnson, A. Markel, and S. Sprik, "Advisor 2.0: A Second Generation Advanced Vehicle Simulator for Systems Analysis," *Technical Report*, 1999.

[3] 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.

[4] 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.

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

[6] 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.

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

[8] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast Semi-Direct Monocular Visual Odometry," in *Robotics and Automation (ICRA), 2014 IEEE Intl. Conf. on*, IEEE, 2014.

[9] H. Zhang and F. Martin, "CUDA accelerated robot localization and mapping," in *Proc. of the IEEE Internation Conference on Technologies for Practical Robot Applications*, 2013.

[10] D. de Niz, L. Wrage, N. Storer, A. Rowe, and R. Rajukar, "On Resource Overbooking in an Unmanned Aerial Vehicle," *IEEE/ACM Third International Conference on Cyber-Physical Systems*, 2012.

[11] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "A Resource Allocation Model for QoS Mgmt.," *IEEE RTSS*, 1997.

[12] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, 2011.

[13] M. Carbin, S. Misailovic, and M. C. Rinard, "Verifying quantitative reliability for programs that execute on unreliable hardware," in *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages and Applications*, 2013.

[14] R. St. Amant, A. Yazdanbakhsh, J. Park, B. Thwaites, H. Esmaeilzadeh, A. Hassibi, L. Ceze, and D. Burger, "General-purpose code acceleration with limited-precision analog computation," in *Proc. of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, 2014.