Hardware optimizations for anytime perception and control

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

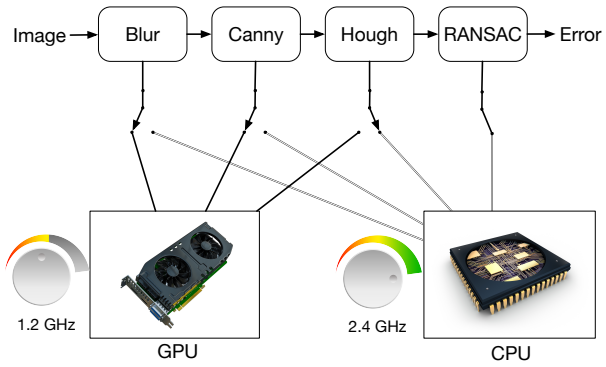

Fig. 1. The vanishing point algorithm with components running on either CPU or GPU at various frequencies, resulting in different power consumptions and execution times.

I. Two stage approach for hardware optimizations

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

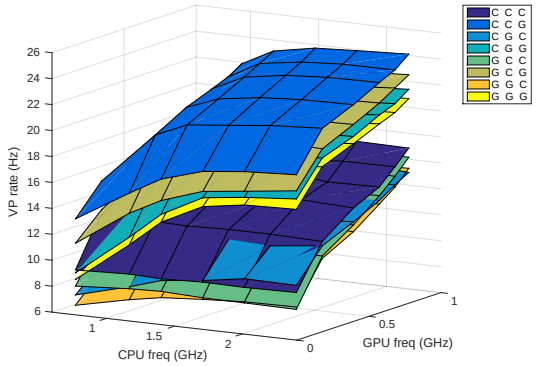

Fig. 2. Vanishing point algorithm update rate for different CPU-GPU assignments at varying frequencies (color in online version).

*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

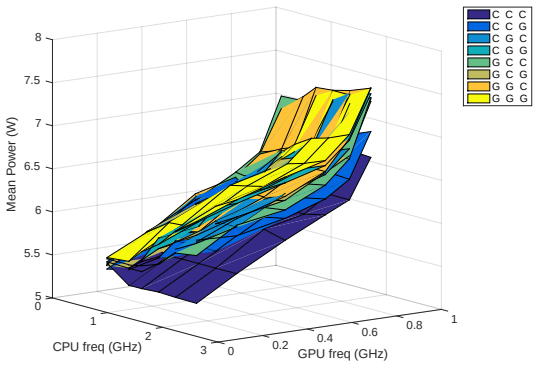

Fig. 3. Mean power consumed by the Jetson for different CPU-GPU assignments at varying frequencies. (Color in online version)

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:

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

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.

II. 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 [1]. 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 [1], 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 [1])

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

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. I-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.

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

III. Simulation results with profiled data

References

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

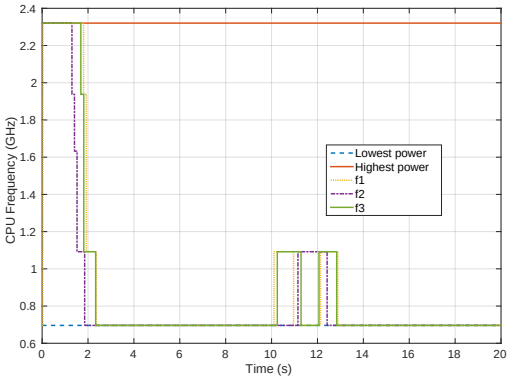

Fig. 4. CPU Frequency selected versus time.

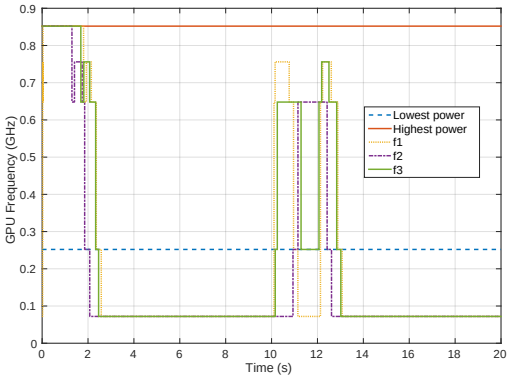

Fig. 5. GPU Frequency selected versus time.

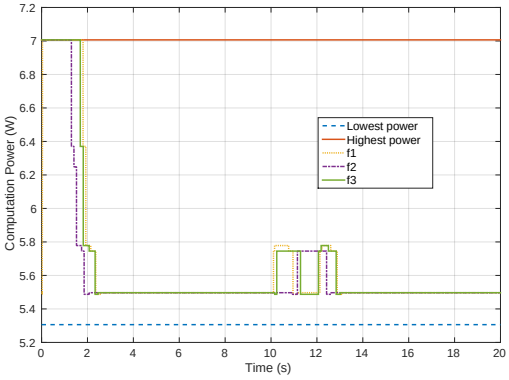

Fig. 7. Expected computation power for running the vanishing point on the Jetson.

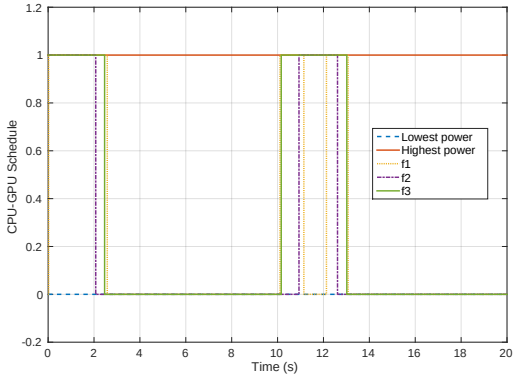

Fig. 6. Resource allocation schedule for the vanishing point algorithm.