Reactive Scheduling of Computational Resources in Control Systems

Gera Weiss
Ben-Gurion University
Beer-Sheva, Israel
geraw@cs.bgu.ac.il

Hodai Goldman
Ben-Gurion University
Beer-Sheva, Israel
hodaig@cs.bgu.ac.il

ABSTRACT

1. INTRODUCTION

Cyber-physical systems (CPS) technologies of integrating software and control are at the heart of many critical applications (c.f. [15]). These technologies aim at handling issues that emerge when the integration of software and hardware brakes the traditional abstraction layers: when researchers and practitioners are required to consider a unified view that includes both software and hardware. An example of such an issue is the challenge of dynamic assignment of computational resources to software based controllers discussed in, e.g., [3, 22, 24]. While the computation burden required by the control loops can be ignored in many situations, this is not always the case. A main motivating example studied in this paper is vision based control, where computer vision algorithms acquire state information to be used in a feedback loop (c.f. [7, 9, 20]). Unlike conventional sensors such as accelerometers, gyros, compasses, etc., a visual sensor requires significant processing of the acquired image to prepare the state information for feedback. Since typical cyber-physical application, such as robot control, consist of many control loops, responsible for different aspects of the system, that run simultaneously and share the same computational resources, the computer vision algorithms cannot always be invoked in full power. Alternatively, we propose in this paper a mechanism to dynamically trade CPU consumption vs. measurement accuracy so that data acquisition algorithms run in full power only when the control loop requires accurate data.

A main challenge in forming mechanisms for the integration of software and control lies in the design of efficient interfaces for integrating the engineering diciplines involved (c.f. [24]). Components with clearly specified APIs, such as Java library classes, allow designers to build complex systems effectively in many application domains. The key to such modular development is that an individual component can be designed, analyzed, and tested without the knowledge of other components or the underlying computing platform. When the system contains components with real-time requirements, the notion of an interface must include the requirements regarding resources, and existing programming languages provide little support for this. Consequently, current development of real-time embedded software requires significant low-level manual effort for debugging and component assembly (cf. [11, 14, 19]). This has motivated many researchers to develop compositional approaches and interface notions for real-time scheduling (cf. [5, 6, 8, 16–18, 21, 23]).

In this paper we present an approach, a proof-of-concept implementation, and a case-study in scheduling computations in embedded control systems. Our approach is based on the automata based scheduling approach, suggested in [1, 2, 25], where automata are proposed as interfaces that allow the dynamicity and efficiency of desktop operating systems with the predictability of real-time operating systems. The approach allows for components to specify the CPU resources that they need in a way that gives an application agnostic scheduler the freedom to choose schedules at run-time such that the needs of all the components are met, even of components that were added only at run-time. The main contributions of this paper relative to the earlier work in this direction is: (1) We propose an extension of the automata based scheduling frame work that allows to direct the schedule based on the state of the controllers; (2) We propose a technique, based on the theory of Kalman Filters, for designing reactively scheduled controllers; (3) We report on our experience with improving the performance of real-time a vision-based control system (a quadrotor that stabilizes itself in front of a window).

2. THE PROPOSED APPROACH

As proposed in earlier work on automata based scheduling (cf. [1,2,25]) we aim at a development process where a system is built as a composition of a set of components where each component is a class in, say C++, accompanied with an automaton. Our addition here is that we allow the automata to be guarded, i.e., the automaton acts as a specification of a reactive system that tells the scheduler which functions of a component it may run in each slot depending on the dynamic state of the controllers. Another addition is that we have implemented the approach as an enhancement of the internal scheduler of the Auto Pilot Mega (APM) open source unmanned vehicle autopilot software suite.

For scheduling tasks within a slot, we adopt the Logical Execution Time (LET) abstraction [12] that allows deterministic external interface even when the underlying physical execution layer introduces bounded non-determinism.

The relation between logical and physical task execution is depicted Figure 1 (adopted from [10]). The diagram depicts an execution of one task in one logical execution slot. Logically, the task is released at the beginning of the slot and terminates at the end of it, i.e., the task processes the inputs captured at the release time and its output are only made available to the outside world at the logical termination time. The actual execution, as shown in the figure, may start and finish anywhere in the logical execution slot and the execution may be interrupted by higher priority tasks. However, the external interface remains deterministic provided only that all tasks always finish within logical execution slots.

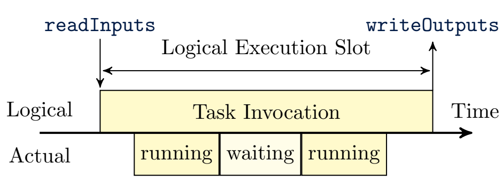


Figure 1: Execution of a task in an execution slot.

Diverging from the way LET is applied in other architectures (c.f. [5,12]), our architecture executes a dynamic assignment of tasks to logical execution slots. Specifically, the set of tasks that run in each logical execution slot is chosen from the composition of all resource specifications of the components. The scheduler has a complete view of resource requirements and can allocate them accordingly. For example, the system we can react

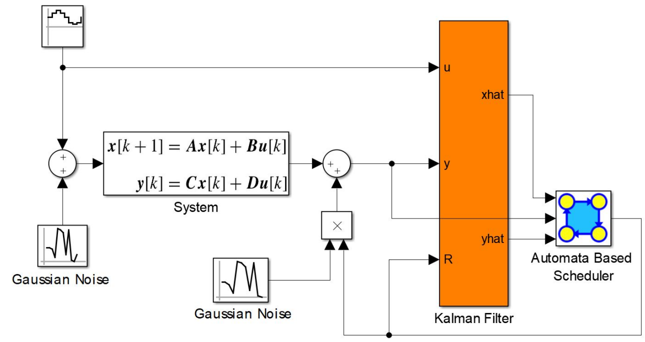


Figure 2: A Simulink model demonstrating our approach.

to an increased need of CPU of one of the components by reducing the non real-time computations.

3. A DEMONSTRATION OF THE APPROACH IN SIMULATION

Our approach for using automata for scheduling resources in software based controllers is based on the observation that in most systems the computational load is in the implementation of the sensors and of the actuators, not in the implementation of the controllers that usually consist of quick arithmetic manipulation of a small amount of variables. We therefore focus our attention on allowing a trade-off between CPU usage of sensors and actuators and their accuracy. In this paper, as a main motivating example, we focus on the case of visual sensors that employ image processing algorithms.

As said in Section 2, we propose to implement the resource scheduling decisions using automata that control which procedures are invoked in the control loops. More specifically, for sensors or actuators that require heavy computations, such as vision based sensors, we propose that the software engineers develop several modes of sensing, each consumes a different amount of CPU and provides a different level of accuracy.

Formally, we assume that the system is given as a Linear Time Invariant system, as depicted in Figure 2. The inaccuracies of the sensors and of the actuators are modeled as additive Gaussian noise. We assume that each sensor can be operated in a range of modes (more than one), each mode consuming a certain percentage of the CPU and giving a certain variance of the measurement noise.

The scheduling of the modes is governed by the automata based scheduler as depicted in Figure 2. We propose to use a standard Kalman filter for the purpose. The filter gets as input the actuations and the measurements (the sum of the output and the noise) and also the variance of the measurement noise, which we assume is a function of the sensor mode chosen by the scheduler. The output of the Kalman filter and the measurement are fed to the scheduler (in addition to, possibly, sending them to the controller) that uses them for deciding the next mode of the sensor. In the Simulink model depicted in Figure 2, the scheduler feeds the variance of the disturbance to the Kalman filter and to the block that multiplies the noise by the variance (the product of a white noise with unit variance and a constant C yields a normally distributed noise with variance C).

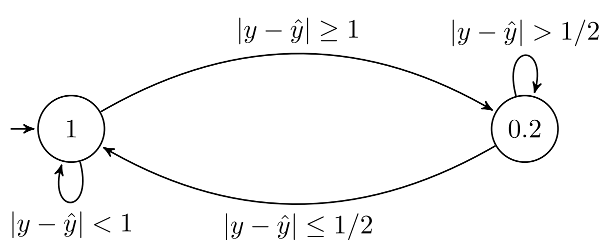
We ran the model depicted in Figure 2 with the linear time invariant system:

$$\begin{aligned} x(k+1) &= \begin{pmatrix} 1.3 & -0.5 & 0.1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} x(k) + \begin{pmatrix} -0.4 \\ 0.6 \\ 0.5 \end{pmatrix} u(k) \\ y(k) &= \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} x(k) \end{aligned}$$

taken from `https://www.mathworks.com/help/control/examples/kalman-filter-design.html`. As seen in Figure 2, we injected a sinusoidal input (with $amplitude = bias = frequence = 1$) to this system. The actuation noise, depicted on the left, is with a unit variance.

The 'Automata Based Scheduler' block is designed to set the variance of the sensing noise dynamically to be either 0.25 or 1 at each step of the simulation. This models a sensor that has two modes of operation: a mode with high accuracy that produces noise with low variance and a mode with low accuracy that produces errors with higher variance. We assume, for the performance measurements presented below, that the CPU consumption of each mode is $\%CPU = 1.1 - errVar$, where $errVar$ is the variance of measurement error in the mode.

We ran this models with three versions of the 'Automata Based Scheduler' block. The first version, called 'High' in Table 1, is where the block acts simply as the constant 1, ignoring its inputs altogether. Similarly, the term 'Low' in the table refers to an implementation where the block is the constant 0.25. These two implementations model the constant schedules, where the sensor is operated in one mode along the whole execution. These two schedules are compared to a third implementation, called 'Automaton' in the table, where the block implements the schedule given by the automaton:



	High	Low	Automaton
%CPU	0.85	0.1	0.46
mean of $\lvert x-\hat{x}\rvert$	0.97	1.24	1.08

Table 1: Simulation results.

where the name of a state defines the variance of the estimation noise when entering it.

The results of the simulation, summarized in Table 1, show, as expected, that the CPU consumption is much lower (0.1) when using the low-quality version of the sensing algorithm and is higher (0.85) when the high-quality version of the sensing algorithm is used. The performance of the estimation in terms of the mean distance between x and $\hat{x}$ better with the high-quality version (0.97) than it is with the low-quality version (1.24). More interestingly, we can see that the experiment with the automaton that switches between the two sensor modes yields performance that is close to the performance of the high-quality sensing algorithm, using much less CPU.

4. CASE STUDY: STABILIZING A QUADROTOR IN FRONT OF A WINDOW

The case study we used to test our concept is the development of a controller that stabilizes a quadrotor [**?**] in front of a window. We implement an autonomous controller for that task and evaluated its performance.

The part of the controller that we focused on is the vision based position controller. Specifically, the main controller, that we will describe below, uses a standard low-level angular controller and a simple image processing algorithm that identifies the position of the corners of the window in the image plane[1]. Its goal is to regulate the position of the quadrotor by tilting it. Note that rotations of the quadrotor generate a more-or-less proportional accelerations in a corresponding direction. A main challenge for this controller is that the computer vision algorithm takes significant time to compute relative to the fast control loop. We can decrease computation time by lowering the resolution, but this also increase the measurement noise. We will demonstrate how adaptive scheduling of the resolution can serve for balancing resource consumption vs. control performance.

4.1 Observer Design

We first implemented an observer based on the work of Efraim at al. [**?**]. The observer gets the positions of the widow corners, enumerated clockwise starting from the top left corner noted by p_1, p_2, p_3, p_4, and extract 4 quantities based on the shape and location of the win-

[1]In the experiment, to simplify the image processing algorithm, we marked the corners of the window with led lights.

dow corners in the image plane: S_x , s_y , V_d and sz. **center of mass:** S_x and S_y represent the window "center of mass" in the image plane along the image x and y axes, respectively. S_x and S_y are normalized to the range of $[-1, 1]$. S_x is used to measure the roll angle of the window center (for stabilize the roll axis), and S_y is used to measure the altitude of the drone (for stabilizing the throttle). **Window size:** sz is the sum of the vertical edges of the window, sz is used to measure the distance of the drone from the window and then to regulate the roll angle [2]. **Vertical difference:** $V_d = \frac{y_1-y_4-(y_2-y_3)}{y_1-y_4+(y_2-y_3)}$, were y_i is the vertical position of p_i in the range of $[-1, 1]$ (0 is the center of the image) first is used to measure the angular position of the drone in relation to the window (θ), then θ and sz are used to calculate the horizontal position parallel to the window surface (x position) of the drone (see Figure 3).

REMARK 1. *We assume that process state (x) is governed by the linear* [3] *stochastic difference equation:*

$$x_k = Ax_{k-1} + Bu_k + w_k$$

And the measurement (z_k) at time k is:

$$z_k = Cx_k + v_k$$

The random variables w_k and v_k represent the process and measurement noise (respectively).

After measuring the relative position and yaw attitude, as usual, we add estimator filter to get better state estimation. In theory, and s shown in the simulation at Section **??**, we should use Kalman filter estimator for best state estimation, in our case is non-linear system and the process noise distribution is problematic to determine (battery state affects the system dynamics), this with the complexity of kalman filter lead us simplify the kalman filter principles using complementary filter, a simple estimation technique that is often used in the flight control industry. Complementary filter is actually a steady-state Kalman filter for a certain class of filtering problems [13]. We use two steps estimator that, (1) *predict* the current state evolved from the previous state using a linearized model of the system ($x_{k|\hat{k}-1}$) and then *update* the prediction with current state measurement from image, noted by z_k. The result estimation (noted by $\hat{x}_{k|k}$) is a complementary filter of the prediction ($\hat{x}_{k|k-1}$) and the measured state ($C^{-1}z_k$): $\hat{x}_{k|k} = K\hat{x}_{k|k-1} + (1 - K)C^{-1}z_k$ were K is constant approximation of the kalman filter gain (K_k).

The image processing have few different operation modes, every mode has different accuracy (shown in Section 4.3). Similarly to the gain of kalman filter, K represent the ratio between process noise and measurement noise distribution, therefore K is defined separately for each mode to achieve the best estimation in each mode.

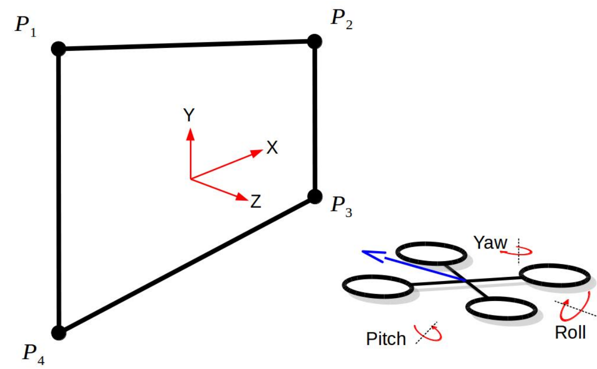


Figure 3: Description of Coordinate System and Rotation Axis.

4.2 Controller Design

In this experiment, we consider the task of *hovering in front of the window*, the controller objective is to hover parallel to the window (center of x axis) at the altitude of the window within distance of 2 meters in front of the window and face pointed to the center of the window (yaw angle). We consider this point as the origin and the coordinates are according to Fig. 3.

The controller consist of 4 independent feedback control loops, altitude, yaw angel, pitch and roll. Pitch and roll controllers composed from, low level attitude controller which his input is the required pitch or roll angel accordingly, and high level position controller which his input is the required distance or x position relative to the window accordingly, the high level controller outputs the required angle (acceleration) to the low level controller as show in Fig. 4. All the loops control regulate the position or attitude relative to the window. The inertial (angular) feedback is generated by the existing *Attitude and heading reference system* (AHRS) library of ArduPilot see Section 5, and the window related feedback come directly from the observer described in Section 4.1.

Based on the separation principle, we can use that controller regardless the measurement quality, in practice we implement basic Proportional Integral Derivative (PID [4]) controller and tuned the parameters with the highest resolution observer.

4.3 Analysis and Specification Automata

The objective of the system is to maintain stable hovering in front of the window. Hence, the performances

[2]we use fixed size window and convert sz to distance (in meter in our case) based on this window, in the general case the distance is relative to the window size

[3]we make approximation of our non-linear system by linear difference equation

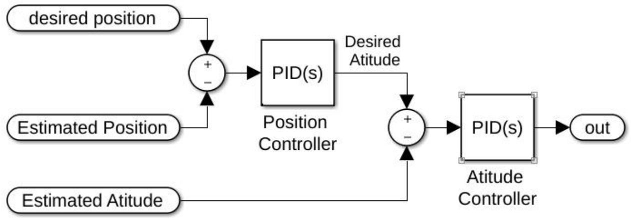


Figure 4: Atitude and Position Controller - Two level structure

of the system is measured by the amount of deviation from the center line of the window in the critical axis x, meaning, we want to minimize $|x|$ (see Figure 3). Our goal is to achieve maximum performance with minimal amount of processing time. Obviously, both goals can not be achieved together, the task is to set the best trade-off between them. The problem comes from the main consumer of computation resources, the image-based observation tasks. In this part, we use different observation modes together in order to improve the resource utilization, meaning we use more resources (CPU time) only when it is needed, and this way reduce the overall CPU usage without significantly affect the performance. In fact, we switch the camera resolution during the flight, the image resolution define the operation mode (see Section 4.1). For simplicity, in this work we examine only two modes, *High quality* with resolution of 960p and *Low quality* with resolution of 240p.

Analyzing the test results shown in Table 2, we see that High quality observation mode provide mean (x) error tolerance of 9.5 *cm* [4], and with the cost of 30% CPU usage. On the other hand, Low quality mode provide worse mean error of co *cm* (three times High quality mode), but cost only 2.1% CPU usage [5]. In this section we show how defining *adaptive scheduling specification* in order to lower the CPU usage without significant worsening of High quality performances.

First we examine a straight-forward solution based directly on the current position (the x part of $\hat{x}_{k|k}$). Using the simple guard automaton A_x presented in Fig. 5, we define dynamic observer that choose the operation mode (high or low quality) based on the x part of $\hat{x}_{k|k}$ absolute value, if the drone is closer to the center line of the window, noted by $|x| < T_x$, we consider it as "safe" state and go to L mode that use the low quality image. Similarly, "BAD" states are when the drone is far, noted by $|x| > T_x$, then we use High quality image in order to "fix" the bad state. The trash-hold T_x value can be changed to define the desire performances and processing time trade-off. Results shows good improvement in term of resource utilization (see Section 4.4).

DEFINITION 1. *Measurement post-fit residual $\tilde{y}_{k|k}$, is the difference between the expected measurement ($C \cdot \hat{x}_{k|k}$) and the measured state (z_k): $\tilde{y}_{k|k} = |\hat{x}_{k|k} - z_k|$* [6].

From the Low quality experiment graph we can see that $\tilde{y}_{k|k}$ accumulates proportional to the deviation from the center of x axis, we can use $\tilde{y}_{k|k}$ vales to predict high deviation from x. In the second experiments set we used A_{err} Automaton presented in Fig. 5, which is similar to A_x but switch states based on $\tilde{y}_{k|k}$ value. In this case the observer operate in low quality when $\tilde{y}_{k|k} < \alpha_{low}$ ("GOOD" state), and change to high quality when $\tilde{y}_{k|k} > \alpha_{high}$ ("BAD" state). The trash holds values α_{low} and α_{high} has determined first from the low and high quality graph and fine tuned by experiments. Different α_{low} and α_{high} achieve different trade-offs of performances and processing time. Note, in Section 4.4 we show how this dynamic observer do not result significant affect to the performance but reduce by factor of $\frac{1}{2}$ the processing time.

4.4 Results

TODO - graphs and tables

all statistics are taken from real indoor flights of 1-2 minutes

5. EXPERIMENT SETUP

In order to perform the experiments we used of-the-shelf quad rotor with *navio2* [?] and *Raspberry pi* [?] flight controller that runs a modified *ArduPilotMega-arducopter* [?] software. *Navio2* is extension board (shield) for the *Raspberry pi* [?] platform, it provides the nessesary interfaces such as remote control inputs and motors output, and it's equipped with 3-axes gyroscop and accelerometer MEMS sensors [?]. We used the standard raspberry pi camera.

6. CONCLUSIONS

Conclusion goes here.

Acknowledgments

Acknowledgement goes here.

7. REFERENCES

[1] R. Alur and G. Weiss. Regular specifications of resource requirements for embedded control software. In *Proceedings of the 14th IEEE Real-time and Embedded Technology and Applications*, 2008.

[4] All lengths are measured in Centimeter.

[5] The CPU usage percentage is the average usage, and is calculates as $\frac{\text{total time spend in image processing}}{\text{total flight time}}$.

[6] Measurement post-fit residual $\tilde{y}_{k|k}$ is also defined by Kalman filter

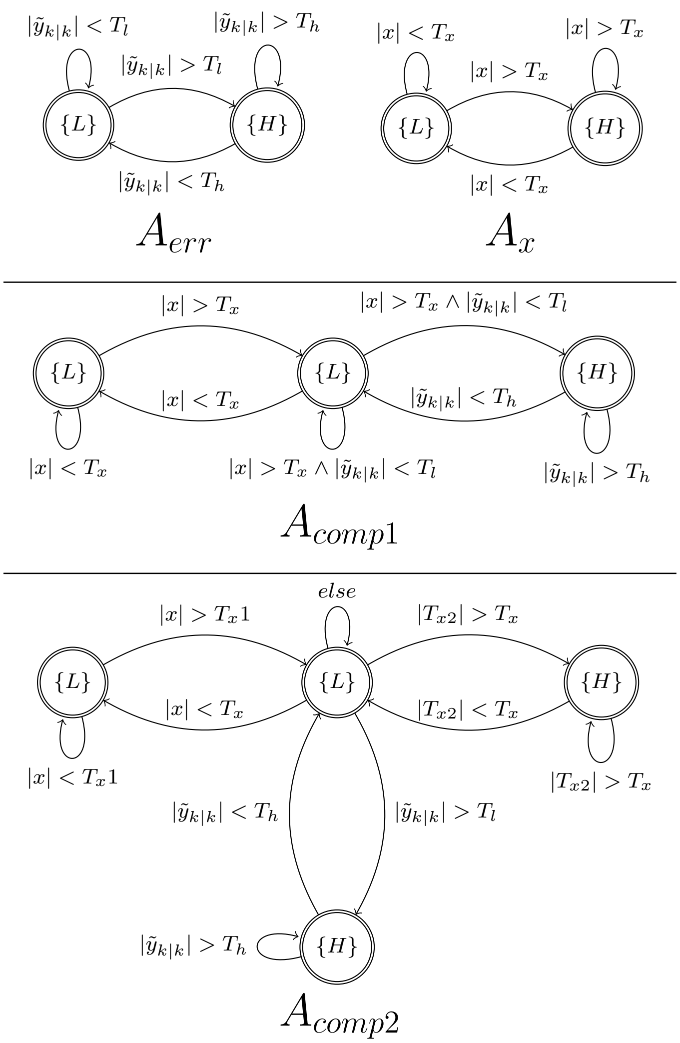


Figure 5: The main Automata we used for describing the specifications in the experiments.
A_{err} uses the *measurement post-fit residual* ($\tilde{y}_{k|k}$), if $|\tilde{y}_{k|k}|$ is too small then next iteration will use small resolution image (L), and if too large the high resolution will be used, T_l and T_h are the transition trash-holds.
A_x operate similar to A_{err} but uses the current position (the x part of $\hat{x}_{k|k}$) instead $\tilde{y}_{k|k}$, with the trash-hold T_x.
A_{comp1} and A_{comp2} are the results of composing the A_{err} and A_x ideas.

Table 2: Test Results

Schedule	% CPU	mean($|x|$) (cm)	max($|x|$) (cm)
Only High	30.9%	9.5	31.2
Only Low	2.1%	30.0	70.9
A_x ($T_x = 10$)	16.6%	10.9	42.5
A_x ($T_x = 20$)	14.0%	14.1	**32.1**
A_x ($T_x = 30$)	8.9%	17.4	**48.4**
A_{err} ($T_l = 10, T_h = 20$)	10.3%	14.9	43.2
A_{err} ($T_l = 10$, $T_h = 15$)	11.7%	11.3	36.3
A_{comp1} ($T_x = 10$, $T_l = 10$, $T_h = 15$)	8.8%	12.9	32.1
A_{comp2} ($T_{x1} = 10$, $T_{x2} = 30$, $T_l = 10$, $T_h = 15$)	10.4%	12.7	36.6

[2] R. Alur and G. Weiss. RTComposer: a framework for real-time components with scheduling interfaces. In *Proceedings of the 8th ACM international conference on Embedded software*, pages 159–168. ACM, 2008.

[3] K.-E. Arzén, A. Cervin, J. Eker, and L. Sha. An introduction to control and scheduling co-design. In *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, volume 5, pages 4865–4870. IEEE, 2000.

[4] K. J. Åström and T. Hägglund. *Advanced PID control*. ISA-The Instrumentation, Systems and Automation Society, 2006.

[5] J. Auerbach, D. F. Bacon, D. T. Iercan, C. M. Kirsch, V. T. Rajan, H. Roeck, and R. Trummer. Java takes flight: time-portable real-time programming with exotasks. In *Proceedings of the conference on Languages, Compilers, and Tools for Embedded Systems*, pages 51–62, 2007.

[6] A. Chakrabarti, L. de Alfaro, T. Henzinger, and M. Stoelinga. Resource interfaces. In *Embedded Software, 3rd International Conference*, LNCS 2855, pages 117–133, 2003.

[7] A. K. Das, R. Fierro, V. Kumar, J. P. Ostrowski, J. Spletzer, and C. J. Taylor. A vision-based formation control framework. *IEEE transactions*

on robotics and automation, 18(5):813–825, 2002.
[8] L. de Alfaro and T. Henzinger. Interface automata. In *Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering (FSE)*, pages 109–120. ACM Press, 2001.
[9] H. Efraim, S. Arogeti, A. Shapiro, and G. Weiss. Vision based output feedback control of micro aerial vehicles in indoor environments. *Journal of Intelligent & Robotic Systems*, 87(1):169–186, Jul 2017.
[10] E. Farcas, C. Farcas, W. Pree, and J. Templ. Transparent distribution of real-time components based on logical execution time. In *Proceedings of the conference on Languages, Compilers, and Tools for Embedded Systems*, pages 31–39, 2005.
[11] T. Henzinger and J. Sifakis. The embedded systems design challenge. In *FM 2006: 14th International Symposium on Formal Methods*, LNCS 4085, pages 1–15, 2006.
[12] T. A. Henzinger, B. Horowitz, and C. M. Kirsch. Giotto: a time-triggered language for embedded programming. *Proceedings of the IEEE*, 91(1):84–99, 2003.
[13] W. T. Higgins. A comparison of complementary and kalman filtering. *IEEE Transactions on Aerospace and Electronic Systems*, (3):321–325, 1975.
[14] E. Lee. What's ahead for embedded software. *IEEE Computer*, pages 18–26, September 2000.
[15] E. A. Lee. Cyber physical systems: Design challenges. In *Object oriented real-time distributed computing (isorc), 2008 11th ieee international symposium on*, pages 363–369. IEEE, 2008.
[16] A. Mok and A. Feng. Towards compositionality in real-time resource partitioning based on regularity bounds. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, pages 129–138, 2001.
[17] J. Regehr and J. Stankovic. HLS: A framework for composing soft real-time schedulers. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, pages 3–14, 2001.
[18] A. Sangiovanni-Vincetelli, L. Carloni, F. D. Bernardinis, and M. Sgori. Benefits and challenges for platform-based design. In *Proceedings of the 41th ACM Design Automation Conference*, pages 409–414, 2004.
[19] S. Sastry, J. Sztipanovits, R. Bajcsy, and H. Gill. Modeling and design of embedded software. *Proceedings of the IEEE*, 91(1), 2003.
[20] O. Shakernia, Y. Ma, T. J. Koo, and S. Sastry. Landing an unmanned air vehicle: Vision based motion estimation and nonlinear control. *Asian journal of control*, 1(3):128–145, 1999.
[21] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *Trans. on Embedded Computing Sys.*, 7(3):1–39, 2008.
[22] P. Tabuada. Event-triggered real-time scheduling of stabilizing control tasks. *IEEE Transactions on Automatic Control*, 52(9):1680–1685, 2007.
[23] L. Thiele, E. Wanderer, and N. Stoimenov. Real-time interfaces for composing real-time systems. In *Proceedings of the 6th ACM/IEEE International Conference on Embedded Software*, pages 34–43, 2006.
[24] G. Weiss and R. Alur. Automata based interfaces for control and scheduling. In *International Workshop on Hybrid Systems: Computation and Control*, pages 601–613. Springer, 2007.
[25] G. Weiss and R. Alur. Automata based interfaces for control and scheduling. In *Proceedings of the 10th workshop on Hybrid Systems: Computation and Control*, 2007.

APPENDIX

Appendix goes here.