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

We present an approach to reactive scheduling of computations in software control systems. The main motivation comes form the growing use of computer vision algorithms in real-time, as sensors. The term reactive here refers to the ability of the scheduler to adapt the schedules dynamically based on physical conditions. We propose an extension of the automata based scheduling approach with an addition of guards to transitions that allow for reactive specifications. We develop a methodology for using Klman filters to provide data that guides these automata, and demonstrate the combined approach in simulations and with a case study. The case study is a development of a software that stabilizes a quadcopter in front of a window using a vision based sensor with a time-varying resolution, i.e., computation load is controlled by taking images in reduced resolution when the state of the controlled loop allows.

1. INTRODUCTION

Cyber-physical systems (CPS) technologies of integrating software and control are at the heart of many critical applications (c.f. [13]). These technologies aim at handling issues that emerge when the interaction 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, 21, 23]. 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, 19]). 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, 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. [23]). 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. [10, 12, 18]). This has motivated researchers to develop compositional approaches and interface notions for real-time scheduling (cf. [5,6,8,14,16, 17, 20, 22]).

In this paper we present an approach, a proof-of-concept implementation, and a case study in scheduling computations in embedded control systems. The proposed design is based on the automata based schedul-

ing approach, suggested in [1, 2, 23], 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 taken into account, 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 a real-time, vision-based, control system (a quadrotor that stabilizes itself in front of a window).

The rest of the paper is organized as follows: In Section 2 we outline the general software approach and the methodologies developed in our research. In Section 3 we present the technical details of the proposed approach using a simulation. The case study is presented in Section 4 with a description of the engineering setup and the data we collected for the evaluation of the approach.

2. THE PROPOSED APPROACH

As proposed in earlier work on automata based scheduling (cf. [1,2,23]) we aim at a development process where a system is built as a composition of a set of components where each component is a software module (a set of procedures) accompanied with an automaton. Our addition here is that we allow the automata to be guarded, i.e., each 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. A second addition is that we have implemented the approach as an enhancement of the internal scheduler of the ArduPilot Mega (APM) open source unmanned vehicle autopilot software suite (`ardupilot.org`). A third contribution of the paper is a proposal of a specific way to use the automata based scheduling framework with a Kalman filter. We elaborate on each of these in the following subsections:

2.1 Guarded Automata as Interfaces for Control and Scheduling

As our goal is to allow dynamic selection of the computation load in the feedback loops based on the states of the systems, we start with a general software architecture in which each component (implementation of a feedback specific loop) is represented by a code module (in our case, a class in C++) and an automaton that specifies when to invoke its methods. The transition relation of the automaton depends, in addition to the current state, also on a real number produced by the estimator of the feedback loop (we experimented with different options for this number, as discussed below).

The motivations of using automata as described above are: (1) Automata allow for a rich specification language; (2) It is easy to construct schedules that obey the specification with negligible computational burden; (3) Automata theory gives a solid framework for composing the specifications of competing requirements for analysis and for schedule synthesis.

In this paper we focus on the first two motivations in the above list. The third is discussed in details in earlier paper on automata based scheduling (cf. [1,2,23]) and is the focus of another paper that we are preparing where we describe some analysis techniques we have developed for guarded automata.

2.2 Implementation in an Auto Pilot Software

As our main case study is in flight control, we chose the ArduPilot Mega (APM) platform for experiments. To this end, we implemented a basic automata based scheduler for this platform. The built-in task scheduling specification in APM consist of a table as shown in Figure 1. This table is easy to maintain and to use, but it is used under an assumption that there is enough CPU power to run all the tasks in the specified frequencies. APM does contain a mechanism to handle overruns, by moving tasks to the next window when there is not enough time to run them now, but the system is designed under the assumption that this only happen in rare situations.

To allow a reactive schedule that runs different modes of the software-based controllers depending on their state, we replaced the scheduling table in our version of the APM with automata that specify when to run the tasks. Note that automata allow for specifying the requirements that the table represents, using simple circular automata without guards (see, e.g., [23]). Automata, however, can model more advanced scheduling instructions with very little addition to the complexity of the scheduler, as we will demonstrate in Section 3 and in Section 4 below.

2.3 Integration With a Kalman Filter

The third layer of the approach we propose is based on the observation that a standard Kalman filter produces information that can be used to guide the automata of the components.

As we will elaborate in the description of the simulations and of the case study below, we propose to sched-

```
/*
  scheduler table - all regular tasks apart from
  the fast_loop() should be listed here, along
  with how often they should be called (in 10ms
  units) and the maximum time they are expected
  to take (in microseconds)
*/
static const AP_Scheduler::Task
     scheduler_tasks[] PROGMEM = {
        { update_GPS,             2,     900 },
        { update_nav_mode,        1,     400 },
        { medium_loop,            2,     700 },
        { update_altitude,       10,    1000 },
        { fifty_hz_loop,          2,     950 },
        { run_nav_updates,       10,     800 },
        { slow_loop,             10,     500 },
        { gcs_check_input,        2,     700 },
        { gcs_send_heartbeat,   100,     700 },
        { gcs_data_stream_send,   2,    1500 },
        { gcs_send_deferred,      2,    1200 },
        { compass_accumulate,     2,     700 },
        { barometer_accumulate,   2,     900 },
        { super_slow_loop,      100,    1100 },
        { perf_update,         1000,     500 }
};
```

Figure 1: APM scheduling specification

ule the functions that implement algorithms for sensing and for actuation based on the error variance of the estimation that a Kalman filter provides.

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 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.

Formally, we assume that each of the physical processes we control is a Linear Time Invariant (LTI) system with a known model, as depicted in Figure 2. The inaccuracies of the sensors and of the actuators are modeled as additive Gaussian noise with a known variance. As a basis for scheduling, we assume that each sensor can be scheduled to operate in one of a range of modes at each computation slot, each mode consuming a certain percentage of the CPU and giving a certain variance of the measurement noise.

The scheduling of the modes, for each of the components, is governed by the automata based scheduler as depicted in Figure 2. We propose to use a standard Kalman filter as a tool to gather the information that guides the state evolution of the automata, as follows. The filter gets as input the actuations, measurements, and the variance of the measurement noise, which we

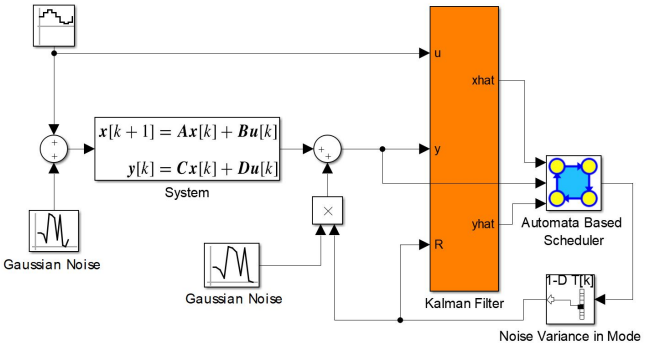


Figure 2: A Simulink model demonstrating the proposed approach.

assume is a static function (represented here as a translation table) of the mode chosen by the scheduler. The output of the Kalman filter is fed to the scheduler that uses it for advancing the states of the automata. In the Simulink model depicted in Figure 2, the covariance matrix of the disturbance, which in this case is 1×1 matrix consisting of the noise variance, is fed to the Kalman filter (as its R input) and to the block that multiplies the Gaussian noise by the variance (the product of a white noise with unit variance and a constant v yields a normally distributed noise with variance v).

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

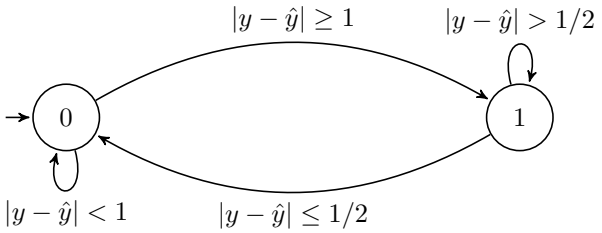
The 'Automata Based Scheduler' block is designed to set the variance of the sensing noise dynamically to be either 0.25 or 1 at each step of the simulation. This models a sensor that has two modes of operation: a mode with high accuracy, that produces normally distributed measurement errors with low variance, and a mode with low accuracy that produces normally distributed measurement errors with low 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 errors in the mode.

We ran this model with three versions of the 'Automata Based Scheduler' block. The first version, called 'High' in Table 1, is where the block acts simply as

	High	Low	Aut. Based
%CPU	85	10	46
mean of $\lvert x - \hat{x} \rvert$	0.97	1.24	1.08

Table 1: Simulation results.

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. 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 'Aut. Based' in the table, where the block implements the schedule given by the automaton:



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 $\hat{x}$, is 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. This demonstrates how automata based reactive scheduling can allow for new ways to balance performance and computational resources in software based controllers.

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 (see, e.g., `rpg.ifi.uzh.ch/aggressive_flight.html`). We implemented 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, hence we can approximate the system with a linear model. 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 increases the measurement noise. We will demonstrate how reactive scheduling of the resolution can serve for balancing resource consumption vs. control performance in this system.

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 from the image (see Section 5 for more details), and extracts the following four quantities based on the shape and location of the window corners in the image plane: **Center of mass:** S_x and S_y represent the window "center of mass" along the x and y axes of the image, respectively, normalized to the range of $[-1, 1]$. These values are used for controlling the roll angle and the altitude of the drone, respectively. **Window size:** sz is the sum of the vertical edges of the window. It is used to measure the distance of the drone from the window [2]. **Vertical difference:** $V_d = ((y_1 - y_4) - (y_2 - y_3))/((y_1 - y_4) + (y_2 - y_3))$, were y_i is the vertical position of the widow corners in the image in the range of $[-1, 1]$, enumerated clockwise starting from the top left corner. It is used to measure the angular position of the drone in relation to the window.

As common in engineering practices, we pass these numbers to a filter that gives us a more informed state estimation. In theory, as demonstrated in the simulation in Section 3, we should use Kalman filter estimator for best state estimation. As we deal with a non-linear system here, and because the process noise distribution is not known (it is significantly affected by the varying state of the battery), and because Kalman filter adds complexity in the code, we propose to use a complementary filter. A complementary filter is actually a steady-state Kalman filter for a certain class of filtering problems [11]. Specifically, in our case study, we implemented a two steps estimator that: (1) *predicts* the current state evolved from the previous state (denoted by $\hat{x}_{k|k-1}$) using a linearized model of the system, and then *updates* the prediction with current state

[1]In the experiment, to simplify the image processing algorithm, we marked the corners of the window with led lights.
[2]We used a fixed size window and converted sz to distance (in meters in our case) based on the size of the window. In the general case the units of distance are relative to the window size.

measurement from the image, denoted by y_k. The result of the estimation, denoted by $\hat{x}_{k|k}$, is a complementary filter of the prediction ($\hat{x}_{k|k-1}$) and the measured state ($C^{-1}y_k$) [3]: $\hat{x}_{k|k} = K\hat{x}_{k|k-1} + (1-K)C^{-1}y_k$ were K is constant that equals the Kalman filter gain (K_k) at a steady-state.

The image processing algorithms can run in different operation modes, each mode with different accuracy (see Section 4.3). The constant K represents the ratio between the variance of the process noise and the variance of the measurement noise. To achieve best peformace, we define a separate value of K for each mode.

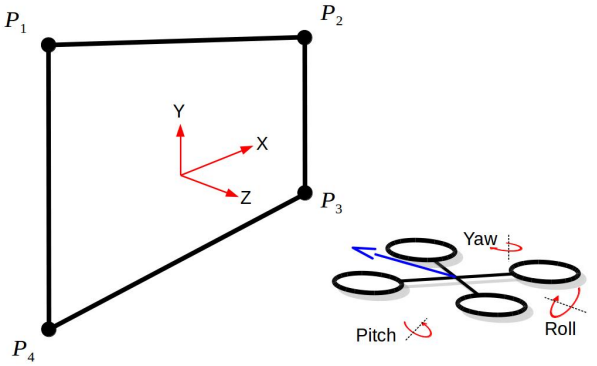


Figure 3: Description of Coordinate System and Rotation Axis.

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 z axis in Figure 3) at the altitude of the window within distance of two meters in front of the window ($y = x = 0, z = 2$) and face pointed to the center of the window (yaw angle). We consider this point as the origin and the coordinates are according to Figure 3.

The controller consists of four independent feedback loops: altitude, yaw, pitch, and roll. The pitch and the roll controllers are copies of the same attitude controller which gets as input the required pitch or roll angel, respectively. These controllers runs in parallel to a position controller that maintains the required distance (z position) and displacement (x position) relative to the window, the position level controller outputs the required angle (acceleration) to the attitude controller as show in Figure 4. All the control loops regulate the position or attitude relative to the window. The inertial (pitch and roll) feedback for the low level controller is generated by the existing *Attitude and Heading Reference system* (AHRS) library of ArduPilot (see Section 5) and the position feedback (x, y, and z position) came directly from the observer described in Section 4.1.

Based on the *principle of separation of estimation and control*, we can use that controller regardless of the measurement quality. We implemented a basic Proportional Integral Derivative (PID [4]) controller and tuned the parameters by tril and error using the highest resolution observer.

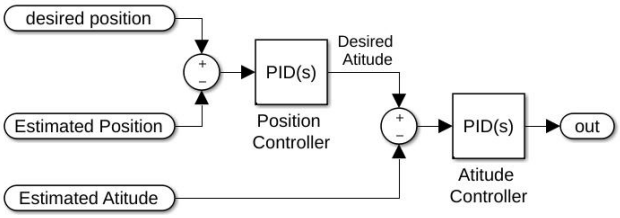


Figure 4: Attitude and Position Controller - two level, cascaded, structure.

4.3 Analysis and Specification Automata

Since the objective of the system is to maintain stable hovering in front of the window, performances is measured by the amount of deviation from the center line of the window in the critical axis x, i.e., 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. We therefor aim at achieving a good trade-off between processing resources and quality of measurement. The focus is on the main consumer of computation resources: the image processing algorithm. We aim at taking resources (high CPU time) only when needed. By this, we will reduce the average CPU usage without significantly affecting the performance. Specifically in our case study, we switch between camera resolutions dynamically during the flight. The observer modes, discussed above, correspond to image resolutions (see Section 4.1). For simplicity, we consider in this paper only two modes: *High quality* with resolution of 960p and *Low quality* with resolution of 240p.

Looking at the test results shown in Table 2, we see that the high quality observation mode provides mean error tolerance of $9.5cm$ [4] (in the x axis) with the cost of 30% CPU usage. On the other hand, the low quality mode provides mean error of $30cm$ (three times the error obtained in the high quality mode) using only 2.1% CPU usage [5].

We explain next several approaches we applied in the design of a *reactive scheduling specification* that com-

[3]The matrix C is the measurement matrix as shown in the model at Figure 2

[4]All lengths are measured in centimeter.

[5]The CPU usage percentage is the average: (total time spend in image processing)/(total flight time).

bines the advantages of both the high and the low resolution sensing modes .

We first examined a straight-forward scheduler based directly on the current position (the x axis of $\hat{x}_{k|k}$). Using the simple guarded automaton A_x presented in Figure 6, we defined a dynamic observer that chooses the mode (high or low sensing quality) based on the absolute value of the x axis of $\hat{x}_{k|k}$. If the drone is closer to the center line of the window, denoted by $|x| < T_x$, we consider it as a "safe" state and go to the mode called L that schedules the low quality image. Similarly, a state is considered "BAD" when the drone is far, denoted by $|x| > T_x$. In this case, we take high quality images in order to "fix" the state. The threshold T_x can be changed to tune the desired performances and processing time trade-off. See Section 4.4 for the data we collected by experiments to validate that this indeed achieves the required goals.

In a second experiments set, we used the A_{err} automaton presented in Figure 6. This automaton is similar to A_x with the difference that it switches states based on $\tilde{y}_{k|k}$ value defined as:

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

The motivation to use this indicator comes form looking at the Low quality experiment graph, shown in Figure 5. We can see that $\tilde{y}_{k|k}$ growth is proportional to the deviation from the center of the window (in the x axis). We can also see that the growth in $\tilde{y}_{k|k}$ precedes the growth in x. This means that we can use $\tilde{y}_{k|k}$ values to predict deviations in x.

In A_{err} the observer operates in low quality when $\tilde{y}_{k|k} < T_l$ ("GOOD" state) and switches to high quality when $\tilde{y}_{k|k} > T_h$ ("BAD" state). The thresholds T_l and T_h were initially taken from the low and the high quality graphs and fine tuned by experiments. Different T_l and T_h give different trade-offs of performances and processing time. In Section 4.4 we show that this dynamic observer gives almost the same performance as the high quality observer with a reduction of a factor of two in the processing load.

The expressiveness of automata allows for creating even more complex scheduling specification. We can, for example, combine the approaches of both A_{err} and A_x. In Figure 6 there is two examples of such combinations. A_{comp1} tries to save unnecessary CPU usage while the drone is in "safe" state, close to the center. It activates A_{err} only if the drone goes far from the center ($T_x$cm from the origin). A_{comp2} adds another constraint: if the drone goes even farther away (T_{x2}cm from the origin) it schedules only high resolution to bring it back to safety. The results show even better resource utilization.

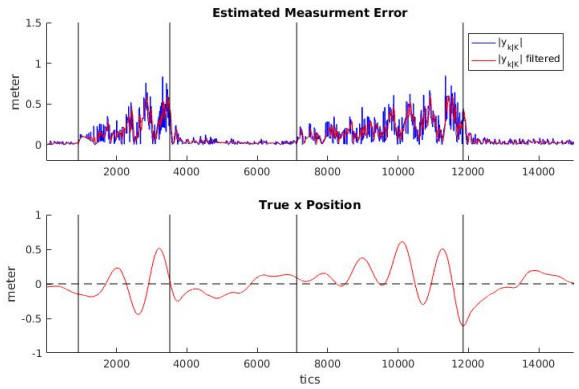


Figure 5: This figure shows in the bottom plot the position of the quadrotor during the flight time and in the top plot the $|\tilde{y}_{k|k}|$ value during the flight. During the flight we manually switch between low and high resolution for further analysis. The modes (high or low) are separated by vertical lines, were the leftmost is high resolution then low and so on.

4.4 Results

Table 2 summarizes the result of the flights experiments made using the specification automata described in Section 4.3. The first column "% CPU" represents processing time usage by the image processing task. The second column "*mean*(|x|)" is the average distance of the drone from the origin in centimeters. All the statistics shown in the table are from real indoor experiment flights. Each experiment consists of a flight (between one and two minutes) recorded using an *OptiTrack* system (see Section 5) and data from the software itself.

We see that, using reactive scheduling methods, we can adjust the trade-off between performance and CPU usage. As expected, if we use **only high** resolution mode we get the best performance but need much %CPU. On other hand, the **only low** resolution mode barely use the processor but the performance are low and unsatisfying for indoor flights.

In Table 2, automata number 3,4 and 5 are the implementation of A_x automaton as describe in Section 4.3 with the threshold values of 10, 20 and 30cm respectively. Those schedulers demonstrate that even a simple scheduling scheme, that schedules the expensive image processing task only when it is most needed, improve the performance. Scheduler number 3 achieves almost the same performance as the only high resolution scheduler but saves half of the CPU time. Schedulers automata 4 and 5 provide different trade-offs.

The A_{err} automaton (shown in Figure 6), that is based on $\tilde{y}_{k|k}$, appears to be even more effective. We tested this automaton with ($T_l = 10, T_h = 20$) and

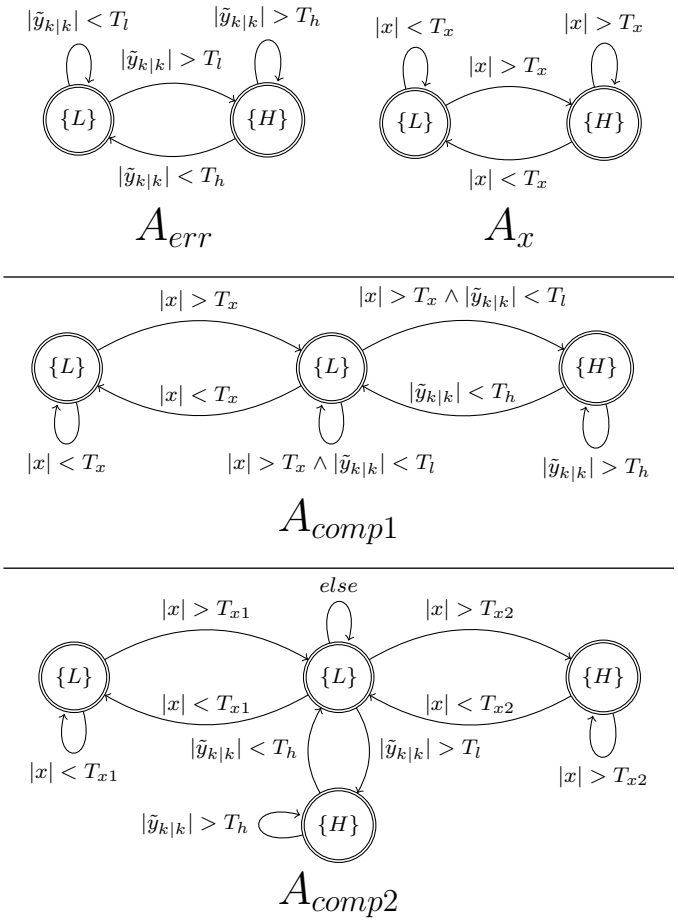


Figure 6: The main Automata we used for the requirements specifications in the experiments.
A_{err} uses the *measurement post-fit residual* ($\tilde{y}_{k|k}$), if $|\tilde{y}_{k|k}|$ is too small then next iteration will use small resolution image (L), and if too large the high resolution will be used, T_l and T_h are the transition thresholds.
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 threshold T_x.
A_{comp1} and A_{comp2} are the results of composing the A_{err} and A_x ideas.

($T_l = 10, T_h = 15$) thresholds as shown by schedulers 6 and 7 respectively in Table 2. They are using even less CPU time but still achieve fairly impressive performance. Testing the more complex schedulers can contribute even more to the performance and utilization, scheduler A_{comp1} for example achieve a bit better resource utilization. The methodology we are proposing is to tune the thresholds and to enrich the automata, as we did, until satisfactory CPU utilization and performance are achieved.

Another advantage we achieved using the dynamic schedulers, beyond improving allowing better performance to computation load balance, is reactiveness. The scheduling scheme we are proposing allows the system to adapts itself to changes in environmental conditions, demonstrated as follows. We experimented in exposing the flying drone to time-varying wind conditions. Specifically, we created artificial wind by turning on a fan while the drone was flying. The scheduler was the A_{err} automaton with the thresholds values $T_l = 10$, $T_h = 15$ (number 7 in Table 2). Figure 7 shows an experiment flight with the CPU usage along the flight where the first part of the flight, colored in blue, is indoor flight without any external wind, and during the rest of the flight we can see increasing CPU usage, colored in red, that is the scheduler respond to the wind disturbance. In this experiment the mean displacement ($mean(|x|)$ was 11.8cm which is very close to the displacement without the wind, and the average CPU usage was 13.2%, instead of 11.7% usage without the wind. This experiment further demonstrates our idea: resources are allocated only when needed.

Table 2: Test Results

	Schedule	**% CPU**	**mean($\|x\|$) (cm)**
(1)	Only High	30.9%	9.5
(2)	Only Low	2.1%	30.0
(3)	A_x ($T_x = 10$)	16.6%	10.9
(4)	A_x ($T_x = 20$)	14.0%	14.1
(5)	A_x ($T_x = 30$)	8.9%	17.4
(6)	A_{err} ($T_l = 10, T_h = 20$)	10.3%	14.9
(7)	A_{err} ($T_l = 10, T_h = 15$)	11.7%	11.3
(8)	A_{comp1} ($T_x = 10$, $T_l = 10$, $T_h = 15$)	8.8%	12.9
(9)	A_{comp2} ($T_{x1} = 10$, $T_{x2} = 30$, $T_l = 10$, $T_h = 15$)	10.4%	12.7

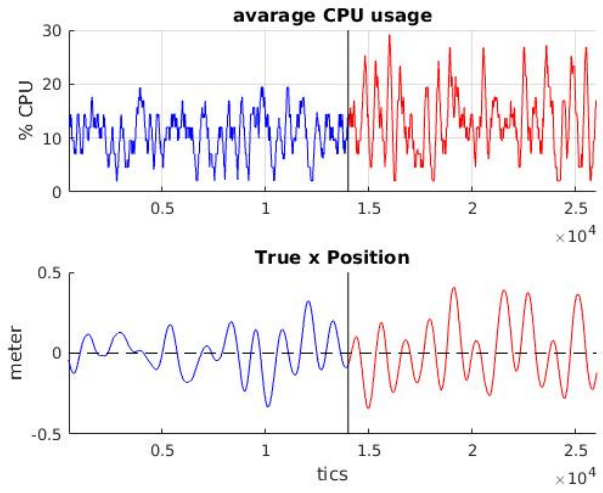


Figure 7: This figure demonstrates how the reactive scheduler adapt to the environment changes, in particular, to change of the wind. The upper part shows the average CPU usage during the flight, and the bottom is the true position at the same time. The first part of the flight (the left Blue graph) is without any external wind and in the rest of the flight (right red graph) the drone was expose to artificial wind. See that the flight exposed to wind use more CPU to overcome the wind disturbance.

5. EXPERIMENT SETUP

In order to perform the experiments we used an of-the-shelf quadrotor with a *raspberry-pi* [15] board with the *navio2* `emlid.com/navio/` shield. We implemented the controllers and the schedulers as libraries for the ArduPilot Mega (APM) autopilot software.

We used the standard raspberry-pi camera with a modified driver to allow for dynamic resolution switching. In order to simplify the image processing algorithm, we marked the four corners of the window with four illuminated balls. This allowed us to apply a simple minded corner detection algorithm: look for a blob of burned pixels and calculate the "center of mass" of all those pixels as a window corner. Higher resolution give us to a more accurate measurement of corner "center of mass".

In order to evaluate the performance, we needed to know the accurate position of the drone during the flight. For this purpose, we used the *OptiTrack* (`optitrack.com`) system consisting of an array of high-speed tracking cameras that provide low latency and high accuracy position and attitude tracking of the drone. The data from the *OptiTrack* system is considered in this paper as the ground truth.

6. CONCLUSIONS AND FUTURE WORK

Conclusion 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.

[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] 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.

[11] W. T. Higgins. A comparison of complementary and kalman filtering. *IEEE Transactions on Aerospace and Electronic Systems*, (3):321–325,

1975.

[12] E. Lee. What's ahead for embedded software. *IEEE Computer*, pages 18–26, September 2000.

[13] 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.

[14] 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.

[15] R. Pi. Raspberry pi. *Raspberry Pi*, 1:1, 2013.

[16] 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.

[17] 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.

[18] S. Sastry, J. Sztipanovits, R. Bajcsy, and H. Gill. Modeling and design of embedded software. *Proceedings of the IEEE*, 91(1), 2003.

[19] 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.

[20] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *Trans. on Embedded Computing Sys.*, 7(3):1–39, 2008.

[21] P. Tabuada. Event-triggered real-time scheduling of stabilizing control tasks. *IEEE Transactions on Automatic Control*, 52(9):1680–1685, 2007.

[22] 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.

[23] 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.