BEN - GURION UNIVERSITY OF THE NEGEV
THE FACULTY OF NATURAL SCIENCES
DEPARTMENT OF COMPUTER SCIENCE

Reactive Scheduling of Computational Resources in Control Systems

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE MASTER OF SCIENCES DEGREE

Hodai Goldman

UNDER THE SUPERVISION OF: Dr. Gera Weiss

December 2017

BEN - GURION UNIVERSITY OF THE NEGEV

THE FACULTY OF NATURAL SCIENCES
DEPARTMENT OF COMPUTER SCIENCE

Reactive Scheduling of Computational Resources in Control Systems

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE MASTER OF SCIENCES DEGREE

Hodai Goldman
UNDER THE SUPERVISION OF: Dr. Gera Weiss

Signature of student: _____ Date: _____

Signature of supervisor: _____ Date: _____

Signature of chairperson
of the committee for graduate studies: _____ Date: _____

December, 2017

Abstract

In this thesis we present an approach, a proof-of-concept tool, and experiments, for reactive scheduling of computations in software control systems. Such scheduling techniques are needed, for example, due to the growing use of computer vision algorithms in real-time, as sensors. The goal of developing reactive schedulers is to combine the dynamicity and efficiency of desktop operating systems with the predictability of real-time operating systems. The term reactive here (and in the title of the thesis) refers to the ability of the scheduler to adapt the scheduling characteristics to the physical conditions at run-time. Specifically, we propose mechanisms that allow the scheduler continuously reacts to the environmental inputs and to the internal system state.

We propose an extension of the automata based scheduling approach, as implemented, e.g., in the RTComposer [2] tool, with an addition of guards on the transitions. This addition allows for schedulers that react to data from the plants and from the controllers. Automata are reach interfaces for requirements specifications of tasks. They allows a dynamic schedule without breaking the predictability of real-time schedulers. The scheduler reactivity that we add allows for efficient schedules that alocated resources only when needed, based on the inputs. The saved processing time can be used for other tasks or for lowering the cost of the system.

The main contributions of this thesis relative to the earlier work of RTComposer [2] and GameComposer [7], are twofolds: (1) We develop methodologies for creating "enviroment"-depended component, using Kalman and Complementary filters that provide valuable data that guides these automata. (2) We validate the concepts with a real-life case study and with computer simulations.

The case study we use in this thesis is the development of a software controller that stabilizes a drone in front of a window. We show how a vision based sensor can be used with a varying resolution, i.e., computation load is controlled by taking images in reduced resolution when the state of the controlled loop allows. We developed this controller using the well known open-source autopilot software ArduPilotMega (APM) [12] with minimum changes of the original software. This allows us to draw some conclusions regarding the possibility of integrating the extended automata based approach proposed here with the state of the art control systems.

need to update Acknowledgments section

Acknowledgments

The acknowledgments.

Contents

1 Introduction **7**

2 Problem Statement **8**

3 The proposed approach **9**

3.1 Architecture 10

3.2 Guarded Automata as Interfaces for Control and Scheduling 10

3.3 Implementation in an Auto Pilot Software 11

3.4 Integration With a Kalman Filter 11

4 A demonstration of the approach in simulation **13**

5 Case Study: Stabilizing a Quad-rotor in Front of a Window **15**

5.1 Observer Design 16

5.2 Controller Design 17

5.3 Analysis and Specification Automata 19

6 Results **21**

7 Experiment Setup **24**

7.1 Hardware - the Quad-Rotor 24

7.2 The Controller Software 26

7.3 Vision Based Position Sensor 26

7.4 Reference Measurement with OptiTrack System 27

8 Scheduler Interface **28**

8.1 Correctness of Scheduler Operations 31

8.2 Some More Words on the Implementation 34

9 Related Work **34**

10 Conclusions and Future Work **36**

11 Apendix **36**

11.1 Kalman Filter and State Estimation 36

11.2 Generalize Non-Deterministic Büchi Game (GNBG) 38

List of Figures

1 A typical feedback control loop where the physical *plant* is monitored via an array of sensors (*Sensing*) which produce noisy sample of the state variables y. And after sensing, *State Estimator* aggregates the measurements from all the sensors and produce estimation of the current state $\hat{x}$. Then the *Control Law* produce output to the actuators (u) in order to close the gap between the current estimated state and the reference state (r). 9

2 The controller framework we implement, Each control loop (depicted in blue) informs the resource allocator (Scheduler) of its quality of estimation, the *scheduler* will allocate CPU time accordingly in order to maintain valid estimation quality. Specifically for the Sensing processes, for example in our case *Computer Vision* task, the guard automaton specify the required sensing quality ($var(v)$) which determines the required execution time ($time$). The Sensing alocation is noted by $\langle time, var(v) \rangle$. 10

3 APM scheduling specification 12

4 A Simulink model used to demonstrating and examine the proposed approach. 13

5 Geometric relation between α (the angle derived from V_d) l (distance from window extracted from sz) and the position of the drone among the x axis. 17

6 The coordinate system and the rotation axes of the drone, the position axis (x, y and z) are in relation to the window, "yaw" angle is the horizontal angle of the center of the window related to the front of the drone, "pitch" and "roll" are angles of the drone itself (calculated from the inertial sensors only). 18

7 Attitude and position controller consisting of a two level, cascaded, structure. 18

8 Free body diagram (force diagram) of the drone $x-y$ or $z-y$ plane forces, this diagram demonstrates the relation between x or z acceleration (a_x or a_z) and the corresponding tilt angle. α represents the tilt angle (roll or pitch) and a represents the acceleration in the corresponding direction. We assume that the altitude controller works as expected and the vertical speed is close to zero ($a_y = 0$), and we get $ma = \arctan(\alpha) \cdot mg$, that is, the acceleration is $a = \arctan(\alpha) \cdot g$ we use tis relation for the controller design (Section 5.2) and we also use this calculation as part of the system model in the Observer (see Section 5.1). 20

9 This figure shows in the bottom plot the position of the quadrotor during the flight time and in the top plot the $\left|\tilde{y}_{k|k}\right|$ value during the flight. During the flight we manually switched 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. 22

10 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 the next iteration will use small resolution image (L), and if it is too large the high resolution will be used, T_l and T_h are the transition thresholds. A_x operates similar to A_{err}, only it uses the current position (the x part of $\hat{x}_{k|k}$) instead of $\tilde{y}_{k|k}$, with the threshold T_x. A_{comp1} and A_{comp2} area composition of the ideas in A_{err} and in A_x. 23

11 This figure demonstrates how the reactive scheduler adapt to the changes in environmental conditions, 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 a flight without any external wind, and in the rest of the flight, right red graph, the drone was exposed to artificial wind (using a fan). Note that when the drone is exposed to wind it uses more CPU to compensate for the disturbance. 25

12 This figure shows the drone hovering in front of the window during one of the experiments in our lab, the window in the tests is the four light balls in front of the drone. 25

13 This figure present the graphics processing unit (GPU) configuration we used for image resizing. 27

14 This figure shows the four light balls we used to represent the window during the experiment. 28

15 Example of a game for vision based sensor component of a robot. The robot must maintain position of $y \leq 1$ where H is high accuracy and L is low accuracy position measuring tasks, and $g1$ refer to $y \leq 0.5$, $g2$ refer to $0.5 < y \leq 1$ and $g3$ refer to $1 < y$. 30

16 Example of a game for resource component that do not allows to exceed a time slot. In this example $T = \{t_1, t_2, t_3\}$, the execution time of each task t_i is half of a single time slot, therefore it is not possible to execute more than two tasks at the same time slot, but there is no limitation on the environment ($\vec{y}$). Square state is an environment player state and round state belong to the scheduler. 31

17 A strategy solution for the game presented in Figure 15. Blue states are all the states which the scheduler have a wining strategy from them, blue transitions are the needed scheduler reaction to win the game. Note: if the initial state is blue then the system is schedulable. 32

1 Introduction

Cyber-physical systems (CPS) technologies for integrating software and control are at the heart of many critical applications (see [21] for a review of such applications). 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, 31, 34]. 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 thesis is vision based control, where computer vision algorithms acquire state information to be used in a feedback loop (see,e.g., [10, 14, 28]). 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. Computer vision algorithms cannot always be invoked in full power in application, such as robot control, where many control loops, responsible for different aspects of the system, run simultaneously and share computational resources. Alternatively, we propose in this thesis 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 developing mechanisms for the integration of software and control lies in the design of efficient interfaces for integrating the engineering diciplines involved (see, e.g., [34]). 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 each 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. [17, 20, 27]). This has motivated researchers to develop compositional approaches and interface notions for real-time scheduling (see, e.g., [6, 9, 11, 23, 25, 26, 29, 32]).

In this thesis 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 scheduling approach, suggested in [1, 2, 16, 22, 34], 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 thesis relative to the earlier work in this direction is: (1) We propose an extension of the automata based scheduling framework that allows reactive scheduling where the schedule is directed by 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 drone that stabilizes itself in front of a window).

The idea of using reactive schedulers, as described in the preceding paragraph, was first introduced by Merav Bukra in her master thesis [7]. Merav's focus was on the algorithmic issues involved with the composition of components. The focus of this thesis is on the integration of the approach with control engineering and with its implementation in the context of a case-study in vision based control. Merav showed that we can provide tool support for the combination of components (even in a plug-and-play fashion). This thesis provides methods for control engineers for designing controllers that make use of reactive schedulers and a proof-of-concept implementation of such controllers in the context of a real case-study. Our goal in the future is, of course, to combine both theses into a tool suite that supports all these features together.

2 Problem Statement

In this thesis we focus on feedback control systems as shown in Figure 1. We assume a closed (feedback) control loop where the physical *plant* state x, is monitored via an array of sensors (*Sensing*) which produce measurements array y that represents noisy sample of some functions of the state variables.

Since the model includes uncertainty in observations, the data collected from the sensors is sent to an entity called *State Estimator*, that aggregates the measurements (the y array) and makes an educated estimation (called $\hat{x}$) of the current state. Then the *Control Law* entity generates the controller outputs (u) sent to the the actuators that, in turn, change the state of the system, such that (hopefully) the gap between the current estimated state and the reference desired state (*Input* r) is narrowed. We assume that the controlled process can be modeled with a general linear dynamical system discretized in the time domain. At each discrete time increment, the true state x_k is evolved from the previews state x_{k-1} according to:

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

where A is the state transition model which is applied to the previous state x_{k-1}, B is the control-input model which is applied to the control vector u_k, and w_k is the process and actuation noise which is assumed to be drawn from a zero mean normal distribution with

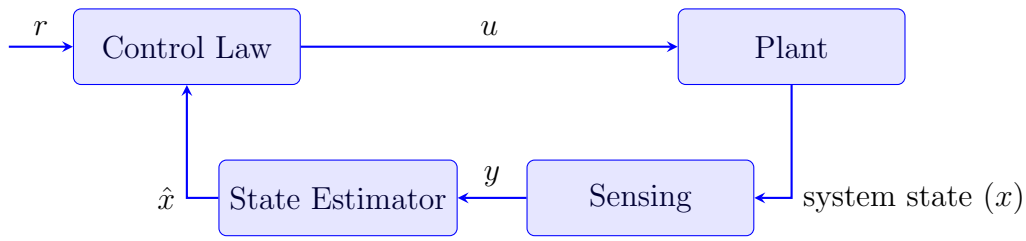


Figure 1: A typical feedback control loop where the physical *plant* is monitored via an array of sensors (*Sensing*) which produce noisy sample of the state variables y. And after sensing, *State Estimator* aggregates the measurements from all the sensors and produce estimation of the current state $\hat{x}$. Then the *Control Law* produce output to the actuators (u) in order to close the gap between the current estimated state and the reference state (r).

covariance Q_k. The observation (or measurement) value y_k at time k, of the true state x_k, is assumed to be:

$$y_k = Cx_k + v_k$$

where C is the observation model which maps the true state space into the observed space and v_k is the observation noise which is assumed to be zero mean Gaussian white noise with covariance R_k.

The current state-of-the-art, as described e.g. in [8], is that cyber-physical control systems are developed as follows: first, control engineers first design control and estimation tasks as periodic computations, then they specify the required periodic frequency for the task, and then software engineers design a scheduler that ensures the periodic frequency requirements are met. The last step is usually done using pre-computed knowledge of the expected (maximum) duration of the tasks. Note that in order to ensure performance in all conditions, for example that engine temperature never exceeds maximal safe temperature, the periodic frequency must be tuned for the worst-case condition that the system might be in.

We believe (and our experiments confirm) that scheduling control systems based on periodic tasks is limiting and inefficient for complex systems. Specifically, we show, in this thesis, how to achieve better performance and better resource utilization for control systems by using richer and more flexible requirements model for the tasks. We develop tools and methodologies such that control engineers are able to specify more accurately the requirement features of their control tasks, requirements that the scheduler can use for executing dynamic resource assignment that will, at the same time, guarantee required control performance and will be efficient in its use of computational resources.

3 The proposed approach

In this section we elaborate on the specifics of the proposed approach.

3.1 Architecture

In our methodology, we use rich component specification represented by automata. As presented above, the main advantage of this technique is the ability of the scheduler to "react" to the state of the system, therefore we need a richer interaction between the *Scheduler* and the control loops, particularly with the estimator. The architecture of control system as we believe, is illustrated in Figure 2. The architecture, that is based on modern controller architecture where the system has multiple controlling tasks, comes to support an efficient scheduling protocols in modern control systems, consisting of a processor that runs all the tasks of many independent control loops in the system. Each control loop (blue components in Figure 2) will tell the *scheduler* of its level of certainty ($P = var(\hat{x} - x)$) and the scheduler will allocate resource (processor time) based on this data, meaning the *scheduler* allocate resource dynamically based on the system current needs rather than the worst case needs, making the scheduler an active part of the control loop.

Our methodology is general and may be applicable in a wide range of applications. However, in this initial phase of the research, we focus on a specific sub-domain and in handling all technical issues in order to prove the concept.

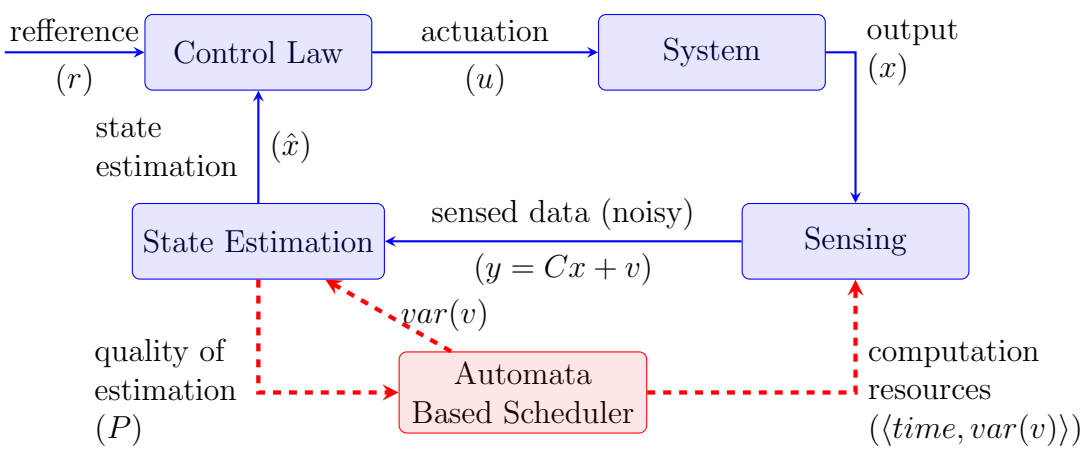


Figure 2: The controller framework we implement, Each control loop (depicted in blue) informs the resource allocator (Scheduler) of its quality of estimation, the *scheduler* will allocate CPU time accordingly in order to maintain valid estimation quality. Specifically for the Sensing processes, for example in our case *Computer Vision* task, the guard automaton specify the required sensing quality ($var(v)$) which determines the required execution time ($time$). The Sensing alocation is noted by $\langle time, var(v)\rangle$.

3.2 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 specific feedback loop or a sub-task of the controller) 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 data produced by the estimator of the feedback loop (we experimented with different options for this data, 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 thesis we focus on the first two motivations in the above list. The third is discussed in details in earlier works on automata based scheduling and is the focus of future paper that we are preparing where we describe some analysis techniques we have developed for guarded automata.

3.3 Implementation in an Auto Pilot Software

As our main case study is in flight control (see Section 5), we chose the ArduPilot Mega (APM) platform [12] 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 3. This table is easy to maintain and to adjust, 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 scheduler 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., [34]). Automata, however, can model more advanced scheduling instructions with very little addition to the complexity of the scheduler, as we will demonstrate in Section 4 and in Section 5 below.

3.4 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 schedule the functions that implement algorithms for sensing and for

```
/*
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 3: APM scheduling specification

actuation based on the estimation quality as reported the Kalman filter (denoted by P in Figure 2). The estimation quality (P) is passed to the scheduler, then the scheduler schedules higher quality sensing if P is less than the required value and lower quality sensing otherwise. Then, of course, the scheduler reports the current sensing quality to the Kalman filter (denoted by $var(v)$ in Figure 2). This way, we schedule different sensing modes in order to **adjust** the quality of estimation (P). There are two basic values that we can use for the quality of estimation (P):

The first option is to use directly the estimate covariance ($P_{k|k}$) that the Kalman filter produces (see Section 11.1). This technique can be used if we just need to adjust or control the error covariance of the estimation. This is offline scheduling, as the value of $P_{k|k}$ is not affected by the environment, i.e., it is calculates only by the noise covariance, not the measured data.

The second option is the technique we implement in this thesis (Section 5.3) is trying to estimate the overall noise of the system, actuating, sensing and process noise together. Here, instead of using $P_{k|k}$, we use the quality measure $\tilde{y}_{k|k}$ which represents the difference between the expected measurement (in case that all the noises are zero) and the actual measurement that was observed. By this definition $\tilde{y}_{k|k}$ is an approximation of the overall noises.

We call this technique 'adaptive scheduling' or 'reactive scheduling'. We show in this thesis that it is powerful in cases where the system behavior is not monotonic, e.g., when we may observe different light sources with vision sensing or changing battery states, or in cases that the system behavior is not precisely known.

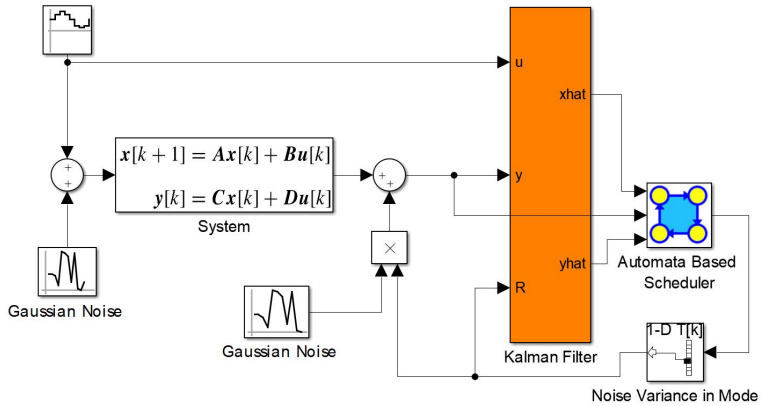


Figure 4: A Simulink model used to demonstrating and examine the proposed approach.

4 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.

As said in Section 3, 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 each of the physical processes we control are a Linear Time Invariant (LTI) system with a known model, as depicted in Figure 4. The figure shows a controlled system, in this case an open control loop, that its state is monitored via vision based sensor. 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 (and, possibly, also the actuators) 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 (or actuation) noise.

The scheduling of the modes, for each of the components, is governed by the automata based scheduler as depicted in Figure 4. 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 (u), measurements (y), and the covariance matrix of the measurement noise (R). We assume that the measurement noise 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 4, 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 Gaussian noise with variance v).

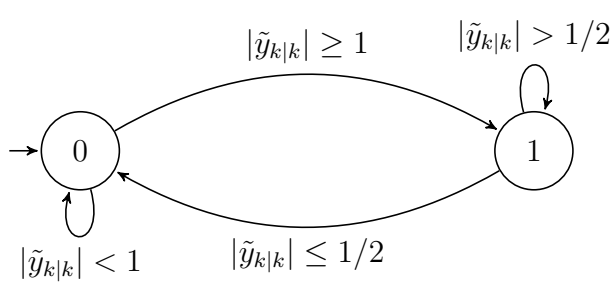
We ran the model depicted in Figure 4 with the linear time invariant system

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

taken from mathworks.com/help/control/examples/kalman-filter-design.html. As seen in Figure 4, 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 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 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 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 states of the automaton represent operation mode of the sensor, 0 for 'Low' mode and 1 for 'High' mode. $|\tilde{y}_{k|k}|$ which equals to $|y - \hat{y}|$ means how far is the last measurement from the expected measurement this value, and act as indication of the overall real noises,

	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.

process, measurement and actuation noises together, as elaborated in Section 3.4. The automaton try to formalize the expression: "use 'High' mode only after observing high distributed".

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 (the real state) and $\hat{x}$ (the Kalman filter estimated state), 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.

5 Case Study: Stabilizing a Quad-rotor 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 by a small angle 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

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

system.

5.1 Observer Design

We first implemented an observer based on the work of Efraim [13]. The observer gets the positions of the widow corners from the image (see Section 7.3 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 yaw angle and the altitude of the drone, respectively. The controller tries to drive these variables to zero so that the drone will always face the center of the window and hover at the altitude of the center of the window. **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:** $V_d = ((y_1 - y_4) - (y_2 - y_3))/((y_1 - y_4) + (y_2 - y_3))$, where 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 measure the angular position of the drone in relation to the window (parallel to the ground) and used to control the position of the drone among the x axis. Note that although V_d does not directly measure the linear x position of the drone, we can estimate the linear x position by simple geometry: as shown in Figure 5 $x = l \cdot \sin(V_d)$ where l (distance from window) can be calculated from sz. In our case because V_d is usually relatively small we approximate it by $x = l \cdot V_d$.

As common in engineering practices, we pass these numbers to a filter that gives us a state estimation. Ideally, when the system is linear as in the simulations in Section 4, we should use a Kalman filter 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 a Kalman filter adds complexity in the code, we use a complementary filter. Complementary filters can implement the steady-state (when the Kalman gains converge) behavior of Kalman filters [18]. 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 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 a constant that approximates the Kalman filter gain (K_k) at a steady-state, see Section 11.1 for more details. The constant K represents the ratio between the variance of

[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.

[3] The matrix C is the measurement matrix as shown in the model at Figure 4. For simplicity, the text is under the assumption that C is not not singular. If C is singular, one can use the observability matrix in a similar fashion, assuming that the system is observable.

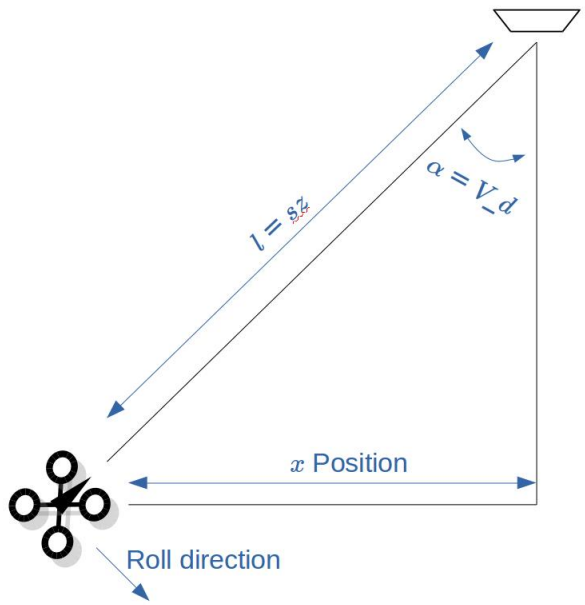


Figure 5: Geometric relation between α (the angle derived from V_d) l (distance from window extracted from sz) and the position of the drone among the x axis.

the process noise and the variance of the measurement noise. To achieve best performance, we define a different value of K for each mode corresponding with the image resolution and the noise variance.

need to add section on the system linearized model? maybe the Figure 8 is enough

5.2 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 6) at the altitude of the window within distance of two meters in front of the window ($y = x = 0, z = 2$) and face to the center of the window (yaw angle). We consider this point as the desired position and use the coordinate system depicted in Figure 6.

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 7. Altitude, yaw, x, and z position are regulated relative to the window. The inertial pitch and roll feedback for the low level controllers is related to the drone body and is generated by the existing *Attitude and Heading Reference system* (AHRS) library of ArduPilot (see Section 7) and the position feedback (x, y, and z position) comes from the observer described in Section 5.1.

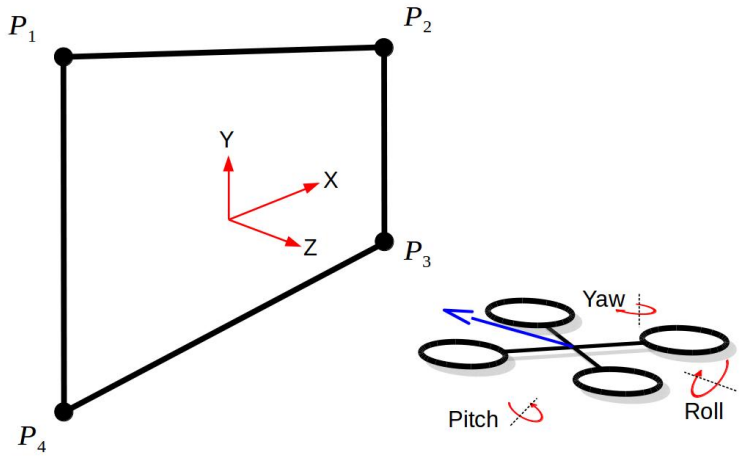


Figure 6: The coordinate system and the rotation axes of the drone, the position axis (x, y and z) are in relation to the window, "yaw" angle is the horizontal angle of the center of the window related to the front of the drone, "pitch" and "roll" are angles of the drone itself (calculated from the inertial sensors only).

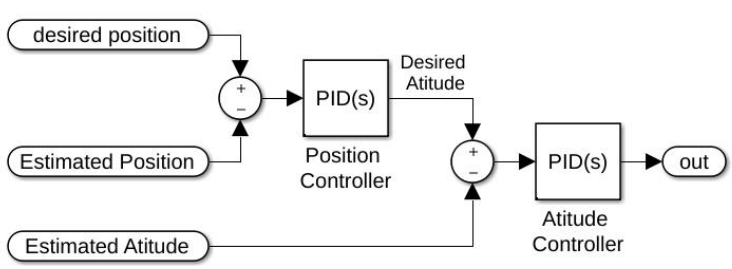


Figure 7: Attitude and position controller consisting of a two level, cascaded, structure.

The position controller is as follows: The altitude (y position) is obtained directly from the S_y value explained in Section 5.1 and it is regulated by changing the overall "throttle" of the rotors, this approach works well assuming that the drone pitch and roll angles are close to zero. In this case, S_y is almost proportional to the real altitude of the drone and the trust vector is pointed up. The z position is assumed to be the distance from the center of the window, which we assume is proportional to the window size (sz). It is regulated by changing the pitch angle. Assuming the drone is always facing the center of the window (yaw angle is zero) the pitch angle produces acceleration in the pitch direction, i.e., it can be use to regulate z. The x position (displacement) is regulated by changing the roll angle. Similar to z position, assuming the drone is always facing the center of the window, the roll angle produces acceleration that increase or decrease the x position depending on the roll angle direction.

As demonstrated in Figure 8, the acceleration in the x direction is $a_x = g \arctan(\alpha_{roll})$ and the acceleration in the z direction is equal to $a_z = g \arctan(\alpha_{pitch})$. In practice, in this work we usually assume a linearized proportional acceleration model, i.e., $a_x = g \cdot \alpha_{roll}$ and $a_z = g \cdot \alpha_{pitch}$.

Note that if x is not zero the drone is not parallel to the window and the roll direction (that used to regulate x position) also affects the z position. Similarly, if the drone is not parallel to the window and the pitch direction (that used to regulate z position) also affects the x position. Although in theory x controller disturb the z controller and vice versa, in practice even if the drone displacement is relatively large, when all the controllers are executed together (and are tuner correctly) they constantly fix those side-afects and because the x position is generally decreased during this process we eventually converge to a stable state were the drone is near-parallel to the window.

Based on the *principle of separation of estimation and control* [4], we can use that controller regardless of the measurement quality. Therefore, we implemented a basic Proportional Integral Derivative (PID [5]) controller and tuned the parameters by trial and error using the highest resolution observer.

5.3 Analysis and Specification Automata

The objective of the system is to maintain stable hovering in front of the window, for analysis, performances is measured by the amount of deviation from the center line of the window in the x axis, i.e., we want to minimize $|x|$ (see Figure 6). Our goal is to achieve maximum performance with minimal amount of processing time. Obviously, both goals cannot be achieved together. We therefor aim at achieving a good and flexible 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

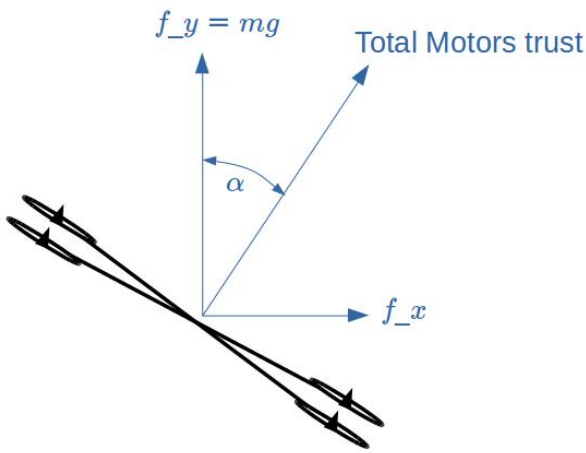


Figure 8: Free body diagram (force diagram) of the drone $x-y$ or $z-y$ plane forces, this diagram demonstrates the relation between x or z acceleration (a_x or a_z) and the corresponding tilt angle. α represents the tilt angle (roll or pitch) and a represents the acceleration in the corresponding direction. We assume that the altitude controller works as expected and the vertical speed is close to zero ($a_y = 0$), and we get $ma = \arctan(\alpha) \cdot mg$, that is, the acceleration is $a = \arctan(\alpha) \cdot g$ we use tis relation for the controller design (Section 5.2) and we also use this calculation as part of the system model in the Observer (see Section 5.1).

switch between camera resolutions dynamically during the flight. The observer modes, discussed above, correspond to image resolutions (see Section 5.1). For simplicity, we consider in this thesis 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 $9.5cm$ (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 [4].

We explain next several approaches we applied in the design of a *reactive scheduling specification* that combines both, high and the low resolution sensing modes in an sufficient way.

We first examined a straight-forward scheduler based directly on the current position (the x axis value). Using the simple guarded automaton A_x presented in Figure 10, we defined a dynamic observer that chooses the mode (high or low sensing quality) based on the current absolute value of the x axis (x coordinate 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

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

performances and processing time trade-off. See Section 6 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 10. This automaton is similar to A_x with the difference that it switches states based on the *measurement post-fit residual* $\tilde{y}_{k|k}$ value, as presented in Section 3.4, and defined as:

$$\tilde{y}_{k|k} = C\hat{x}_{k|k} - y_k$$

Where $C\hat{x}_{k|k}$ is the expected measurement and y_k is the real measurement at time k.

The motivation to use this indicator comes form looking at the low quality experiment graph, shown in Figure 9. We can see that $\left|\tilde{y}_{k|k}\right|$ growth is proportional to the deviation from the center of the window (in the x axis). This means that we can use $\left|\tilde{y}_{k|k}\right|$ values to predict deviations in x.

In A_{err} the observer operates in low quality when $\left|\tilde{y}_{k|k}\right| < T_l$ ("GOOD" state) and switches to high quality when $\left|\tilde{y}_{k|k}\right| > T_h$ ("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 6 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 10 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.

6 Results

Table 2 summarizes the result of the flight experiments made using the specification automata described in Section 5.3. 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 with the *OptiTrack* system (see Section 7.4) and with 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 the other hand, the **only low** resolution

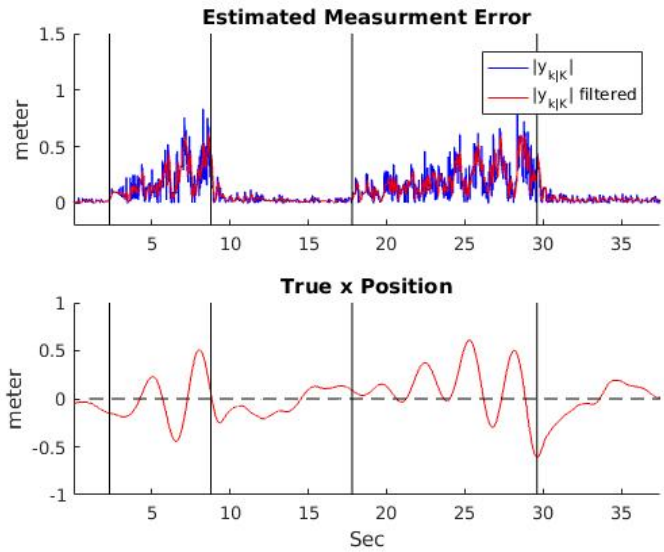


Figure 9: This figure shows in the bottom plot the position of the quadrotor during the flight time and in the top plot the $\left|\tilde{y}_{k|k}\right|$ value during the flight. During the flight we manually switched between low and high resolution for further analysis. The modes (high or low) are separated by vertical lines, were the leftmost is high resolution then low and so on.

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 5.3 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 the state is critical, improves the performance. Scheduler number 3 achieves almost the same performance as the only high resolution scheduler but saves half of the CPU time. Scheduler automata 4 and 5 provide different trade-offs.

The A_{err} automaton (shown in Figure 10), which bases its transitions on $\tilde{y}_{k|k}$, appears to be even more effective. We tested this automaton with $(T_l = 10, T_h = 20)$ and $(T_l = 10, T_h = 15)$ 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. The more complex schedulers contribute even more to the performance and utilization. Scheduler A_{comp1} for example achieves a little 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 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 $T_l = 10$

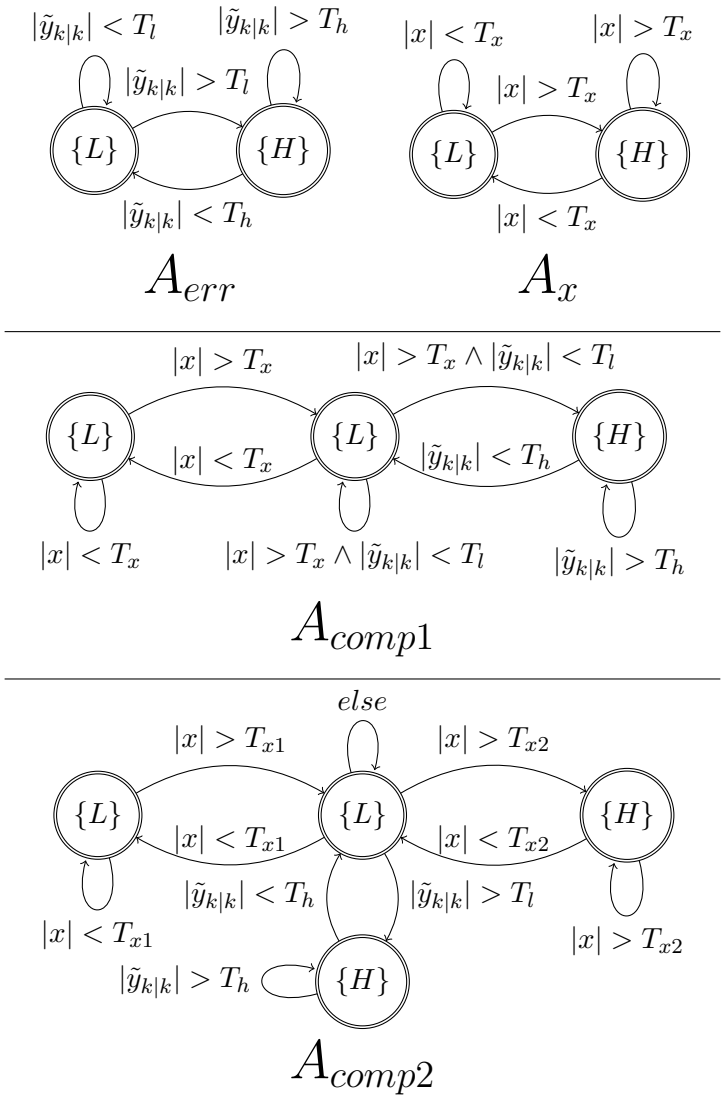


Figure 10: The main Automata we used for the requirements specifications in the experiments. A_{err} uses the *measurement post-fit residual* ($\tilde{y}_{k|k}$), if $|\tilde{y}_{k|k}|$ is too small then the next iteration will use small resolution image (L), and if it is too large the high resolution will be used, T_l and T_h are the transition thresholds. A_x operates similar to A_{err}, only it uses the current position (the x part of $\hat{x}_{k|k}$) instead of $\tilde{y}_{k|k}$, with the threshold T_x. A_{comp1} and A_{comp2} area composition of the ideas in A_{err} and in A_x.

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

and $T_h = 15$ (number 7 in Table 2). Figure 11 shows data collected in one of the flights with the CPU usage and the displacement of the drone along the flight. The first part of the flight, colored in blue, is indoor flight without any external wind. The second part, colored in red, is a flight with the disturbance caused by the wind of the fan. We can clearly see increasing CPU usage when there is wind. We can also see that the wind does not affect the displacement, as the increased accuracy of the sensor compensates for it. The mean displacement ($mean(|x|)$ in this experiment was 11.8cm which is very close to the displacement without the wind, and the average CPU usage increased to 13.2% from 11.7%. This experiment further demonstrates our idea: resources should be allocated only when needed.

7 Experiment Setup

This Section present the technical details related to the implementation of the experiment in Section 5.

7.1 Hardware - the Quad-Rotor

In order to perform the experiments presented in Section 5, we used an of-the-shelf quadrotor, named *Spedix-S250*, shown in Figure 12 with improved flight controller: *Raspberry-pi*, a single-board computer [24], with the *navio2* [15] shield. *Navio2* is extension board (shield) for the *Raspberry-pi* platform. It provides the necessary interfaces such as re-

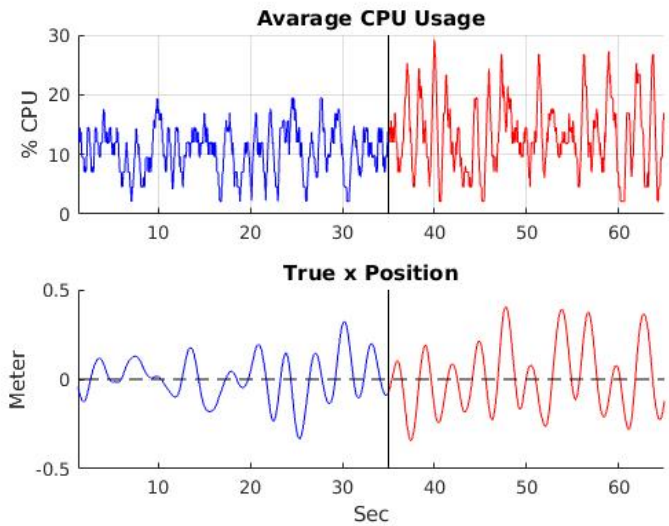


Figure 11: This figure demonstrates how the reactive scheduler adapt to the changes in environmental conditions, in particular, to change of the wind. The upper part shows the average CPU usage during the flight, and the bottom is the true position at the same time. The first part of the flight, the left Blue graph, is a flight without any external wind, and in the rest of the flight, right red graph, the drone was exposed to artificial wind (using a fan). Note that when the drone is exposed to wind it uses more CPU to compensate for the disturbance.

Figure 12: This figure shows the drone hovering in front of the window during one of the experiments in our lab, the window in the tests is the four light balls in front of the drone.

mote control inputs and motors output, and it is equipped with on-board Micro Electro-Mechanical System (MEMS) sensors, such as rate gyros and accelerometers. The software we used to controll the drone is ArduPilot Mega (APM) autopilot software [12] with few modifications.

7.2 The Controller Software

We implemented the controllers and the schedulers as libraries for the ArduPilot Mega (APM) autopilot software [12]. APM architecture is very modular, the component are well separated but has well defined interface between them. This architecture allows us to easy replace, add, or modify the necessary parts. The main changes in software that we made are, The addition of position sensing module based on image processing, and addition of our own flight-mode for handling the autonomous flight mission (in APM, flight-mode is the module responsible to apply the control law). We also add position sensing module just for experiment references, this is module use OptiTrack in order to provide very precise position measurement.

7.3 Vision Based Position Sensor

As explained in Section 5 we use vision-based sensor. This sensor first need to identify the window in front of the drone, and then produce position measurements relative to the window. In order to identify the window we used a standard raspberry-pi camera module (V1) (see `www.raspberrypi.org/documentation/hardware/camera/`), mounted in the front of the drone. This camera can take pictures in different resolutions and at different frame rate. The controller software implemented in C++, and we initially use the most common used driver for C++ called RaspiCam (see `www.uco.es/investiga/grupos/ava/node/40`. RaspiCam have two main difficulties for us:

1. The raspberry-pi camera take a picture in a constant pre-define frame rate [5], this mean that if we want to grab an image the driver (RaspiCam) will wait till the next frame is taking. This approach is not suitable for real-time system.

2. As describe in Section 5 we need to switch image resolution many times during the flight, raspberry-pi camera resolution is defined only at initialization, therefore changing resolution of the camera results in significant delay, we need to initialize the camera again and wait till the camera is ready at the new resolution.

We solve the first difficulty by modifying RaspiCam driver. We add to RaspiCam the ability to notify the client when the next frame is ready, this way whenever a new frame is taken the callback function is called. RaspiCam is open-source project, and we contribute

[5] The constant pre-define frame rate is only when using "vid" mod which allows fast frame rate.

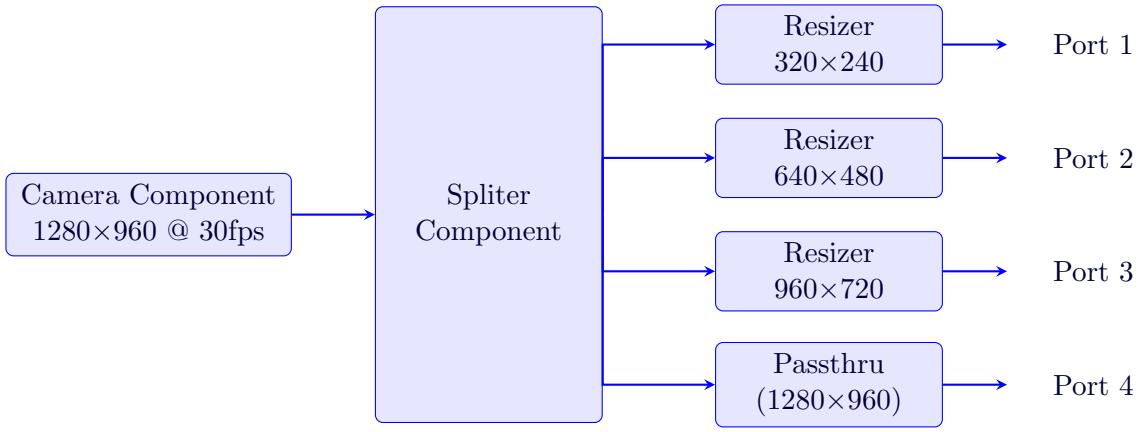


Figure 13: This figure present the graphics processing unit (GPU) configuration we used for image resizing.

to the project by adding this modification to the official RaspiCam code that is maintained in `github.com/cedricve/raspicam`.

In order to overcome the second difficulty we try to take pictures at the high resolution and then resize by software (using OpenCV) to the desire resolution, this solution still add undesirable delay for each frame that is taking. At the end, we create a new camera driver based on the work of Chris Cummings at `robotblogging.blogspot.co.il/2013/10/an-efficient-and-simple-c-api-for.html`. In this driver we use the internal GPU hardware of the Raspberry-py in order to resize the frame by hardware. Using the ARM side libraries, *userland* (`https://github.com/raspberrypi/userland`). We construct network of four hardware resizers connected to four video ports, each resizer configure to a different resolution, and all are connected to the camera port (see Figure 13). Hardware operations takes relatively zero time to perform, and now each time we want to take a picture we can choose the image resolution by retrieving the image from the corresponded video port.

In the case-study, shown in Section 5, we always use the camera at 30 frames per second. In order to simplify the image processing algorithm, we marked the four corners of the window with four illuminated balls (see Figure 14). This allows us to apply a simple algorithm for corner recognition: look for a blob of burned pixels in the image, then calculate the "center of mass" of all those pixels as a window corner. Higher resolution give a more accurate measurement of corner "center of mass", but require more processing time.

Then we just perform the calculations presented in Section 5.1 in order to extract the position measurements from the window corners.

7.4 Reference Measurement with OptiTrack System

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

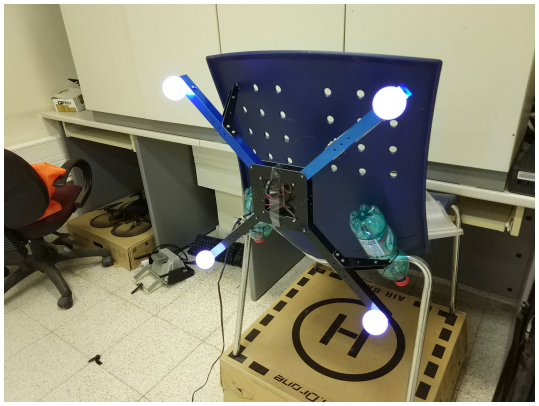


Figure 14: This figure shows the four light balls we used to represent the window during the experiment.

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 *OptiTrack* system is sent over the client-server data streaming protocol *NatNet*.

For best synchronization, we create a NatNet client library inside the APM code. This library allows the controller check anytime for the current state and log that information in parallel to the other controller parameters. We also use the OptiTrack information in early stages of the research as the feedback of the controller before we use the Raspberry-pi camera.

8 Scheduler Interface

In Section 3 we explained how automata allow for rich component specification for reactive scheduling. While the automata used in the experiments where crafted by hand without automatic composition, we outline in this section how to formally create and compose automata using Büchi game theory. The formalisms described below allow for automating the composition and the schedule generation using tools such as GOAL [33].

In our framework, similarly to most traditional real time systems, the resource is allocated in discrete *time slots* of fixed duration in the style of time-triggered architecture [2]. A typical software control system consists of multiple tasks that need to be taken care of simultaneously in order to control the system. Each task is implemented by one or more subroutines (code functions). We suggest to design control system in the *component* level, where each component corresponds to a specific task in the system consisting of all the subroutines of the task along with a Büchi game specify how the task subroutines should be executed. This is formalized in the following definition:

Definition 1. *A component is a pair $\langle T, G\rangle$ where T is a set of subroutines (representing, e.g., procedures of a class in C++) and $G = \langle A, \langle P_{schd}, P_{env}\rangle\rangle$ is a generalized Büchi game (GNBG), over a generalized Büchi automaton $A = \langle Q, \Sigma, \Delta, q_0, \mathcal{F}\rangle$ (see Section 11.2),*

such that:

1. *The alphabet is* $\Sigma = 2^T \cup \mathbb{R}^n$ *where* n *is the number of scheduler feedback variables in the system (real numbers that the scheduler can base its decision on).*

2. *Alternating turns with different parts of the alphabet: The transitions of* A *are of the form:* $s_{schd} \xrightarrow{\sigma_T} s_{env}$ *where* $s_{schd} \in P_{schd}, s_{env} \in P_{env}$*, and* $\sigma_T \subseteq T$ *(called "scheduler" transitions), or of the form* $s_{env} \xrightarrow{\sigma_e} s_{schd}$*, where* $s_{schd} \in P_{schd}, s_{env} \in P_{env}$*, and* $\sigma_e \in \mathbb{R}^n$ *(called "environment" transitions).*

3. *The environment plays first:* $q_0 \in P_{env}$*.*

The Büchi game, $G = \langle A, \langle P_{schd}, P_{env} \rangle \rangle$, of a component is a scheduler-against-environment game. It represents the interaction of the controller with the environment in the real world. The game goes as follows: before each time slot the environment player (P_{env}) plays first and takes the transition $s_{env} \xrightarrow{\sigma_{\vec{y}}} s_{schd}$ signaling that the observation values at the beginning of this slot is $\vec{y} \in \mathbb{R}^n$. Then, the scheduler needs to choose a transition $s_{schd} \xrightarrow{\sigma_{T'}} s'_{env}$ and to execute all the tasks in $\sigma_{T'}$. The environment transition represents the evolution of the system state caused by the actuators and the environment of the previous time step, and then the scheduler reacts to the state evolution by the schedule transition. This simultaneous walk over the game automata A creates The scheduler goal is to pass through accepting states infinitely many times during the execution. In other words, if the word $\omega = \vec{y}_1, T_1, \vec{y}_2, \ldots$ is created by this simultaneous walk over the game automaton A is an accepting world ($\omega \in \mathcal{L}_\omega(A)$) . Note, the scheduler feedback variable (or observations) is a vector $\vec{y} \in \mathbb{R}^n$ that may includes state variables and internal variable of the components.

???

WTF

This interface is demonstrated in Figure 15 in the context of a vision based sensor component of a robot. The component we focus on here estimates the robot position using a camera. It has two operation modes: high (H) or low (L) accuracy. The variable y is the estimated position. In this example the component requirement is to ensure that y never exceeds one (there is a cliff that starts at $y = 1$).

As we said a typical system composed of multiple components that need to be executed simultaneously by the same processor. Each component have its own game (its own requirements), therefore we will construct a composed component that correspond to all the system components together, then the scheduler have only one component, with only one game to win for scheduling all the components. Let's define composition of the components, that preservative the logic and correction of its composed components.

Definition 2. *The composition of the components* $C_1 = \langle T_1, \langle A_1, \langle P^1_{schd}, P^1_{env} \rangle \rangle \rangle$ *and* $C_2 = \langle T_2, \langle A_2, \langle P^2_{schd}, P^2_{env} \rangle \rangle \rangle$ *is the component* $C_c = C_1 \times C_2 = \langle T_1 \cup T_2, \langle A_c, \langle P^1_{schd} \times P^2_{schd}, P^1_{env} \times P^2_{env} \rangle \rangle \rangle$ *where* $A_c = \langle Q_1 \times Q_2, (T_1 \cup T_2) \times \mathbb{R}^n, \Delta_c, Q^1_0 \times Q^2_0, \mathcal{F}_1 \cup \mathcal{F}_2 \rangle$ *is a product of* C_1 *and* C_2 *and*

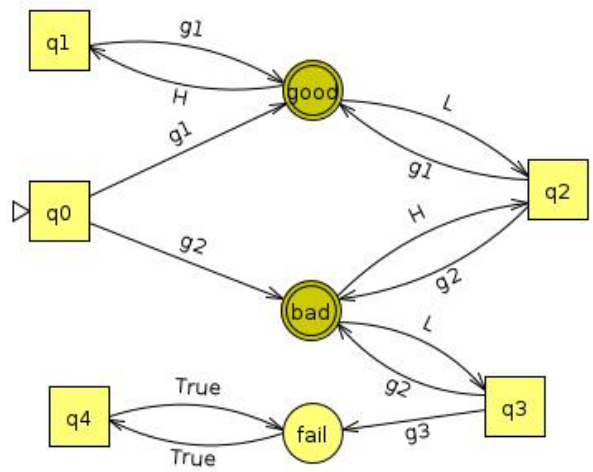


Figure 15: Example of a game for vision based sensor component of a robot. The robot must maintain position of $y \leq 1$ where H is high accuracy and L is low accuracy position measuring tasks, and $g1$ refer to $y \leq 0.5$, $g2$ refer to $0.5 < y \leq 1$ and $g3$ refer to $1 < y$.

1. *Environment transitions:* $\forall y \in \mathbb{R}^n, s_1, s_1' \in Q_1, s_2, s_2' \in Q_2$: $\langle s_1 s_2 \rangle \xrightarrow{y} \langle s_1' s_2' \rangle \in \Delta_c$ *if and only if* $s_1 \xrightarrow{y} s_1' \in \Delta_1$ *and* $s_2 \xrightarrow{y} s_2' \in \Delta_2$.

2. *Scheduler transitions:* $\forall T_1' \subseteq T_1, T_2' \subseteq T_2$, $s_1, s_1' \in Q_1, s_2, s_2' \in Q_2$: $\langle s_1 s_2 \rangle \xrightarrow{T_1' \cup T_2'} \langle s_1' s_2' \rangle \in \Delta_c$ *if and only if* $s_1 \xrightarrow{T_1'} s_1' \in \Delta_1$ *and* $s_2 \xrightarrow{T_2'} s_2' \in \Delta_2$.

Note: components composition is defined for compose two components, but we can compose all the system component, $C_1, ..., C_m$, one by one: $C_c = (\cdots((C_1 \times C_2) \times C_3) \times \cdots \times C_m)$.

In addition to the component requirements (requirements from control objective) we also have resources requirements, we have limited resources (limited CPU abilities) that need to be taking in account. Resources requirements are reflected as limited time slot capacity, each subroutine (procedure) take some time to perform but the slot is limited in time. For example, two subroutines t_1 and t_2 that require 1.5 millisecond each to execute, cannot be executed in two millisecond time slot. We will enforce the slot size with a special and simple component $C_{resource}$ that can be generated automatically as follows: assuming we know the maximum duration of each subroutine of the components in the system (it can be result of analyzing, testing, or manually pre-defined) we construct component with the tasks set T that is union of all the subroutines in the system, and the game ($G_{resource}$) is as demonstrate in Figure 16 with one state for environment player (square state) and one for the scheduler (circle state), the environment have no limitations but the scheduler can execute only set of tasks that not exceeds the time slot.

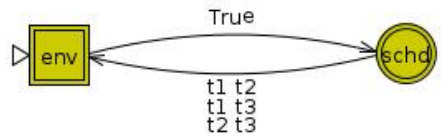


Figure 16: Example of a game for resource component that do not allows to exceed a time slot. In this example $T = \{t_1, t_2, t_3\}$, the execution time of each task t_i is half of a single time slot, therefore it is not possible to execute more than two tasks at the same time slot, but there is no limitation on the environment ($\vec{y}$). Square state is an environment player state and round state belong to the scheduler.

Finally, the resource component is composed with all the rest components of the system resoult in the schedule component $C_s = C_{resource} \times C_c$.

The component C_s aggregates all the system constrains, in order to successfully schedule the system, we need to find a "wining" strategy $A_{strategy} \in NBA$ for the scheduler. $A_{strategy}$ is in fact, a reduction of the game automaton $A_s = \langle Q_s, \Sigma, \Delta_s, Q_0, \mathcal{F}_s \rangle$ of C_s. It is contained all the environment transitions in A_s, but only the scheduler transitions that allays can lead to a winning of the scheduler player, that is, runs that we visit infinitely times a state of every color $F' \in \mathcal{F}_s$. Example of such strategy is shown in Figure 17, the blue partial automaton is the scheduler wining strategy for the game presented in Figure 15 discussed before. We call this winning strategy a guarded automaton .

we can add here reference to the simpler guarded automata presented in the previews sections

The complete operation of the scheduler is described by the pseudo code 1. Every time the system is executed, the scheduler construct the strategy automaton A_s, and then just walk through the strategy as describe by the pseudo code 2.

Algorithm 1 Scheduler start up procedure

```
procedure ScheduleStartUp(C_1, ..., c_m)    ▷ schedule the components C_1, ..., c_m
    C_c ← Compose(C_1, ..., c_m)
    C_resource ← CreateResourceComponent(C_1, ..., c_m)
    C_s ← Compose(C_c, C_resource)
    A_s ← FindWiningStrategy(C_s)
    if A_s is not empty then
        Schedule(A_s)                            ▷ start to run the system
    end if
end procedure
```

8.1 Correctness of Scheduler Operations

Definition 3. *A component C_i is* ***complete*** *if for each $t_1, \ldots, t_m \subseteq T_i$, $\vec{y}_1, \ldots, \vec{y}_m \in \mathbb{R}^n$ and $q_e \in P_{env}$ such that there is a path from q_0 to q_e with the word $w = \vec{y}_1, t_1, \vec{y}_2, t_2, \ldots, \vec{y}_m, t_m$*

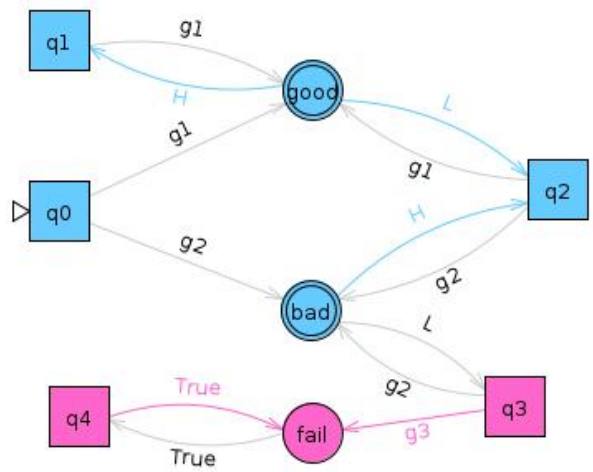


Figure 17: A strategy solution for the game presented in Figure 15. Blue states are all the states which the scheduler have a wining strategy from them, blue transitions are the needed scheduler reaction to win the game. Note: if the initial state is blue then the system is schedulable.

we have that for each $\vec{y}_{m+1}$ that can be observed after any run of the system that corresponds to this word (i.e., a run where after each observation $\vec{y}_j$ we scheduled t_j) there is a transition from q_e labeled with $\vec{y}_{m+1}$.

Definition 4. *A component $C_i = \langle T_i, \langle A_i, \langle P^i_{schd}, P^i_{env} \rangle \rangle \rangle$ is* ***well-formed*** *if it is complete, and in any execution whose observations and task schedules constitute a word $\omega \in \mathcal{L}(A_i)$, the component achieves its control objectives.*

Claim 1. *If $C_1 = \langle T_1, \langle A_1, \langle P^1_{schd}, P^1_{env} \rangle \rangle \rangle$ and $C_2 = \langle T_2, \langle A_2, \langle P^2_{schd}, P^2_{env} \rangle \rangle \rangle$ are a well-formed components then their composition $C_c = C_1 \times C_2$ is also a well-formed component.*

Proof. 1. C_c is a complete component: Assume that after we get to state $\langle q_1, q_2 \rangle \in P^1_{env} \times P^2_{env}$ we may observe $\vec{y}$, then because C_1 is complete there is a transition $q_1 \xrightarrow{\vec{y}} q'_1$ and same for $q_2 \xrightarrow{\vec{y}} q'_2$ in C_2, therefore by the composition definition A_c contains the transition $\langle q_1, q_2 \rangle \xrightarrow{\vec{y}} \langle q'_1, q'_2 \rangle$

2. C_c is a well-formed component: by the definition of composition the word $\omega = \vec{y}_1, t_1, \vec{y}_2, t_2, ... \in \mathcal{L}(A_c)$ if and only if there exist words $\omega_1 = \vec{y}_1, t^1_1, \vec{y}_2, t_2, ... \in \mathcal{L}(A_1)$ and $\omega_2 = \vec{y}_1, t^2_1, \vec{y}_2, t^2_2, ... \in \mathcal{L}(A_2)$ such that $t_i = t^1_i \cup t^2_i$. Therefore, for each $\omega = \vec{y}_1, t_1, \vec{y}_2, t_2, ... \in \mathcal{L}(A_c)$ a system execution that before each time slot j observe $\vec{y}_j$ and then execute the tasks t_i in that slot, is in particular execute t^1_i, and because $\omega_1 = \vec{y}_1, t^1_1, \vec{y}_2, t_2, ... \in \mathcal{L}(A_1)$, C_1 achieve its control objectives. Similarly, C_2 also achieve its control objectives with ω, then ω is a system execution that satisfies the requirement of the system components.

Algorithm 2 Scheduler run-time operation

```
procedure SCHEDULE(A_s)                    ▷ schedule the system with strategy automaton A_s
    cur_state ← INITIAL(A_s)               ▷ start with the initial state

    loop
        y⃗ ← GETSTATEVARS()                 ▷ the scheduler observations
        cur_state ← NEXT(cur_state , y⃗)    ▷ take the environment transition for y⃗
        ⟨From, Tasks, To⟩ ← NEXT(cur_state)  ▷ take scheduler transition
        for all t ∈ Tasks do               ▷ execute all the tasks defined by the transition
            EXECUTE(t)
        end for
        cur_state ← To                     ▷ take the transition
        Wait till next time step
    end loop
end procedure
```

□

Definition 5. *A system is* ***schedulable****, if for any possible behavior of the environment (measurements that it may produce), there is a way for the scheduler to assign tasks such that all components achieve their control objectives.*

Claim 2. *If $C_1, \ldots, C_m$ are a well-formed components and there is a non-empty wining strategy for the composed component $C_s = C_1 \times \cdots \times C_m \times C_{resource}$, then the system defined by C_s is schedulable.*

Proof. First note that $C_{resource}$ is a well-formed component: The only environment state of $A_{resource}$ accept all $\mathbb{R}^n$ for the observations, therefore $C_{resource}$ is complete. And the only requirement from $C_{resource}$ is that no time step is never exceeded it length, this is guarantied by the contraction of scheduler transitions in the resource component, therefore $C_{resource}$ is a well-formed component. All components $C_1, C_2, .., C_m$ and $C_{resource}$ are a well-formed components then by Claim 1 C_s is also a well-formed component.

Assume there is a wining strategy for C_s then the scheduler that random walk trough the wining strategy automaton, and operate as elaborate in Algorithm 2 is:

1. Never get stuck: C_s is complete so each environment state of its automaton, A_s, have transition for any possible observation, therefore we never get stuck at environment state. And each scheduler state in the strategy have at least one outcome transition because if not then the strategy is not a wining strategy.

2. All the transitions of the component $C_{resource}$ game have a task sets that, even at the worst case, never exceeds a single time step. $T_{resource}$ contains all the system components subroutines, therefore, C_s also do not contain tasks sets that exceeds a single time step.

3. The system achieves the control objectives: The component C_s is well-formed,and the random walk of algorithm 2 is constructing a word $\omega \in Ł(C_s)$, therefore, by the definition of well-formed compnents (Definition 4), the run of algorithm 2 that construct the word ω achieves the system control objectives.

□

8.2 Some More Words on the Implementation

We defined the alphabet of a component game to be $\Sigma = T \cup \mathbb{R}^n$, but $\mathbb{R}^n$ is of course infinite set so we can not store the game in the finite storage of the control computer. We need to transfer the infinite continues space $\mathbb{R}^n$ to discrete space, without loosing any flexibility. In practice, we do not need to create explicit transition for every possible value of the observation $\vec{y}$. The environment transitions are annotated with values intervals, which define a subset of $\mathbb{R}^n$, for example the transition $q \xrightarrow{-1<y_1<1,-1<y_2<1,\ldots,-1<y_n<1} q'$ can be used for every $\vec{y'} = \langle y'_1, \ldots, y'_n \rangle \in \mathbb{R}^n$ such that $\forall 1 \leq i \leq n : -1 < y'_i < 1$. With this annotation we can store infinite games (and strategies solutions) in a small computer.

In addition, for successfully calculate the automata operations, such as, compose games with intersect intervals on the transitions, we need to proper define the intervals. The solution, which elaborated in details in the work of Merav [7], is to collect all the scalar numbers in all the component games, and define interval for every two adjacent scalars. Then the alphabet contain only tuples of n intervals for every n intervals combination.

With the above discrete alphabet, we can perform most of the automata operations with common automata algorithms and existing automata tools. In this thesis we used the GOAL tool [33] for composing games, finding wining strategy for a game and other analysis such as emptiness and simplification of automata.

An important characteristic of task scheduler is run time, sometimes referred to as switching time. The scheduler must consume minimum of CPU time. In the proposed methodology the scheduler need to perform many automata and game related operations, which can take significant processing time to calculate. In practice, all those calculations need to be performed only once for a system. We can perform all those calculations once in the compilation time, and then the scheduler only need to walk through the strategy automaton, which require only constant amount of calculations before every time slot.

9 Related Work

This thesis is mainly based on the RTComposer paper [2] and the followed work, GameComposer, of Merav [7]. In this thesis like in RTComposer we present a framework for design cyber-phisical systems with dynamic scheduling for real-time. RTComposer presents so called component-based framework in the sense that a single component correspond to

a single control loop in a complex cyber-phisical system. Each component has a clearly specified interface that includes method for sensing, method for actuation, and additional computational methods that can be executed between the time of sensing and actuation. Each component have requirement specification given by finite automaton over infinite words, every infinite sequence over the automaton specifies, for every time slot, the set of computational methods to invoke in that slot.

In this thesis we also continue the use of automata based scheduler, automata specification is expressive, analyzable, and composable specification framework as stated in Section 3 and in RTComposer paper [2]. After reviewing few embedded control systems, we conclude that RTComposer component design can be limiting. Most control systems have multiple control loops but they may have shared tasks. An example for such system is shown in Section 5, there is a three dimensional flying machine with four control loops (pitch, roll, yaw and throttle) that share a single image based sensing task along with some private tasks like compass sensor used only by the yaw control loop. In this work, unlike RTComposer, we loosen the definition of a component, and remove the separation of sensing and actuation methods, and remains only with computational methods, then a component is not necessarily correspond to a single control loop (see Section 8). In this way we can define a single component for the vision based sensing, ans 4 additional components for pitch, roll, yaw and throttle control loops that gets information from the vision component.

The GameComposer by Merav [7], similarly to this thesis, extends RTComposer by adding the ability of "reacting" to the environment. Inspired from GameComposer we use Büchi game as the main component specification, which allows the scheduler to make scheduling decision based on the current characteristics of the plant. Those games extends the specification automata of RTComposer with the addition of environment player and environment transitions (see Section 8). The environment may help or interfere with the control operation, we manage to improve the controller performance by consider this environment influence, see Section 6. General speaking Merav focuses on the technical aspects of the scheduler and automata related algorithms. This work focus on the integration of the automata based scheduling in a cyber-phisical systems.

RTComposer also describes techniques for generate the specification automata. For example, automata for a traditional periodic tasks, or automata that guaranty stability objectives for a given linear switched system. Those RTComposer techniques are all techniques for offline scheduling and they can be also applied in our proposed framework. In this work we present online techniques that can take in to account the current state of the system. We show how to use information from the classic Kalman fiter observer (Section 4) and a simplest linear version with complementary filter (Section 5.3. We manage to extract an estimation of the process and measurement errors from the filter, and schedule sensing tasks base on this estimation.

The case-study in this thesis, presented in Section 5, is based on the work of Hanoch [**?**]. Hanoch present an iterative technique for calculating the position and attitude of the drone relative to a window. In this technique the drone is equipped with a camera pointed to the front of the drone, the camera is capable of photographing a window and finding the four corners of the window in the image surface. Hanoch's algorithm takes the position of the corners and calculate the position and attitude relative to the window. In the case-study of this thesis we control and regulate the position and attitude of the drone (see Section 5.1) using the same techniques developed by Hanoch.

ask hanoch for reference

10 Conclusions and Future Work

We introduced a new approach for resource allocation that allows dynamic scheduling where resources are only allocated when needed.

We presented a compositional design strategy using rich automata based interface between software components and the task scheduler. We also proposed how to use this design in the context of software control systems using the data provided by standard filters. We demonstrated the advantages of the resulting dynamic schedulers in terms of control performance and resource utilization. These advantages are demonstrated both in simulations and with a real case-study.

The use of automata allows to compose and to analyze the scheduling characteristics. As future work, we plan to apply automata theory and, more specifically, the theory of hybrid automata to develop new techniques and tools for automatic generation of specification automata. The input for this construction can be a specification of the dynamical system that we want to control and a specification of the required close-loop characteristics.

11 Apendix

11.1 Kalman Filter and State Estimation

Te role of state estimation as the name suggest is to estimate the current state of the system, usually represented as the vector x. One can monitor the system with an array of sensors, the measurement devices intend to be uncertain, the measurement vector noted by $y = Cx+v$ where v represent the measurement error. Another estimation technique would be prediction, the system dynamics are known and the system inputs (u) are known, then create a model of the system using physics equations to predict the system state evolution in time (assume the initial state is known). The predicted state, noted by $\hat{x}$, is also not precise the real world is too complex to express in reasonable way, and we mark the predicted error by w.

In order to improve the estimation certainty a *filter* is added to the state estimation process, the filter aggregate the sensors and produce better estimation of the state. We will cover two well known filters, *complementary filter* and *Kalman filter*.

A short brief on *complementary filter*, this is basically combination of two filters (witch are complement each other), in control field it usually refers to *Low-Pass filter* and *High-Pass filter*. Low-pass filter allows the low frequencies signals to pass and filter out the high frequencies signals, is used for signals with high frequency error, e.g accelerometer signal. high-pass filter is its complementary, meaning it filters out the low frequencies signals, is used for signals like gyroscope that have continuous accumulated error.
General complementary filter notation: $\hat{f} = \alpha \cdot f_h + (1 - \alpha) \cdot f_l$ s.t. $\alpha \in [0, 1]$
if α is significant high, f_l is considered *low-pass*, as rapid value changes less significant and d slow value changes can be accumulated over time, and f_h is considered *high-pass*. In vision base measurements we use the only Low-pass filter: $\hat{x}_k = \hat{x}_{k-1} + \alpha \cdot (y_k - \hat{x}_{k-1})$.

Complementary and Low-pass are simple and very easy to implement but they not necessary produce the best possible estimation. A well known estimator called *Kalman filter* is a more complex but optimal filter, Kalman filter assume linear system ($x_k = Ax_{k-1} + Bu_k + w_k$) and zero mean Gaussian errors with covariance matrices $cov(w_k) = Q$ and $cov(v_k) = R$, if those conditions holds kalman filter produce the best possible estimation, based on all previous information available.

This is recursive algorithm that works in a two-step process, *prediction* step and *update* step. In the prediction step, the system model used to predict the current state, this called *A Priory Estimate* and noted by $\hat{x}_{k|k-1}$, the error covariance $P_{k|k-1}$ of the prediction is also calculated. Then in the *update* step the prediction is updated to find out the optimal estimation called *A Posteriori Estimate* noted by $\hat{x}_{k|k}$) and it's error covariance $P_{k|k}$. Practically the update step is a weighted average between the prediction ($P_{k|k-1}$) and the measurement (y_k), with more weight being given to estimates with higher certainty [19].

assumes the true state at time k is evolved from the state at $(k - 1)$ according to:

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

and the measurement at time K is:

$$y_k = Cx_k + v_k$$

Where x_k is the system state at time step k and y_k is the measurement by the sensors at time step k. w_k and v_k represent the process and measurement noise with covariance matrices Q and R. The system state at time $k - 1$ (x_{k-1}) is unknown, therefore the previous estimation is used to predict the next step state using the recursive equation:

$$\hat{x}_{k|k-1} = A\hat{x}_{k-1|k-1} + Bu_k$$

the error covariance of the prediction composed of the error of previous estimation which is: $cov(A\hat{x}_{k-1|k-1} - x_{k-1}) = Acov(\hat{x}_{k-1|k-1} - x_{k-1})A^T = AP_{k-1|k-1}A^T$ and the process error of time k (Q) and we get prediction error covariance of:

$$P_{k|k-1} = AP_{k-1|k-1}A^T + Q$$

After the prediction was calculated the update step combine the prediction and measurements to get the Kalman estimation:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(y_k - C\hat{x}_{k|k-1})$$

K_k, also called Kalman Gain, represent the contribution of each partial estimation ($\hat{x}_{k|k-1}$ and y_k) to the Kalman estimation, and is calculate each time by:

$$K_k = \frac{P_{k|k-1}C^T}{CP_{k|k-1}C^T + R}$$

Note, if the error covariance (Q and R) are constants, K_k is converged to a constant value. The Kalman estimation covariance is:

$$P_k = (I - K_kC)P_{k|k-1}$$

The Kalman filter is designed for linear systems. It is not suitable in the case of a nonlinear systems, if the state transition function or the measurement function are nonlinear:

$$x_k = f(x_{k-1}, u_k) + w_k$$

$$y_k = h(x_k) + v_k$$

A Kalman filter that linearizes the nonlinear function around the mean of current state estimation is used, this referred to as an extended Kalman filter (EKF). The transition and mesurment functions must be differentiable in order to linearize them, and the Jacobian matrices are used. Extended Kalman filter is not guaranty to be optimal filter.

11.2 Generalize Non-Deterministic Büchi Game (GNBG)

Generalize Non-Deterministic Büchi automaton (GNBA) $A = \langle Q, \Sigma, \Delta, q_0, \mathcal{F} \rangle$, is an ω-word automaton (a variant of Büchi automaton). Similar to all types of finite automata [30], Q is the set of states, q_0 is the initial state and $\Delta : Q \times \Sigma \to Q$ is the transition function over the alphabet Σ. The main difference between all finite automata types is the acceptance condition. In generalize Büchi , $\mathcal{F} = \{F_1, \ldots, F_m\}$, where $F_i \subseteq Q$. A valid run of A that correspond to $\omega \in \Sigma^\omega$ ($\omega = \omega[1], \omega[2], \ldots$), is a sequence $p = q_0, q_1, \ldots$ such that q_0 is the initial state, and $\forall i > 0 : q_i \in \Delta(Q_{i-1}, \omega[i])$. The GNBA automaton A accepts infinite

word $\omega \in \Sigma^\omega$ iff there exist a run p correspond to ω such that $\forall 1 \leq i \leq: \inf(p) \cap F_i \neq \emptyset$ [6], noted $\omega \in \mathcal{L}(A)$.

Generalize Non-Deterministic Büchi Game $G = \langle A, \langle P_{schd}, P_{env} \rangle \rangle$, is a two player game, and is played on the induced graph $\langle V, E \rangle$ of A. In this graph, $V = Q$ and for every $q, q' \in Q$ such that there exist $\sigma \in \Sigma$ that $q' \in \Delta(q, \sigma)$ we add the edge $\langle q, q' \rangle \in E$. During the game $P_{schd} \subseteq Q$ are all the graph nodes where the *sched* player choose the next transition, and $P_{env} \subseteq Q$ are all the graph nodes where the *env* player choose the next transition. P_{schd} and P_{env} is a division of Q, i.e $P_{schd} \cup P_{env} = Q$ and $P_{schd} \cap P_{env} = \emptyset$ the *sched* player win the game iff the resulting run p of the implies that, $\forall 1 \leq i \leq: \inf(p) \cap F_i \neq \emptyset$. In other words, if *sched* player win, the corresponding word ω of this game is in $\mathcal{L}(A)$.

Note: for the mater of this thesis, the turns are always alternating, and the *env* (envirinment) player is always first. Hence, the notion of P_{schd} and P_{env} is not needed. $q_0 \in P_{env}$, then for every $\langle p, p' \rangle \in E$ $p' \in P_{schd}$, and so on.

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. *8th ACM/IEEE Conference on Embedded Software*, 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. *Introduction to stochastic control theory.* Courier Corporation, 2012.

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

[6] 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.

[7] M. Bukra. GameComposer: A Framework for Dynamic Scheduling. Master's thesis, Ben-Gurion University, Israel, October 2014.

[8] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. rzn. How does control timing affect performance?, 2003.

[6] $\inf(p)$ is the set of elements in p that appears infinite times in p.

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

[10] 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.

[11] 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.

[12] A. Developers. ArduPilot Mega (APM): open source controller for quad-copters. www.arducopter.co.uk.

[13] H. Efraim. *Vision Based Control of Micro Aerial Vehicles in Indoor Environments*. PhD thesis, Ben Gurion University of The Negev, 2017.

[14] 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.

[15] EMLID. navio: Linux autopilot on Raspberry Pi. https://emlid.com/introducing-navio2/.

[16] M. Esnaashari and M. Meybodi. A learning automata based scheduling solution to the dynamic point coverage problem in wireless sensor networks. *Computer Networks*, 54(14):2410 – 2438, 2010.

[17] 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.

[18] W. T. Higgins. A comparison of complementary and kalman filtering. *IEEE Transactions on Aerospace and Electronic Systems*, (3):321–325, 1975.

[19] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

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

[21] 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.

[22] J. Liu, N. Ozay, U. Topcu, and R. M. Murray. Synthesis of reactive switching protocols from temporal logic specifications. *IEEE Transactions on Automatic Control*, 58(7):1771–1785, 2013.

[23] 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.

[24] Raspberry Pi Foundation. Raspberry pi. www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[25] 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.

[26] 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.

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

[28] 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.

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

[30] M. Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.

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

[32] 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.

[33] Y.-K. Tsay, Y.-F. Chen, M.-H. Tsai, K.-N. Wu, and W.-C. Chan. Goal: A graphical tool for manipulating büchi automata and temporal formulae. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 466–471. Springer, 2007.

[34] 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.

תקציר

בעבודה זו אנו מציגים גישה, הוכחת היתכנות ע"י כלי שיבוץ, וניסוי, לשיבוץ תגובתי (reactive schedule) של משימות חישוב במערכות בקרה מבוססות מחשב. שיטות שיבוץ (תזמון) אלו נדרשות בעיקר עקב השימוש ההולך וגדל של אלגוריתמי עיבוד תמונה ממוחשב במערכות זמן-אמת, המשמשים כחיישנים עבור מערכות הבקרה. המטרה בפיתוח של מערכות שיבוץ תגובתיות היא לשלב בין הגמישות והיעילות של המשבצים במערכות הפעלה למחשב האישי, ובין יכולת החיזוי (הבטיחות) שמספקים משבצים במערכות הפעלה בזמן אמת. הביטוי "תגובתי" (reactive) בעבודה זו מתייחס ליכולת של המתזמן "להגיב" לסביבה ולהתאים את מאפייני התזמון לתנאים הפיזיים (והסביבתיים) בזמן-אמת. בעבודה זו, אנו מציגים מנגנונים המאפשרים למתזמן להגיב באופן מתמיד לקלטים מהסביבה ולשינויים במצב הפנימי של המערכת.

אנחנו מציגים הרחבה של גישות תזמון מבוססי אוטומטים, כמו שמומש לדוגמה ע"י הכלי RTComposer, עם התוספת של תנאי סף (guards) על המעברים של האוטומט. תוספת זו מאפשרת ממשק שבו המתזמן מגיב למידע מהסביבה החיצונית ומהבקר עצמו. אוטומטים משמשים כממשק עשיר וגמיש עבור פירוט הדרישות של משימות בקרה. הם מאפשרים לבצע שיבוץ גמיש אך מפוקח (שיבוץ תגובתי) ללא הפרה של יכולת החיזוי שמספקים משבצים בזמן-אמת. גמישות זו מאפשרת שיבוצים יעילים ע"י כך שהמשבץ משבץ משאבי חישוב גדולים רק בתנאים (לדוגמה תנאי סביבה) שבהם הם באמת נדרשים. המשאבים שנחסכו (זמני מעבד לרוב) ע"י תזמון תגובתי יכולים לשמש עבור משימות משניות, או עבור הפחתה של של העלות הכללית של המערכת, כלומר הפחתת דרישות המשאבים.

התרומות העיקריות של עבודה זו ביחס לעבודות הקודמות של RTComposer ושל GameComposer הן: (1) אנחנו פיתחנו שיטות ליצירת רכיבי בקרה מבוססי-סביבה (environment depended), אשר משתמשים במסנן קלמן (Kalman filter) או במסנן משלים (complementary filter), ומנצלים נתונים חיוניים המתקבלים (כתוצר משני) של מסננים אלו בכדי להנחות את המשבץ. (2) בכדי לאמת את הגישה הנ"ל ערכנו ניסויי של דוגמא מהעולם האמיתי וגם יצרנו הדמיה (סימולציה) ממוחשבת.

משימת הבקרה ששימשה אותנו בביצוע הניסויים היא פיתוח של בקר ממוחשב אשר מייצב רחפן ממול חלון, המשתמש במצלמה כדיי להתמצות ולזהות את החלון. בעזרת הניסוי אנחנו מדגימים איך בעזרת שימוש ברכיב מדידה (חיישן) המבוסס על עיבוד תמונה, בעל אפשרות לעבוד עם גודל תמונה (רזולוציה) משתנה, אפשר לאזן ולשלוט על עומסי המשאבים במערכת. כלומר, בניסויים אנחנו מווסתים את העומס על המעבד ע"י שימוש בתמונות בעלות פחות סיביות כאשר תנאי הסביבה ומצב הבקר מאפשרים זאת ללא פגיע משמעותית בביצועים. אנחנו פיתחנו את מערכת הבקרה הזו על בסיס תוכנת בקרת טיסה הנפוצה ArduPilotMega (APM) שמפותחת כתוכנת קוד-פתוח, והוספנו שינויים מזעריים כדיי להתאימה למשימה מונחית עיבוד תמונה. העובדה שהתבססנו על תוכנת בקרה נפוצה במימוש הניסוי עזרה לנו במהלך המחקר להגיע למסקנות טובות יותר לגבי הדרך הנכונה לשלב את גישת שיבוץ התהליכים מבוסס אוטומטים במערכות הבקרה הנפוצות בימינו.

אוניברסיטת בן-גוריון בנגב
הפקולטה למדעי הטבע
המחלקה למדעי המחשב

שיבוץ תגובתי של משאבי חישוב במערכות בקרה ממוחשבות

חיבור לשם קבלת התואר "מגיסטר" בפקולטה למדעי הטבע

מאת: הודי גולדמן
בהנחיית: דר' גרא וייס

דצמבר 2018

Need to add contents in hebrew?