NTNU – Trondheim
Norwegian University of
Science and Technology

Using online worst-case execution time analysis and alternative tasks in real time systems

Fredrik Bakkevig Haugli

Master of Science in Cybernetics and Robotics
Submission date: June 2014
Supervisor: Sverre Hendseth, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Assignment

This assignment will be about online execution time analysis: The idea that we can obtain a better estimate of a tasks execution time after it has been released, based on program state, input etc. than we can with offline worst case execution time estimates. Not much work has previously been devoted to online execution time analysis today, and the emphasis of this assignment will be to explore the concept.

The parts of the assignment are:

A Find a suitable application for demonstrating the benefits of online execution time analysis

B Find a way of scheduling tasks that exploits the online execution time estimate

C Perform an experiment demonstrating the benefit of using online execution time analysis

D Write a paper to be submitted to the 2014 WCET Workshop

E If time allows: Design and implement a scheduling simulator that can be used to perform further experiments on online execution time analysis

Preface

This thesis is the conclusion of my Master studies at the Department of Engineering Cybernetics at the Norwegian University for Science and Technology.

I would like to thank my supervisor Sverre Hendseth for his guidance, motivation and interesting discussions.

I would also like to thank Amund Skavhaug for his assistance in writing the paper and for providing input on relevant previous work.

Lastly, I would like to thank Robetro Rigolin Rerreira Lopes for his help and advice on paper writing.

Fredrik Bakkevig Haugli

Abstract

As embedded hardware becomes more powerful, it allows for more complex real time systems running tasks with highly dynamic execution times. This dynamicity makes the already formidable task of producing accurate WCET analysis even more difficult. Since the variation in execution time depends on task input and the state of the system, it is postulated that a more accurate estimate for the WCET can be found online with knowledge about the task parameters.

This thesis will explore the concept of online execution time analysis and its potential utilization. Line detection in images through Hough line transform is found to be a relevant application whose execution time can be estimated by the contrast of the input image. A system for scheduling tasks utilizing their online WCET estimate is then discussed. It dynamically checks for potential deadline misses and degrades tasks, either by running a more efficient alternative task instead or by aborting the task, until timely execution is guaranteed. An experiment is presented, demonstrating a higher throughput of tasks with online WCET estimation. Finally, the work on a framework for more precise simulations and experiments is presented.

Sammendrag

Etter hvert som maskinvare for sanntids datasystemer blir kraftigere, blir også programmene som kjøres mer komplekse. Dette medfører at kjøretiden blir vanskelig å beregne nøyaktig. Siden variasjonene i kjøretid avhenger av programmets input samt systemets generelle tilstand, anntas det at et bedre kjøretidsestimat kan finnes når man kjenner til disse parameterene.

Dette arbeidet utforsker ideen om å foreta en kjøretidsanalyse mens systemet er i gang og den eventuelle nytten og bruksområdet en slik analyse vil kunne ha. Linjedetektering i bilder via Hough-transform blir brukt som eksempelapplikasjon. Kjøretiden blir vist å være omtrent lineær med kontrasten i bildet og kan dermed beregnes utifra denne. Et system for tidsdeling av oppgaver som tar i bruk online kjøretidsestimater blir så diskutert. Det sjekker for mulige overskridelser av tidsfrister og kan etter behov bytte ut oppgaver med alternative varianter som bruker mindre tid eller avbryte dem. Et eksperiment blir presentert som demonstrerer at et høyere antall oppgaver blir gjennomført i tide ved bruk av online kjøretidsestimering og alternative tasks. Til slutt er et påbegynnt arbeid på et rammeverk for videre eksperimentering presentert.

Glossary

$WCET_{OFF}$ Offline Worst Case Execution Time. The worst case execution time of a task calculated while knowing nothing of the actual program state and therefore very pessimistic as it can make no assumptions. 2, 8, 9

$WCET_{ON}$ Online Worst Case Execution Time. This is a more precise estimate of a tasks execution time calculated while the systems is running, using certain parameters that are available during execution time. 2, 9

computer vision An advanced form of image analysis, mainly used as input for robots, drones and other intelligent systems. 14

edge pixel An image pixel that lies on the edge between two or more homogeneous areas of the image. 12

fork A system call that spawns a new process. 40

OETA Online Execution Time Analysis. 2, 9

OpenCV An open source library providing basic and advanced computer vision functionality. 15

pthread A POSIX library for running and synchronizing programing threads. 39

WCET Worst Case Execution Time. 2

Contents

1 Introduction

A continuing trend in real time embedded systems is that the complexity of their programs increases. In addition to this, an increasing number of tasks are moved onto the same physical hardware. One example of this development is the emergence of smart phones. Modern smart phones have evolved from cell phones that could send and receive calls, and very little else, into what are essentially tiny computers, running a full fledged operating system and applications made by completely unrelated programmers. A similar transition will likely follow in other areas, such as home automation and electrical distribution system (Smart Grid).

As the tasks of real time systems grow increasingly more complex they exhibit a higher degree of *dynamicity*, meaning that the execution time varies, for example based on input values. It is assumed that this fact can be exploited to allow the scheduler to make a better execution time estimate at task release using online data like task input and general system state.

This thesis will explore the idea of using online execution time analysis (OETA) as a tool for the scheduler to make better decisions. In a preliminary project [5], the OETA was used to help in scheduling mixed criticality systems. These are real time systems that run many tasks of varying importance, and where the scheduler can abort a task of low importance (criticality), if it is necessary in order to ensure that a more critical task completes within its deadline. The basic idea of the work done in the preliminary project was to use the OETA to determine if it really is necessary to abort tasks of low criticality given the current set of active tasks, and not just based on a model.

The rest of this thesis is organized as follows: Section 2 contains background information. In Section 3, an example application is found. Section 4 presents a novel way of exploiting online WCET analysis for scheduling. Section 5 presents an experiment simulating the benefits of OETA scheduling. Section 6 describes the work done in implementing a scheduling simulator and Section 7 concludes this thesis.

2 Background

2.1 WCET Analysis

One has traditionally used very pessimistic estimates for execution times, WCET, when designing real time systems and checking if they will be able to handle all tasks without missing deadlines. However, the assumption that all tasks will always execute for their Worst Case Execution Time (WCET) often results in severe under-utilization of the systems resources, since most tasks have a varying execution time. They can, for example, spend merely a small percentage of their WCET most of the times they run and only in a few cases, or potentially never, spend the entire WCET.

Instead of always using the worst case execution time, a system was suggested by Sverre Hendseth and Giorgio Buttazo in [6] where the scheduler checks certain task parameters (input, program state etc.) to determine a better estimate for how long the task will need to execute in that particular release. This execution time estimate, computed online, will be called $WCET_{ON}$, and the traditional WCET determined offline will be referred to as $WCET_{OFF}$. The system utilizing the added information provided by the $WCET_{ON}$ will be referred to as Online Execution Time Analysis (OETA). In this system each task will have two entry points: the regular entry point for the actual task code and one for the OETA code. This allows the scheduler to run the analysis and get a better estimate of actual runtime required by the task than the $WCET_{ON}$ before letting the task run.

The idea of computing the execution time of a task at runtime has also been suggested by Stancovic et al. [10] for use in the Spring kernel. This system allowed a tasks execution time to be "a formula that depends on various input data and/or state information". Instead of giving each task two entry points which allows for complex program code to compute the execution time, it is done with a simple mathematical expression.

Parametric timing analysis is another method that has been previously expored, in relation to energy saving [9]. It uses loop bounds to predict execution time online.

2.2 Computer Vision in UAVs

Unmanned Aerial Vehicles (UAVs) are pilot-less aircrafts that operate either by remote control or autonomously. They come in different sizes and configurations from small quad-copters the size of a hand to large airplane-like vehicles like the Predator drone. Most UAVs have some sort of camera, either to allow the operator to see where the UAV is, or to be used as a sensor for autonomous control, for example to enable collision avoidance. Advanced image processing is an important component in UAVs [4], used for example for autonomous landing [12]. UAVs have traditionally been used for military applications, such as gathering intelligence, aiming and firing missiles. Lately several civilian applications for UAVs have been proposed, such as search and rescue missions, filmmaking and delivering parcels.

Another possible application for UAVs is surveillance of power lines [8]. Many long power lines go through forests and other terrain with a lot of vegetation. These have to be regularly monitored to ensure that the surrounding vegetation does not grow into the power lines. This is usually done with a manned helicopter, costing large sums of money. An UAV with a camera that automatically follows the power lines and takes pictures will make this job both easier and cheaper.

2.3 Degraded Level of Service

In some systems it can be desirable to have the possibility to resort to a degraded level of service. Consider a UAV used to perform surveillance of powerlines, as discussed in 2.2. The onboard computer will have to perform many tasks in order for the UAV to do its job, such as:

- Keeping a stable flight (not crashing)
- Receiving commands from the operator
- Sending flight status (position, battery level etc.) information back to the operator
- Flying along the correct path
- Taking pictures of the power lines

- Analyzing the pictures

Clearly, not all of these tasks are equally critical. For example, it's more important to keep the UAV in the air, than to perform the picture analysis. If the CPU doesn't have enough resources to handle all the tasks, it should abort the low criticality tasks instead of the high criticality ones, as there is no point in doing a good picture analysis if it causes the UAV to hit the ground directly afterwards. Aborting the low criticality tasks to ensure that those of higher criticality are run, is a form degraded level of service or graceful degradation [3].

Ensuring that the tasks that are regarded as critical are prioritized can be done for example either by explicitly giving the task a high priority, or implicitly through for example period transformation. This method divides a highly critical task is into several tasks with shorter period in a system with a rate monotonic priority assignment scheme, thereby giving it a high priority. Other systems have a inherent concept of criticality, like the Adaptive Mixed Criticality Scheme (AMC) [1].

2.4 Adaptive Mixed Criticality

Most of this section is unchanged from the report written on the authors preliminary project [5].

A classic real-time system consists of a set of tasks that run together on the same hardware, and must complete execution before their respective deadlines. To assure that every task finishes before its deadline, each task is given a priority, determined by some scheduling policy. If a task with a higher priority than the currently running task is released (becomes ready to run), the lower priority task will be interrupted in order for the new task to be run, and then resumed after the high priority task is finished. These priorities does not, however, reflect the criticality of the task. In an aircraft the low criticality task that handles the cabin air conditioning may be given a higher priority than the task making sure that the landing gear is lowered on time (a very highly critical task) if the scheduling algorithm finds that this arrangement of priorities makes sure that all tasks will finish by their deadline.

This does, however, tend to become very inefficient as more tasks of varying criticality are moved onto the same hardware. This trend has become increasingly prominent, as concerns about space, energy efficiency and cost have grown [2]. The issue is that since important, potentially life-preserving, functionality is a part of this system, the other less important tasks must undergo the same rigorous tests and certification. This might lead to a severe under-utilization of the system, since making sure that all the highly critical tasks finish by their deadline requires all tasks to be given a pessimistic execution time estimate. In theory, one must allocate the WCET to each task. Finding the exact WCET is, however, not trivial [11]. The solution then, is to give each task a very generous WCET, to be "sure" that the task will not exceed it.

One way of handling different degrees of criticality is by giving highly critical tasks a high priority, and thereby ensuring that they will run without interference from the non-critical tasks.

In [11], Steve Vestal proposed a scheme for scheduling mixed criticality systems. The motivation behind this was that the different tasks of a real-time system does not necessarily need the same level of assurance in regards to certification. When a system is to be certified, only a certain number of the tasks are actually interesting to the certification authority. These are the highly critical task, e.g. those that deal with preventing damage to the system, death etc.

Several models have since been suggested, that improve the original work in [11]. One of these, called Adaptive Mixed Criticality (AMC) has been shown [1] to surpass the others in amount of task sets that are schedulable. In his 2013 review article on mixed criticality systems [2], Alan Burns writes that this is still the case[1]. The basic idea of AMC is that if a task executes for longer than its allotted time at a given criticality level, all tasks of the lowest criticality are aborted.

2.5 Alternative tasks

Another form of degraded level of service is the use of alternative tasks. Instead of completely aborting a task, it may be possible to perform a simpler, less time

[1] He does, however, note that an extension of the AMC by Zhao et al. has improved the stack usage.

consuming algorithm instead. This way, the purpose of the task is still fulfilled, although with lower quality. Such a system is described in [7] page 92.

Alternative tasks can be used in two ways. One is as a back up mechanism to be used when the normal task cannot be used. Another way of looking at it is a possibility of running a task variant that you normally would not have the resources for, but that you every now and then can run when the resource demand in the rest of the system is very low.

The use of alternative tasks was also suggested as a feature in the Spring Kernel [10]. Here it is used as a backup in case there is not enough time to run the proper task. Burns and Wellings [3] discuss the use of alternative modules. These are alternative tasks that can be called if the output of the original task fails an acceptance test.

3 Finding example Application

3.1 Criteria

Part of the work has been devoted to finding a suitable application for the use of online execution time analysis. A suitable application should exhibit the following properties:

High degree of dynamicity The application must have a varying execution time that can easily be computed based on information available online.

Applicable in real-time systems Since real-time systems rely heavily on the execution time of tasks, they are believed to benefit the most from such a system.

Relevant for cybenetics applications This is not a absolute demand, but would be an attractive property for an application.

Finding a good application proved to be more challenging than initially believed. Even though the execution time of most computer algorithms depend on the input size, when used in a practical application the input size from iteration to iteration rarely changes. For example doing matrix calculations is a part of many control systems, and it is trivial to show that inverting a 50 by 50 matrix takes longer time than inverting a 5 by 5 matrix. However, the sizes of the matrices in a given system rarely changes. They are usually given by the number of state variables and inputs to a system.

There were two candidates for an application: Variable MPC controller and Computer vision in UAVs.

This section start with a presentation of the online WCET analysis. Then the two alternatives are presented and the decision is made. Finally an experiment evaluating the applicability of the application is described.

3.2 Online WCET Analysis

Online WCET analysis is the concept of using online data to make a better estimate for the execution time of a task than what can be achieved with traditional offline

analysis. There are mainly two types of data that can be used: task input and system state.

Task input can affect execution time in several ways. The most obvious is that most algorithms have an execution time that is a function of the input size. Sorting a list of 10 elements, for example takes less time than a list of 10000 elements. In addition to the input size, the value or type of input can dictate the time it takes to process it. A message handling task can for example spend a relatively constant amount of time handling normal messages, but spend a lot longer handling other such as error messages that require the task to signal other parts of the system or take other extraordinary actions.

System state can be used to estimate execution time in multiple ways. If the system is modeled as a state machine a tasks behavior might be dictated by the current state. One can also use information about shared resources (whether they are locked, how many tasks are waiting etc.) to estimate how long it will take for a given tasks to access it and continue running. Hardware state is another part of the system that can be used. Many smart phones, for example, have a low power mode that can either be activated manually, or is automatically enabled when the battery reaches a certain charge level. This mode can for example put some component in a sleep-state and will thus impact the execution time of a task that needs to wait until the component is active again.

There are several challenges regarding the use of online data for estimating the execution time. One is how the actual estimation will be done. In some cases it may be relatively easy, if the task is simply traversing a list, performing a matrix inversion or processing a message frame. Here we can use the size of the list, matrix dimensions etc. as a parameter for our estimation algorithm and can use either a simple mathematical function, like suggested in [10], or values in a previously computed lookup-table. It gets slightly harder if the execution time is dependent, not only on the size of the input data, but on the actual data itself. Some of the hardest are those where the execution time depends on some of the calculations themselves. In the worst case scenario the estimation will have to perform the entire task program in order to find an estimate. In this case one might be better off with just having the estimation function return the static $WCET_{OFF}$. Another problem is tasks that can block, for example while waiting for a semaphore, or a

message from another thread. In some cases one can potentially take the blocking time into account when designing the estimation algorithm, or have the scheduler check the state of blocking semaphores when computing the $WCET_{ON}$

Another problem is the fact that the scheduler will be executing the estimation function, which is written by an application programmer. This function may take a lot of time and introduce an overhead that need to be taken into account during system design, schedulability proofs, etc. In some cases it might be necessary to add the execution time of the OETA algorithm to the $WCET_{OFF}$ of the task in schedulability proofs. In some systems that separate between user- and kernel-space code, another problem may occur, as application code is run by the kernel. Such a system could run the estimation code with limited permissions. On the other hand, in an embedded real time system, one usually have complete control over what tasks will be running on the system, and the probability that someone will sneak in "malicious" code is not as big as for example in a desktop system. A lot of real time operating systems do not operate with distinct kernel and user space.

The fact that the online execution time analysis takes time is an issue that must be taken into account when verifying the schedulability of the system.

An important requirement of the estimation algorithm is that it must never give an estimate that is too low. That is, in order to be useful, it should provide a better estimate than the $WCET_{OFF}$, but it must still be sufficiently generous in order to ensure that the actual execution time does not exceed the $WCET_{ON}$. It is, after all, still a *worst* case execution time, although computed with better accuracy, due to the availability of more information. Furthermore, the execution time of the estimation algorithm should be reasonably modest, so that it does not incur excessive overhead on the system.

3.3 MPC controller with variable time horizon

One idea was to use the online execution time analysis to make an MPC controller with a variable time horizon. An MPC controller generates an input by modeling the future and thereby "looking ahead" to the best future state and then giving the system an input that corresponds to the first step of the planned input. It

is possible that a longer time horizon would allow a better prediction to be made and indirectly allow for a improved control of the process. It would, however take longer to run the model prediction.

One could make a system that uses the $WCET_{ON}$ of all tasks currently running in the system to determine how large time horizon could be used for the MPC-controller, while ensuring that all other active tasks finish by their deadline.

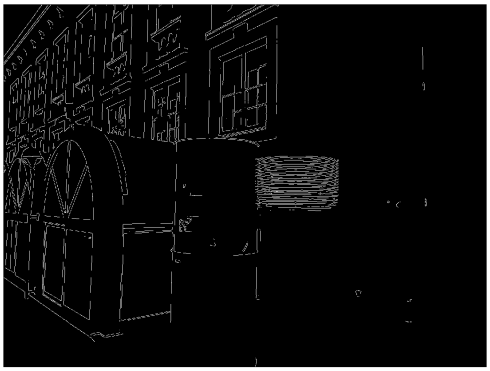
3.4 Hough Line Transform

The Hough transform is a method for finding shapes in an image. The shapes to be detected has to be parameterizable. For example, a straight line can be represented by its angle θ and distance to the origin r. The line can then be mapped as a point in the (θ, r)-space. If every line that goes through a point is mapped in (θ, r)-space, it forms a wave-like line linke the ones seen in Figure 1c. The Hough Line Transform is briefly described below, followed by a more in depth description with an example. It keeps an accumulator array of all possible (θ, r)-values.

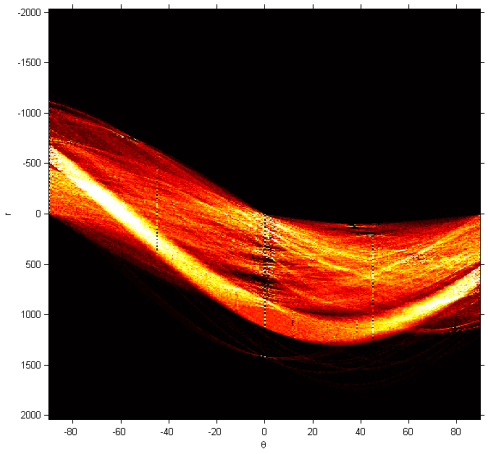
1. A preliminary blurring using a Gaussian filter to remove noise from the image.
2. Perform an edge detection algorithm, such as the Canny edge detector to locate all the edge pixels (edge pixels are explained below).
3. For every edge pixel find the (θ, r)-values of all lines that goes through the point.
4. Increase the values corresponding to the (θ, r)-values in the accumulator array.
5. Pick out the lines that have the larges values in the accumulator array.

Canny edge detector Edge pixels are pixels that lie along the edge between two uniformly colored areas of an image. The Canny edge detector outputs a binary image where all the edge pixels are white and the rest black. This is shown in Figure 1b.

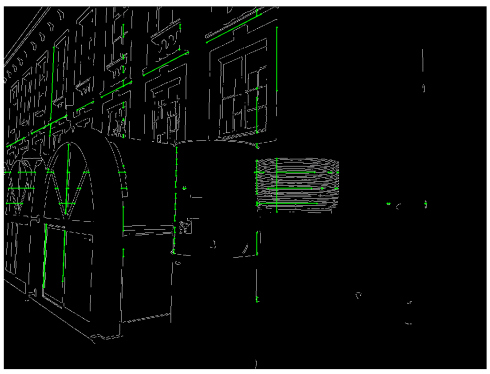
(a) Original image



(b) After Canny edge detection



(c) Accumulator array



(d) Detected lines

Figure 1: The steps of the Hough Line Transform

Find the parameter values of all lines that might go through the point A set of quantized θ-values are chosen. For a given edge pixel (x_i, y_i). All (θ, r)-values that satisfy

$$r = x_i cos(\theta) + y_i sin(\theta) \tag{1}$$

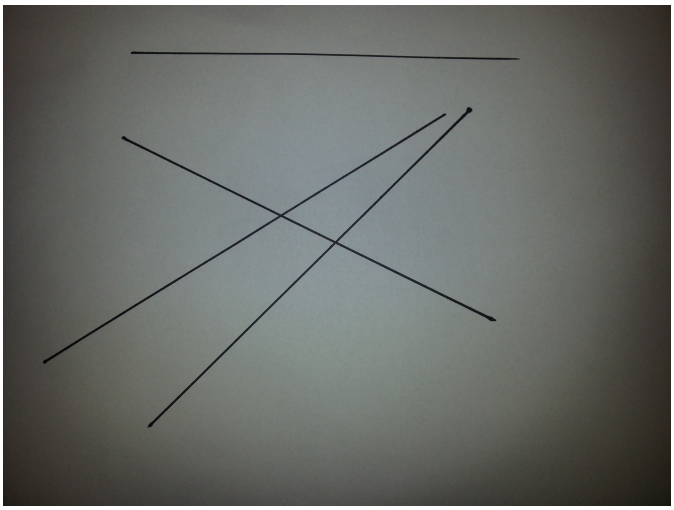
represent lines that pass through (x_i, y_i). The corresponding value in the accumulator array is increased.

The accumulator array is shown in Figure 1c.

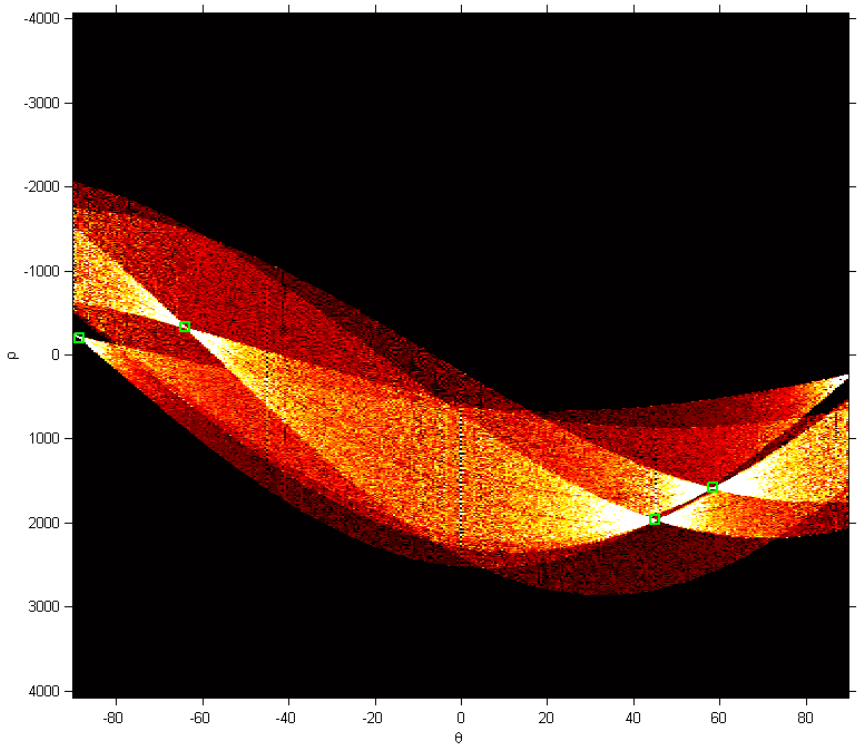
Pick out the lines with the highest value in the accumulator array When all edge pixels has been added to the accumulator array, a given value in the accumulator array $A(\theta_i, r_i)$ contains the number of edge pixels that lie along the line represented by (θ_i, r_i).The maximas of this array correspond to a line in the image. Usually a threshold is used to select which of the values will be considered a line.

For every point in the original image, we can plot all lines that potentially go through the point. This plot looks like a sine-wave. We do this for all points that are candidates for line pixels and see if their "sine wave" intersect. The intersection point in (θ, r)-space represents a line that the points of all the intersecting waves lie on. One can then choose to either return the n longest lines, all lines with more than a given number of pixels, all lines with a certain angle or another subset of possible lines in a picture. Figure 2 shows a simple image with four lines, and the corresponding accumulator array. The four lines can easily be seen as the maximas in the accumultor array.

One possible use case for this is in the power line surveillance example[8]. An UAV could take an image, estimate how long it would take to analyze it and if it would take longer than a set limit, it could wither slow down the UAV to give itself more time to analyze before a new image is taken, or scale down the image, to make the processing go faster, but be less accurate.



(a) A simple image with four lines



(b) The accumulator array, the four maximas are marked with green squares

Figure 2: Correlation between the lines in an image and the accumulator array

3.5 Decision

It was decided to examine the Hough transform further. The reasons it was chosen over the MPC controller are:

It is more suitable for small embedded systems with limited resources. The Hough transform is being used in UAVs, which have limited battery supply and therefore limited computation power. The UAVs computation power must also be used for other purposes, like hardware control and communication. The MPC-controller, on the other hand, is primarily used in larger systems like oil platforms, ships and chemical factories. In this setting one could easily buy a relatively powerful computer just to run the MPC algorithm. This reduces the benefit of developing a more time-efficient MPC-controller.

It is easier to test. Making a test program to analyze a set of pictures can be done easily with the Hough transform. There are already computer vision libraries available online that performs the hard work, such as OpenCV.

In contrast, testing the MPC algorithm would need a process to be controlled, as well as a set of tasks to be run on the same hardware with variable execution time that can be executed.

The runtime of the Hough transform can be reduced by scaling down the image. In a UAV setting the scheduler can choose to scale the image down if there is not enough time to run the Hough transform on the entire image.

One problem with using the Hough Transform is that it requires a lot of tuning to produce good results. When studying the lines detected in Figure 1d, it is clear that not all lines in the original image were detected, and that some marked lines are false positives. There are several ways to tune the Hough transform: Changing the Gaussian filter size used for blurring, the threshold values for the Canny detector, and the threshold and resolution values for the Hough transform itself. These values must be tweaked for the images used in the specific application in order to obtain a good result. In this work, no effort has been made to ensure good line detection, as this would require a lot of time. It also would probably not be possible to find a set of parameters suiting a diverse set of pictures like the

one used in this work. This is a problem since it makes it impossible to determine whether a reduced variant of the Hough transform gives a satisfactory result.

3.6 Testing execution time estimate

The hypothesis is that the number of edge pixels corresponds linearly to the execution time of the hough transform. To test this, 100 pictures was taken of buildings in and around the NTNU campus. They were then run through an OpenCV implementation of the Hough line transform and timed. The results are shown in Figure 3.

The resolution of these images are 1632x1224 pixels. Figure 3 shows that the execution time of the Hough analysis is approximately linear with respect to the number of edge pixels.

Figure 4 shows all the data points in the experiment. This shows that the variations are relatively small. One can easily make an online WCET estimator by making a linear function that goes above the data points like $WCET_{ON} = \frac{3}{3500000} * n_{edgepixels} + 0.15$. Note that this is not found analytically and is only a statistical observation. A proper real time system would demand more rigorous analysis.

The next step would be to see how long each of the steps of the Hough transform takes compared to the others. For this test, five stages of the Hough transform was timed: loading the image, blurring, running the Canny edge detector, counting the edge pixels and adding all the edge pixels to the accumulator array. The latter is referred to as "Hough Transform" in the graph because this is the name of the OpenCV function responsible for this. The timing has been done by running the same program multiple times and aborting after each step. Because of this, the numbers for the Canny edge detector comes from running the image loading, blurring and Canny detector. In retrospect, it would be better to run the entire algorithm and output the time after each step. The results are shown in Figure 5.

Several observations can be made from Figure 5. The first is that the "preprocessing", i.e. everything up to the Hough Transform takes more or less constant time. Furthermore, counting the pixels takes very short time. This is good as this step constitutes the added overhead needed in order to perform the OETA.

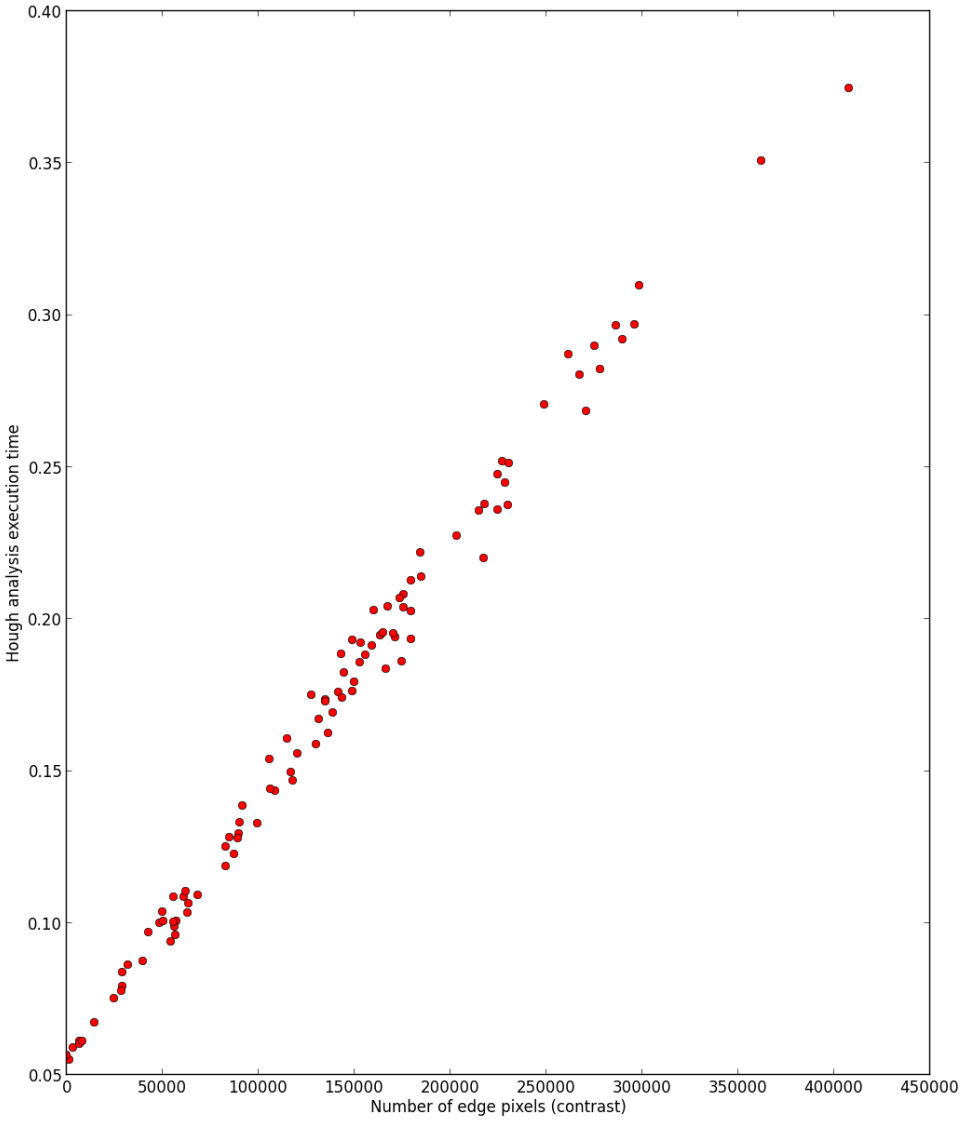


Figure 3: Average Execution time of the Hough transform relative to the number of edge pixels

By doing the pixel counting as part of the Canny edge detector step, the time consumption could possibly be lowered further. However, as this would require

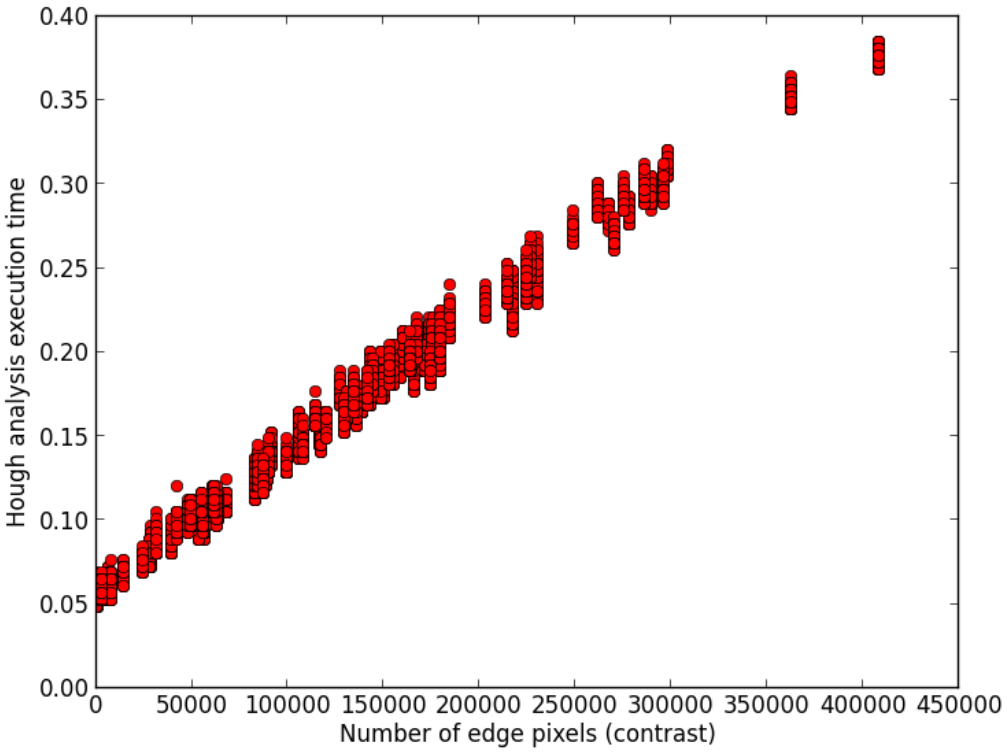


Figure 4: Execution time of all runs of the Hough transform relative to the number of edge pixels

rewriting the OpenCV library, it was not done for this experiment.

Figure 6 shows the execution time of the same images scaled down by 50%. This shows that scaling the images down is a possible way of reducing the execution time of the Hough transform. As previously stated, this experiment gives no information about the quality of the results when the image is scaled down.

The tests performed on the pictures indicate that the Hough analysis is a suitable application for online execution time analysis.

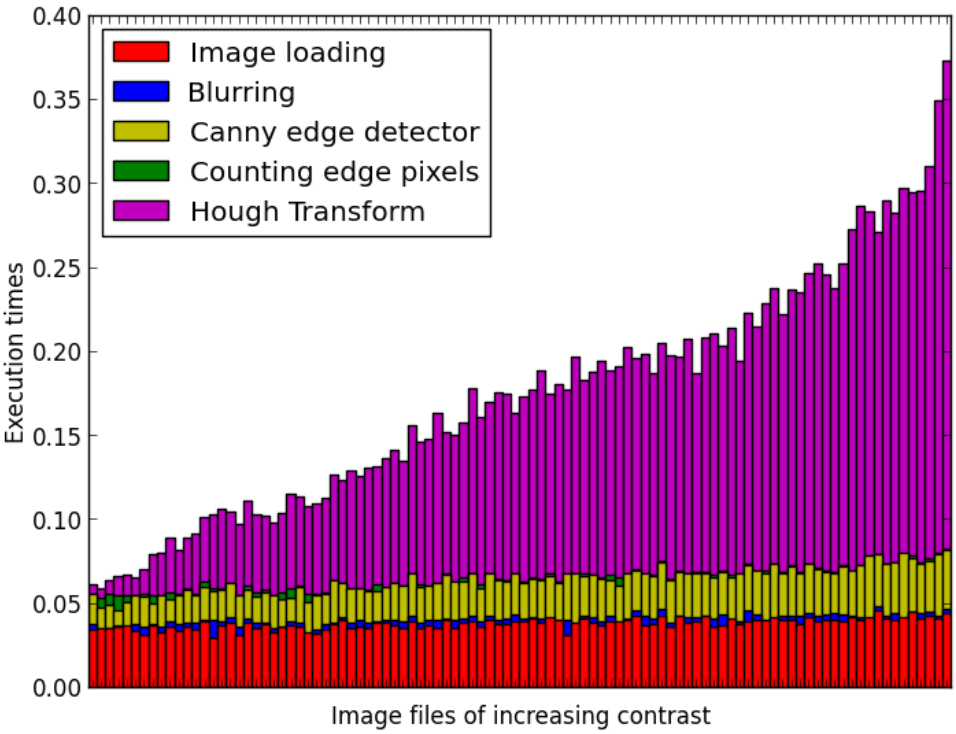


Figure 5: Average Execution time for different steps of the Hough transform. Time is given in seconds

4 Using Online WCET estimates for scheduling

This work presents a new scheduling scheme. Its prominent features include:

- Online WCET-analysis, which is run by the scheduler at every task release providing more precise timing information.
- Tasks consist of one or more task variants. If there is not enough time to run the preferred variant of a task, an alternative task variant can be run instead. This decision is made by the scheduler.
- Tasks have a criticality level associated with them to determine what tasks to prioritize over others.

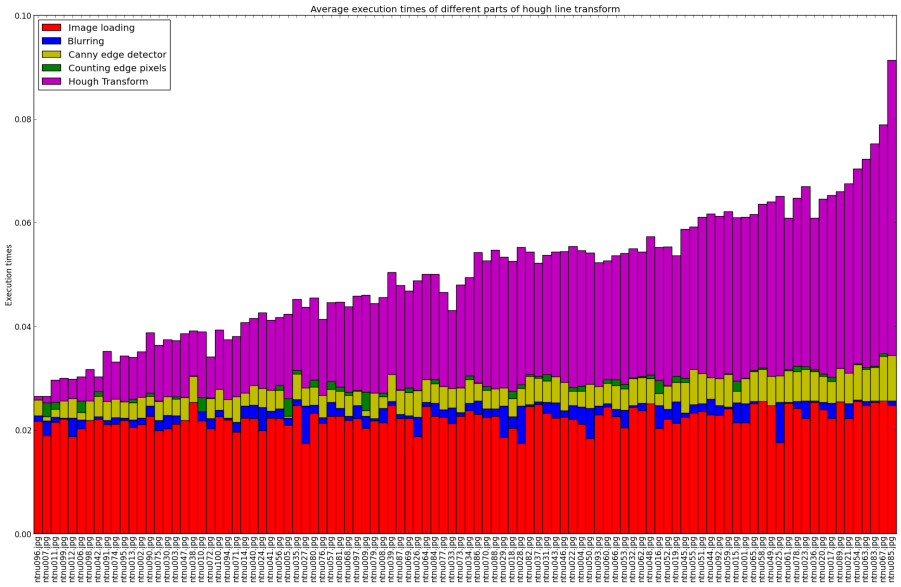


Figure 6: Average Execution time for different steps of the Hough transform with scaled down images. Time is given in seconds

It is important to note that the concept of using online worst case execution time analysis to help the scheduler is not restricted to the scheme described here. The following scheme is a somewhat simple example of how scheduling can be done, with suggestions for variations and changes in Chapter 4.4.

4.1 Model

The system consists of tasks. Each task has the following information associated with it:

- A priority: P
- A deadline: D
- A period: T
- A set of task variants
- An estimator preprocessing routine

- A criticality: L

A task variant has the following information associated with it:

- An entry point for the actual task
- An entry point for the OETA code

When referring to tasks and task variants, a number is used to label the task and a letter is used to label the task variant. These task variants are ordered alphabetically by preference. This means that if task 1 has the three task variants 1A 1B and 1C, 1A is the preferred and most time-consuming task variant. If this task variant cannot be used the task is "degraded" and the next task variant 1B is considered. The details surrounding this are given below.

Generally, a task has a single purpose that it needs to accomplish. This can be done in several ways, represented by the task variants.

An important property of the envisioned scheduling system is that it is the schedulers responsibility to make decisions regarding what task variant to use or whether a task is to be ran at all. The alternative to this is that the application programmer makes this functionality in the source code. This might allow for some more flexibility in the way the task is structured, but is more error prone. An analogy would be the use of a memory manager that ensures separation between processes in an operating system, instead of having all programmers reference the memory directly.

4.2 Choosing which task variant to use

The system may at any point degrade a task. Degrading a task can mean two things. The task will try to use the next task variant if it has any. If it is currently running the lowest task variant, it is aborted. This is shown in Figure 7.

When a task is released, its estimator preprocessing routine is run by the scheduler. This routine acquires and sets up all the necessary data for computing the WCET of a variant. The OETA algorithm for that tasks first task variant is run, and the result is saved. I the Hough transform example, the preprocessing would be everything up to the edge pixel counting. A test is performed to check

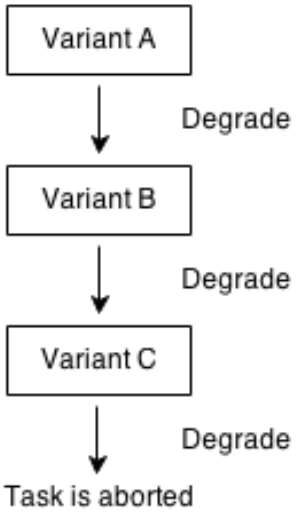


Figure 7: Degrading a task either causes it to switch to a lower task variant or abort it of it is already at the lowest variant.

whether the addition of the new tasks will result in a deadline miss in the system. If that is not the case, all tasks are allowed to run as normal. If the test finds that some task will miss its deadline, the scheduler aborts the lowest criticality task, and checks the WCET ON of that tasks next task variant. If no task will miss a deadline with that task variant, the task variant is used. If not, the next task variant is checked until either the system is schedulable, or the task has no more task variants. If it has no more task variants, the task is aborted and the scheduler runs the dadline check again. This time the second least critical task is degraded. This continues until the system is schedulable. This is illustrated in Figure 8. Note that in this example Task 1 is the least critical task in the system. Had another task been less critical, it would have been degraded in stead of task 1.

4.3 Testing for deadline misses

This is based on the system made in the preliminary project [5].

Testing for deadline misses is done with a simplified response time analysis. The general idea is that if the sum of $WCET^{ON}$ for all tasks with priority equal

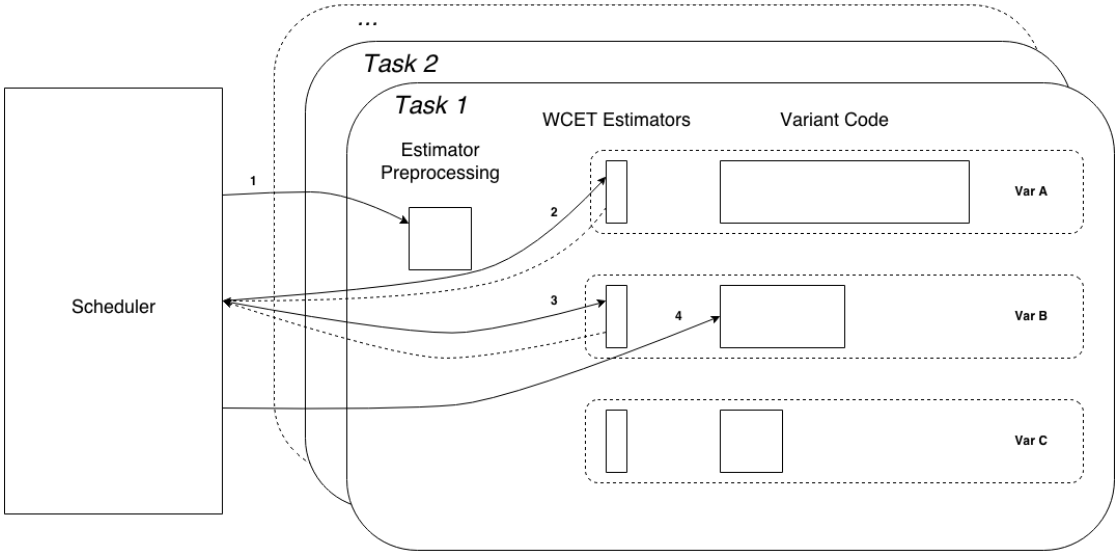


Figure 8: Deciding on a task variant. The released task is in this case the least critical task currently running. The process starts with the scheduler running the estimator preprocessing routine (1), then the WCET estimator for task variant A is run and its result returned to the scheduler (2). The scheduler deems it impossible to run task variant A, and calls the WCET estimator for task variant B (3). As the system is schedulable with the estimate for variant B, the scheduler starts running its actual code.

to or higher than a given priority is less than the time to the earliest deadline of any task of that priority, no tasks with that priority will miss a deadline. This test is then carried out for every priority level. The number of steps for this algorithm is the same as the number of tasks currently ready to run. This will run in $O(n)$ time, where n is the total number of tasks in the system. The WCET analysis is not run for every task each time, only for the task that was released. The other tasks $WCET^{ON}$ are saved by the scheduler from their release time. This can be written as

$$\sum_{\{i|p_i \geq p^*\}} WCET_i^{ON} < \min_{\{j|p_j = p^*\}} D_j \quad , \qquad \forall p* \in P \tag{2}$$

Where P is the set of priorities, and p^* is the priority level we are currently checking.

This is a simplified version of the response time analysis. The simplification is

that it does not take into account periodic tasks that will be released again (for example task 1 could be released again while task 2 is executing). This is because we treat every task as a sporadic task for the purposes of this test, and run the check for deadline misses at every task release.

This test is performed only on the tasks *that are currently ready to run*, not the entire task set of the system. In a system without time slicing, this test is sufficient, but not necessary. Further optimizations are likely possible, like considering the amount of time a task has already spent executing.

4.4 Variations

4.4.1 Assigning criticality levels to task variants instead of tasks

Normally, one considers criticality on a per-task basis. In other words, one task is given a higher criticality than another because we would like the airplanes landing gear to function correctly more than we would like to have good air condition inside the plane. This model, however, specifies a criticality level for each task variant. This is because it may be desirable to degrade a higher criticality task to a lower task variant instead of completely aborting a lower criticality task. Consider the example task set in Figure 9, consisting of tasks 1, 2 and 3, each with its set of task variants (A and B for task 1 and 3, with a third variant C for task 2).

Assume that task 1 and 2 is running, both at variant A, and task 3 is released. The scheduler finds that the system is not schedulable and must degrade one task. Task 1 is the most critical of the three, followed by task 2 with task 3 being the least critical task. As such task 3 is degraded and forced to execute its B-variant. The scheduler checks to see if the new task set is schedulable. If that is not the case a new task must be degraded. If criticality is assigned to each task, the scheduler will degrade task 3, thereby aborting it. It may be desirable to instead degrade task 2 to its variant B in order to let task 3 execute at all. This can be ensured by assigning criticality-values to task variants instead of tasks. If it is desirable to operate with a criticality-level per task, this can be ensured by giving all task variants of the same task the same criticality-level.

Another possibility would be to consider more permutations of tasks. Given the example above, if we decide to degrade task 2 instead of aborting task 3, the

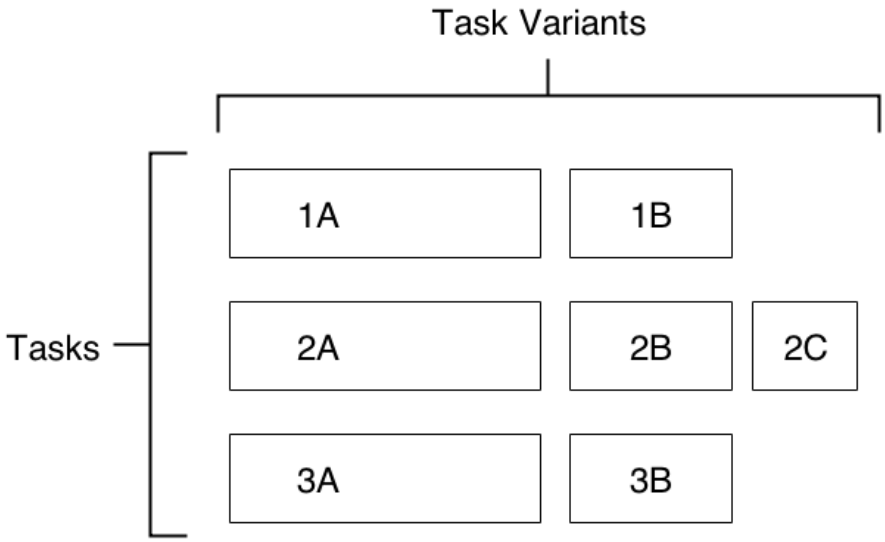


Figure 9: An example task set. It shows the three tasks labeled 1, 2 and 3, as well as their respective set of task variants.

current task set would be (1A, 2B, 3B). Since the decision of degrading task 3 to variant C was taken when task 2 was running as variant A, the system could now potentially run task 3 with variant A. This was proposed in the feedback on the paper submitted to The 2014 WCET Workshop. The full feedback is given in Appendix D. This would increase the complexity of the schedulers decision.

4.4.2 Changing criticality levels

In some applications it might be advantageous to allow for task to change criticality level based on the system state or circumstances of the application. In a UAV flying by GPS-waypoints and autopilot would probably want to treat these features as highly critical, but when landing, the elevation sensor and remote control communication would be more critical.

4.4.3 Using a value function

A third variant is the use a value function to determine the task to be degraded. This would compare all the possible sets of variants for the currently running tasks that are schedulable, and compare them based on a value function describing how

valuable each task is to the system at any given time. The scheduling would then be performed by finding the set of task variants that maximizes the value function.

This function can take into account factors like how long a task has been executing, how much time would be freed by degrading the task in addition to its criticality. The function could also use information about the past behavior or, in the case of periodic tasks, future task releases. This would allow it to prioritize tasks that have recently been degraded. This can be applicable in situations where two tasks are of almost equal importance. Normally, the system should degrade the slightly less critical task to run the other, but if this has happened three times in a row, it might be sensible to degrade the high criticality task instead. Information about future task releases can be used in a similar way.

There are several challenges with such a scheme. First and foremost making such a value function is not trivial. The value function does not need to take into account all the factors suggested above, but even simple value functions would be difficult to construct, as one would have to not only decide which tasks are more or less critical, but how high the value of each task or task variant is compared to each other. Another large problem would be to formally verify the correctness of such a system, and ensuring that the value function actually chooses a desirable course of action in any set of circumstances. It may be that the value function deems the value of running ten virtually unimportant tasks higher than the value of running one life critical task. This kind of behavior is difficult to predict and requires rigorous testing and analysis.

5 Experiment

A simulation was conducted to demonstrate the potential benefit of using online WCET analysis for scheduling.

5.1 Assumptions and simplifications

Assumptions:

Tasks are not aborted after they exceed their deadline

The OETA overhead is constant for all tasks While it is unrealistic that the OETA overhead is absolutely constant (and equal for all tasks), it should not vary greatly. The timing results of the Hough transform shown in Figure 5 show this behavior.

Every task can have its execution time predicted and the predictions are accurate In reality the online WCET estimation will predict an execution time that is larger than the actual execution time. In this case the difference in estimated and actual execution time has been disregarded.

Simplifications:

No task switching overhead Since the normal task switching is the same both with and without OETA, it has not been included in the simulation. The OETA overhead includes both the estimation algorithm itself and the time used to check schedulability and pick a suitable task variant.

Criticality is not considered Adding the criticality level of tasks to the experiment would not provide any interesting information as the scheduler would always degrade the least critical task. There is no measure of the value of the task variants executed.

5.2 Method

A task set consisting of five tasks has been given random execution times and deadlines. Variations in execution time between runs due to different input were

simulated by randomly doubling or tripling the execution time of any task. The task set was run by a simulated standard deadline monotonic scheduler and a simulation of our system that added a constant overhead to represent the online WCET analysis, and also allowed a task to spend only half or 1/4 its execution time if needed to represent an alternative task being run. It is assumed here that the scheduler runs the online WCET analysis for all tasks before starting the first task. The simulation was done for 10000 task sets.

The standard parameters for the simulation are given in Table 1.

Table 1: Standard parameters used in simulation. The doubling and tripling are mutually exclusive: only one will happen for each task

Number of tasks	5
Number of task sets generated	10000
Number of runs for each task set	100
Execution time	Uniformly distributed between 1 and 10
Deadline	Uniformly distributed between 5 and 50
Probability of doubling execution time	9%
Probability of tripling execution time	1%
OETA overhead	0.1

Several variations of the standard values were used to test how the OETA scheduling would perform:

- Standard parameters
- OETA overhead set to 1 instead of 0.1
- OETA overhead set to 1.5 instead of 0.1
- Shorter deadlines

The execution time is a relative time and the deadline is an absolute time. In the example task set given in Table 2, tasks 2, 3 and 5 would be able to finish by their deadline, while the rest would miss it. How the tasks will run is illustrated in Figure 10.

The source code for the simulation is given in Appendix A.

Table 2: Example task set

Task Number	Execution Time	Deadline
1	9	5
2	2	15
3	9	24
4	7	26
5	3	39

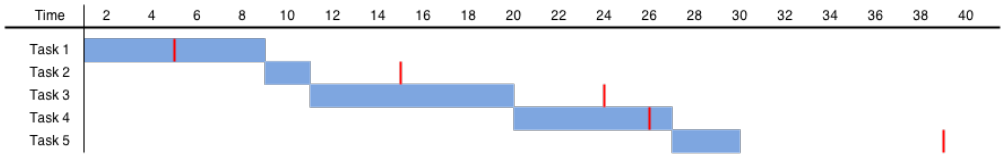


Figure 10: A diagram showing the execution times (blue areas) and deadlines (red lines) of the task set example described in Table 2.

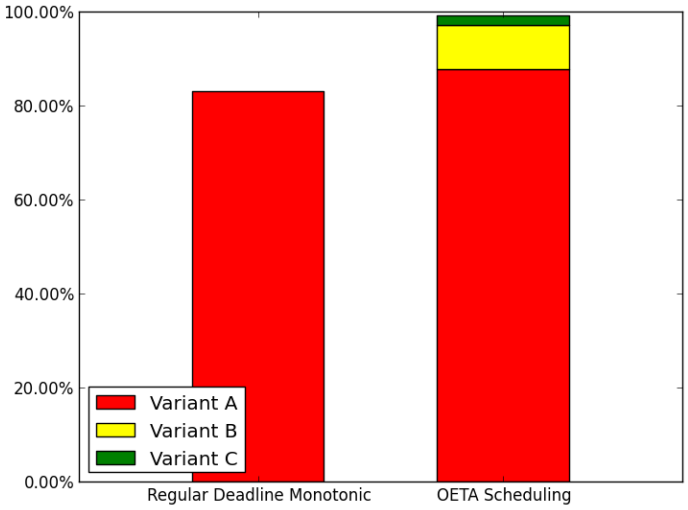
5.3 Results

As the values of execution times and deadlines as well as the behavior of the tasks are somewhat arbitrary, the qualitative observations are more interesting than the quantitative. The results are therefore only presented as bar charts. The actual numbers are given in Appendix B.

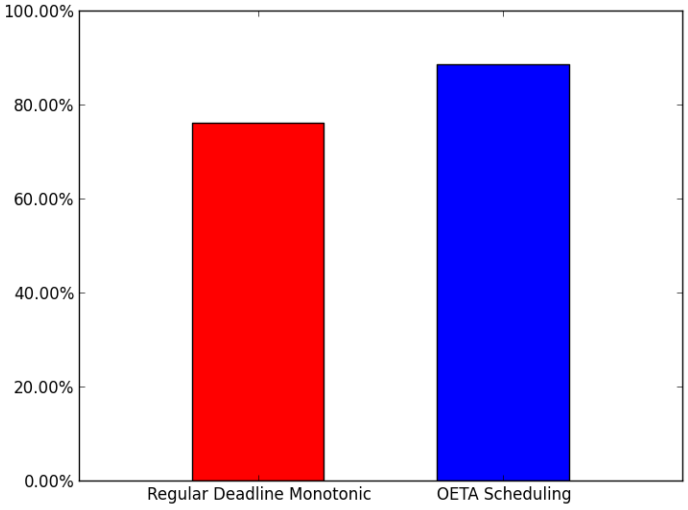
Two metrics were measured:

- The number of tasks that finished within their deadline: N_{fin}
- The amount of time spent performing tasks that finished before their deadline T_{fin}

5.3.1 Resuslts with standard parameters



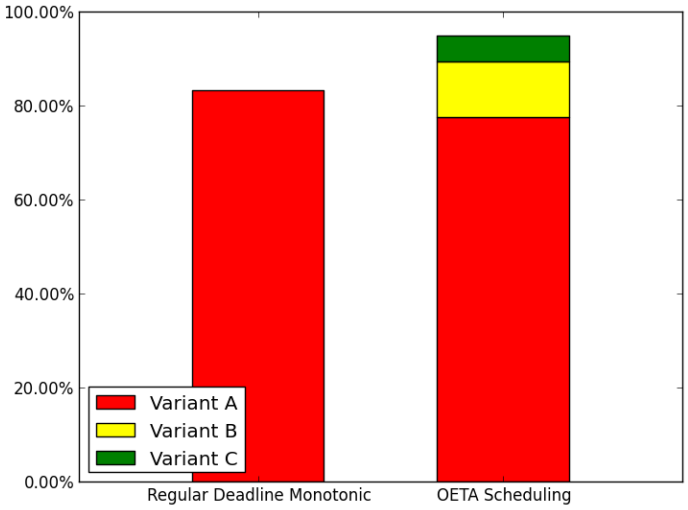
(a) Percentage of tasks that finished by their deadline



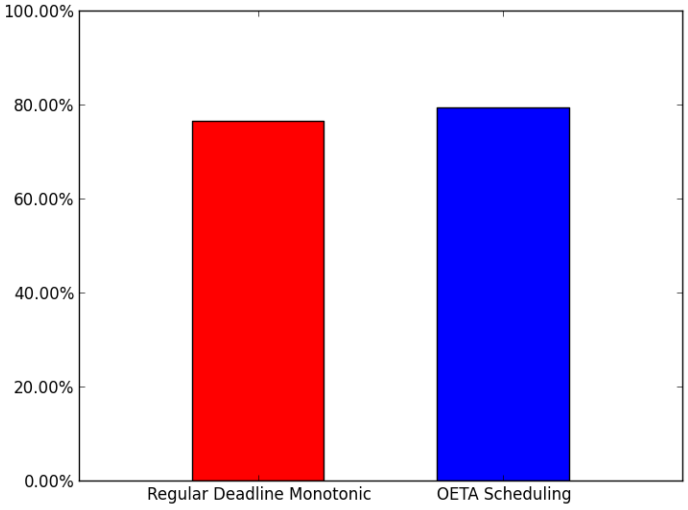
(b) Percentage of time spent executing tasks that finished by their deadline

Figure 11: Simulation results with standard parameters

5.3.2 Results with increased estimation overhead

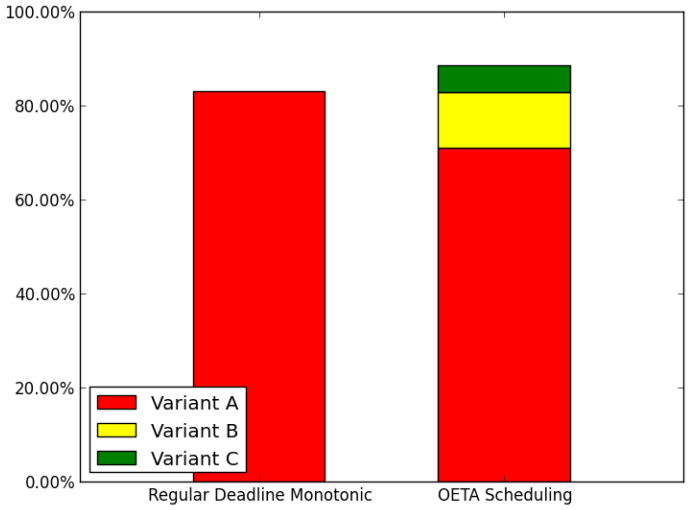


(a) Percentage of tasks that finished by their deadline

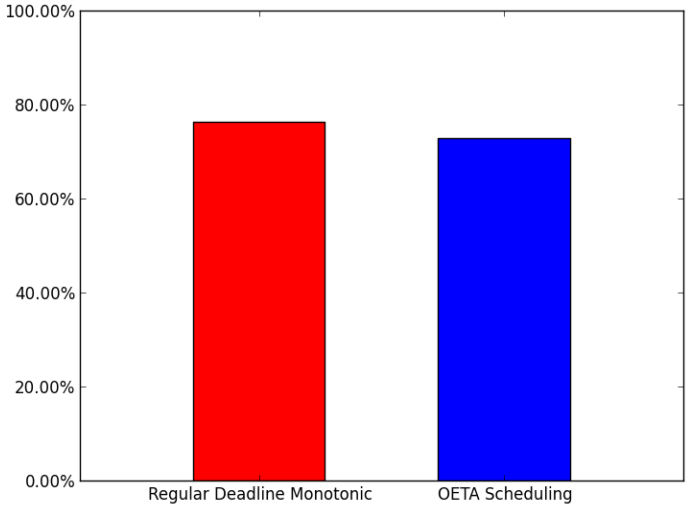


(b) Percentage of time spent executing tasks that finished by their deadline

Figure 12: Simulation results with OETA overhead set to 1



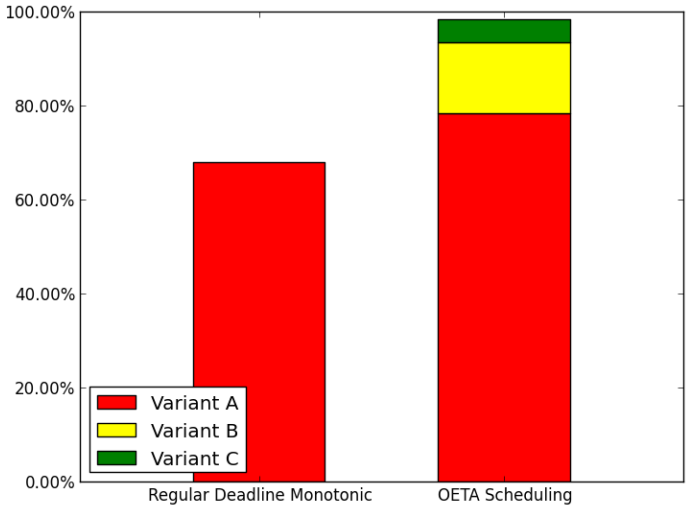
(a) Percentage of tasks that finished by their deadline



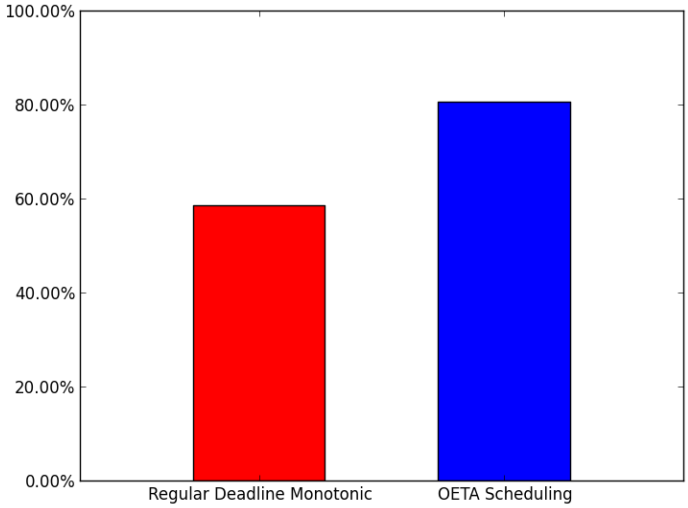
(b) Percentage of time spent executing tasks that finished by their deadline

Figure 13: Simulation results with OETA overhead set to 1.5

5.3.3 Results with tighter deadlines



(a) Percentage of tasks that finished by their deadline



(b) Percentage of time spent executing tasks that finished by their deadline

Figure 14: Simulation results with earlier deadlines

5.4 Discussion

Figure 11a shows how many tasks were able to finish by their deadline. We can observe two things from it. Firstly, the fact that more tasks were able to finish by their deadline using OETA Scheduling. This is to be expected, since a task can be swapped out with a shorter variant if there is not enough time to run it. Secondly, we can see that with OETA Scheduling more tasks of Variant A were able to complete before their deadline. Since it completes some tasks faster when needed, there is more time available in the system. Figure 11b shows the amount of time that was not wasted by executing tasks that did not meet their deadline. Since it can switch to shorter tasks, we can see that OETA scheduling is outperforming the regular deadline monotonic scheme.

When the standard results are compared with the results in Figure 13, it is clear that increasing the OETA overhead from 1-10% of the execution time to 10-100% reduces the number of variant A tasks that can be scheduled by the OETA scheduler, see Figure 12a. Nevertheless, the total number of task that complete before their deadline is higher. Figure 12b shows that the T_{fin}-value of the OETA scheduling has decreased and is now about the same level as for the regular deadline monotonic scheme. This decrease is expected as the scheduler spends more time performing the OETA algorithm than with the standard parameters. One of the primary concerns with OETA is that it will introduce an extra overhead that outweigh any benefit it might offer. This results indicate that even with fairly large OETA overhead, the OETA scheduler with alternative tasks still performs better than traditional deadline monotonic scheduling. This is attributed both to the possibility of running an alternative task, but it is also a result of the online execution time scheduling itself, since it will not run tasks that it knows will miss their deadline, thereby saving execution time for other tasks.

The trends of increasing the OETA overhead continues when it is increased to 1.5, as shown in Figure 13. An OETA overhead of 1.5 is in this case between 15% and 150% of the actual execution time of the task and as such is unrealistically high. One interesting observation to make here is that even though the T_{fin} of the OETA scheduling is lower than for regular deadline monotonic scheduling, it still manages to complete more tasks by their deadline. This indicates that comparing

CPU utilization alone is not a good metric for quality of scheduling policies with alternative tasks.

Lastly a test was performed with earlier deadlines. Figure 14 shows that the OETA scheduling scheme is less sensitive to this than regular deadline monotonic scheduling. Although the number of variant A tasks that are able to complete is smaller the total number of tasks completing by their deadline remains virtually unchanged.

One important factor not addressed in this experiment is the quality of service provided by the different task variants. We have assumed that all tasks can be degraded to more efficient task variants that provide an acceptable quality of service. This might not be the case for all applications.

6 Implementation of scheduling simulator

This section describes a scheduling simulator for testing OETA scheduling. As described in Section 4.4, there are several ways to perform the scheduling while taking advantage of online execution time analysis. The variations and extensions described offers more flexibility, like the option of degrading a highly critical task instead of aborting a less critical task if this is favorable. The cost of this flexibility is higher computational overhead. Since the relative advantages and disadvantages of these scheduling systems are hard to predict, a framework for testing should be implemented. The purposes of this framework is both to be able to test different scheduling schemes and applications suitable for OETA scheduling. The design described has not been completely implemented.

6.1 Design goals

The goals of the implementation are:

It should simulate a preemptive scheduler. The scheduler can interrupt the execution of a task at any time to run another task. It can also resume execution where it left off.

Tasks can run arbitrary programs. The application programmer should be able to run any code as a task.

Scheduler can run each tasks WCET estimator. The application programmer can specify the OETA routine that computes a tasks execution time. This can be run by the scheduler.

Tasks can have alternative tasks associated with them. A task can have a set of alternative tasks that the scheduler can switch between when required.

Separation between the scheduler and application. The application programmer should not need to worry about task preemption/resuming, computing WCET or switching of alternative tasks.

It should be easy to change the scheduling scheme. Since part of the purpose of the scheduler simulator is to make it possible to experiment with

different ways of scheduling, changing the scheduling scheme should be possible without affecting other parts of the simulator.

6.2 Implementation details

The scheduling simulator will be implemented as a normal program running on a standard Linux computer. In a preliminary project, the author made a similar testing platform by rewriting parts if the FreeRTOS kernel, to allow for OETA code to be run. This was run on an Atmel micro controller development kit. The reasons for not going with this design in this thesis were:

Limitations in the code that can be run Computer vision applications are not able to run on in this system because of hardware and the file management of FreeRTOS.

Added complexity of FreeRTOS Since it is a version of FreeRTOS, changing the scheduling policy of the system is not trivial.

Because of the authors familiarity with the FreeRTOS kernel, the implemented scheduler follows a similar design to that of FreeRTOS.

It was decided to implement the scheduler in C, as this is the language used for most schedulers and operating systems. It was believed that the system would be easier to implement the simulator as a real scheduler in a real system later. In retrospect, it would be better to use a more feature-rich language like C++, that allows for functionality like exception handling and polymorphism, as implementing the system to run as a real scheduler would require a lot of rewriting anyway.

The tasks are implemented as processes that are forked from the scheduler process when they are started. This was chosen over the use of pthread because processes can be suspended and resumed by sending signals (SIGSUSP and SIGCONT). There is, to the authors knowledge, no way to halt execution and be able to resume exactly where it left off by using pthreads. Using processes instead of threads requires special attention to be given to any data that needs to be shared between the task and the scheduler. This is handled by allocating shared memory in the initiation function for the task.

6.3 Design

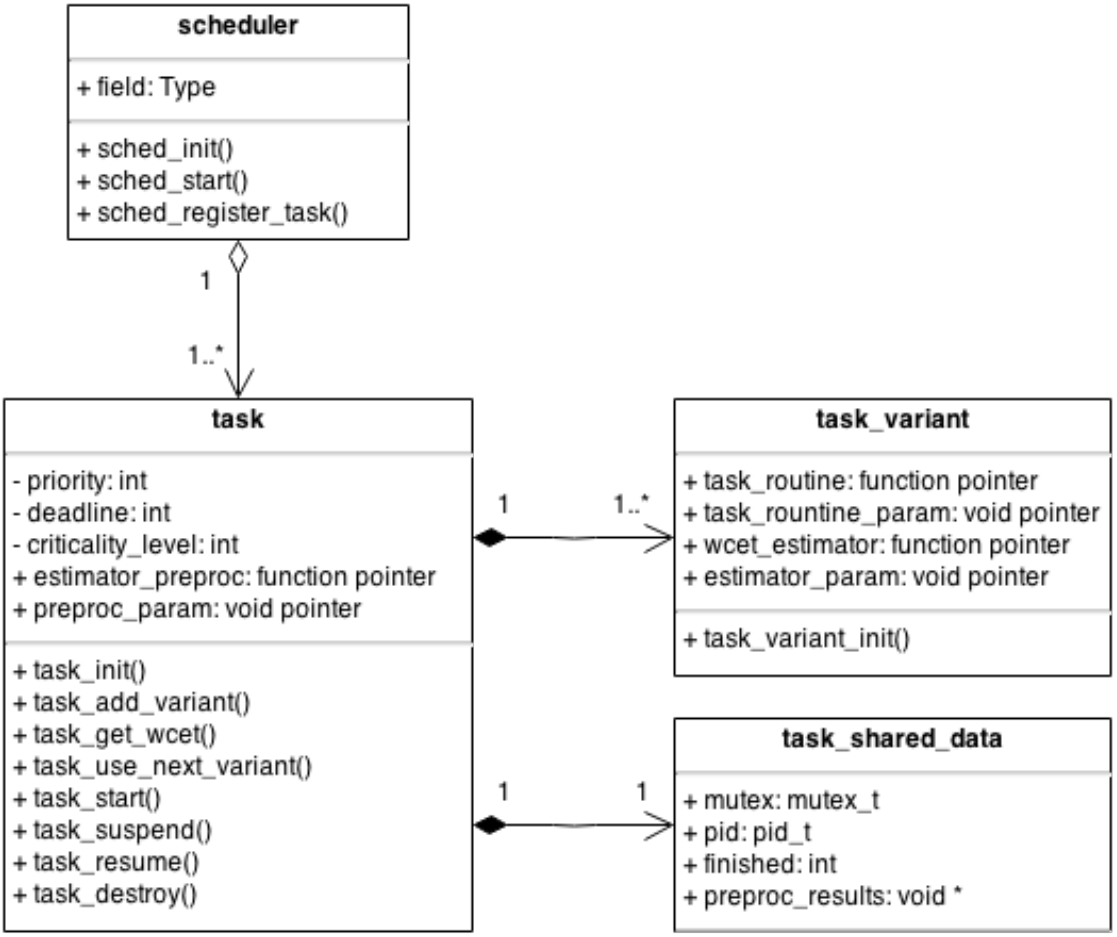


Figure 15: Class diagram

Figure 15 shows a class diagram with the components of the scheduling simulator. The Scheduler keeps a list of Task objects. Each task object has a list of task variants that contain a task routine and its estimation function. The shared object contains data that is put into shared memory. This is necessary because the tasks are created by spawning new processes, and this information needs to be accessible by both the scheduler process and task process.

Note that there is a problem with the current design as a void pointer in shared memory will not reference the same data in both processes. Even if the memory is allocated before the fork, the processes will not access and manipulate the same

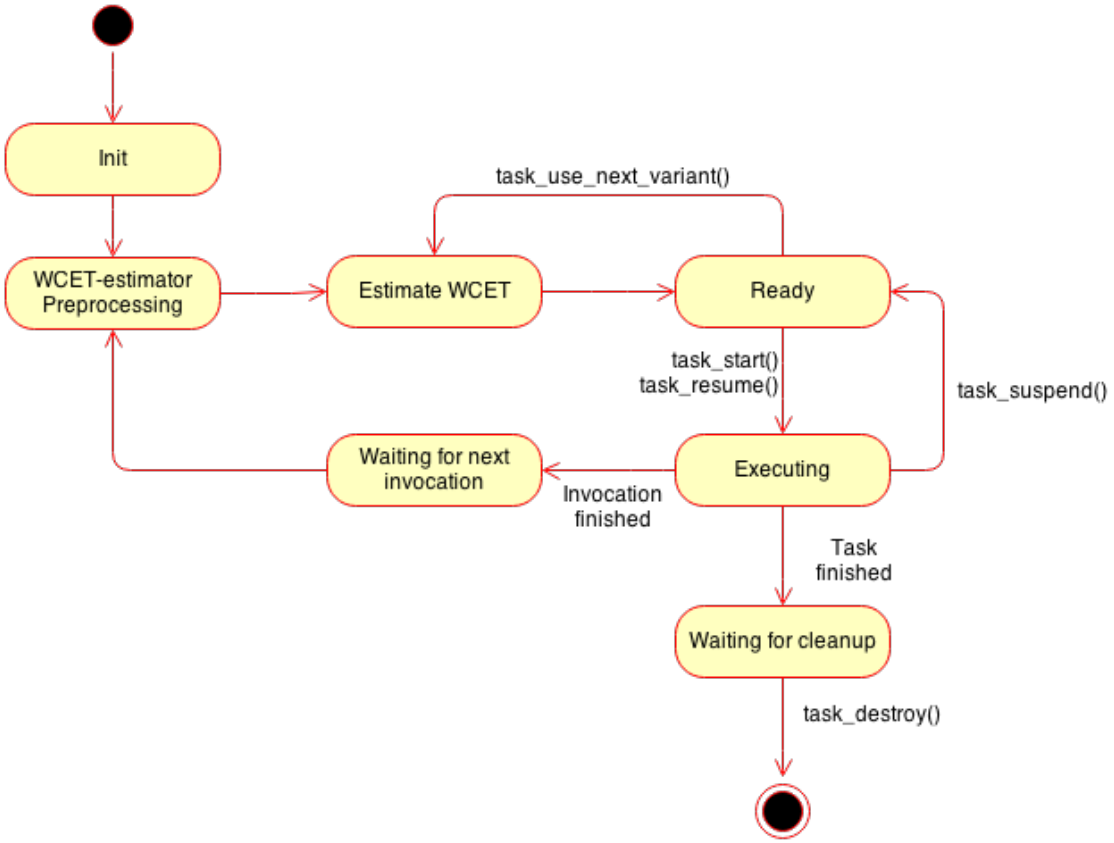


Figure 16: State diagram for a task

data. One solution here would be to allocate enough space for the kind of data the preprocessing estimation produces in the Shared Data . In the example of the Hough line transform, the preprocessing produces both the binary image from the Canny edge detector and the number of edge pixels. The scheduler class shows the interface between the scheduler and the application. This is explained further in section 6.4.

Figure 16 shows the state diagram for a task. This behavior is controlled by the scheduler.

6.4 Task programming interface

The relevant functions to an application programmer are:

- task_init()

- task_variant_init()
- task_add_variant()
- sched_init()
- sched_register_task()
- sched_start()

The seemingly redundant functions `task_init()` and `sched_regster_task()` are necessary to keep the scheduler separated from the task implementation. This is done to make it possible to modify the scheduler independently of the task implementation.

6.5 What has been implemented

At the time of this writing the basic task functionality has been implemented. This include starting, suspending and resuming a task, as well as changing between task variants. There is an issue with the communication between the estimator preprocessing and the estimation function itself.

A scheduler like the one described in section 4 has been partially written.

In addition to this a simple linked list implementation has been made. This is used by the tasks to store the task variants and by the scheduler to store the tasks.

The code is given in the digital attachment described in Appendix E. Along with the code, there are some simple module tests that demonstrate the functionality of these modules.

7 Conclusion and discussion

One of the most challenging parts of this work has, somewhat surprisingly, been to find a good application that illustrates the benefit of online execution time analysis. When discussing the subject generally it seems intuitively apparent that as real time systems grows in complexity, tasks become increasingly dynamic, and this development would call for the use of a system like the online WCET analysis. However, it proved difficult to find an application where the execution time is dynamic from one task invocation to the next and was also used in systems where time constraints are of high priority. Finally, computer vision was found to be a very suitable example.

The difficulty of finding suitable applications for online WCET analysis could be seen as an indication that the concept itself is not as universally applicable as initially believed. Another explanation is that finding general examples is a difficult angle to look at the problem from. A better, and possibly more productive, way might be to look at a specific system such as a UAV and analyze the tasks it consists of. When one has the source code it might be easier to make a good estimation algorithm for that particular task. This is the way it will be done by the programmer if this system is used in a real application.

It might also be possible that cybernetics based real time systems (control systems etc.) are not an ideal area to use as an example for this kind of system. These systems are usually required to perform equally well all the time, and not as good as the rest of the system allows. With this type of application one will in many situations simply buy a more powerful processor to make sure that it can handle all variations in task execution time, rather than introduce a system with online WCET analysis. Other areas, like multimedia may have a larger number of relevant applications. On the other hand, the trend of moving tasks of different levels of criticality onto the same hardware, can make it necessary to consider these areas coexisting in the same system.

One issue of using the OETA system is the added complexity it introduces. In addition to making the application code, the programmer must also make the estimator code for each task. More code in a project increases the risk of errors as well as the need for testing. The development process will be a lot more time

consuming as the online estimation code must be designed, written and tested. In addition, this process must be repeated for every change to the tasks actual code. It has been briefly mentioned that the static formal verification of a OETA systems might be a challenge as well.

Simulations on the Hough line transform showed that the execution time of such an algorithm followed a relatively linear relation to the complexity of the image, measured in the number of edge pixels. This result proved that the online WCET analysis can be used in a practical setting. One possible source of worry that has been brought up with online WCET analysis is the effect of features like caching and branch predictors that improve the average execution time of programs, but makes it less deterministic. Figure 3 shows the execution times of the Hough transform performed on several pictures on an Intel i5 processor running Linux. The linearity of the plot shows that even with cache and other optimizing technologies online WCET analysis can give useful results.

Designing a run-time system that could utilize the online WCET analysis was less difficult. Several properties of real time systems can be improved with online WCET analysis, such as the number of tasks that can be run or the utilization of mixed criticality systems [5].

The simulations that have been run indicate that the use of OETA for scheduling performs better than traditional deadline monotonic scheduling. Even though the simulations are greatly simplified compared to any real implementation, some observations can be made. The most important is that the overhead introduced by estimating the WCET does not necessarily need to be as small as initially believed. When discussing the concept with my supervisor we agreed that a estimation time of 1-2% of the tasks execution time should be the standard. The simulations indicate that an estimation time of up to 10% of the task execution time is not necessarily problematic.

A paper has been written and submitted to the 2014 WCET Workshop hosted by TACLe. It is given in Appendix C. The paper was rejected, mainly because it was poorly written and failed to give a good presentation of the concept (see the reviews in Appendix D). Some time later, however, the author was invited to present this work at a meeting held by the COST Action TACLe. This interest in online WCET analysis from an international community of researchers suggests

that the concept has merits.

The work done in this project indicates that the idea of online WCET analysis is a concept worth further study. Both simulations and response from international research communities has shown that there are benefits to incorporating such functionality in a real time system.

Finally some suggestions for future work: When the simulation framework is finished it should facilitate further and more precise experimentation with the pros and cons of OETA scheduling. Especially the added time overhead should be compared to the potential benefits. A good experiment would be to take a full scale real time system and analyze all the different tasks, to see how many are suited for online WCET estimation.

References

[1] S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Proceedings of the 2011 IEEE 32Nd Real-Time Systems Symposium*, RTSS '11, pages 34–43, Washington, DC, USA, 2011. IEEE Computer Society.

[2] Alan Burns and Bob Davis. Mixed criticality systems - a review, 2013.

[3] Alan Burns and Andy Wellings. *Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX*. Addison-Wesley Educational Publishers Inc, USA, 4th edition, 2009.

[4] Pascual Campoy, JuanF. Correa, Ivan Mondragón, Carol Martínez, Miguel Olivares, Luis Mejías, and Jorge Artieda. Computer vision onboard uavs for civilian tasks. *Journal of Intelligent and Robotic Systems*, 54(1-3):105–135, 2009.

[5] Fredrik Bakkevig Haugli. Improving mixed-criticality real time systems by the use of online execution time analysis, 2013.

[6] Sverre Hendseth and Giorgio Buttazzo. Exploiting online WCET estimates. In *RTSOPS 2012 Final Proceedings*, pages 9–10, 2012.

[7] C.M. Krishna and K.G. Shin. *Real-Time Systems*. Computer engineering. McGraw-Hill Education, 1997.

[8] Zhengrong Li, Yuee Liu, Rodney Walker, Ross Hayward, and Jinglan Zhang. Towards automatic power line detection for a uav surveillance system using pulse coupled neural filter and an improved hough transform. *Mach. Vision Appl.*, 21(5):677–686, August 2010.

[9] Sibin Mohan, Frank Mueller, William Hawkins, Michael Root, Christopher Healy, and David Whalley. Parascale: Exploiting parametric timing analysis for real-time schedulers and dynamic voltage scaling. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 233–242, 2005.

[10] J. A. Stankovic and K. Ramamritham. The spring kernel: A new paradigm for real-time operating systems. *SIGOPS Oper. Syst. Rev.*, 23(3):54–71, July 1989.

[11] Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, RTSS '07, pages 239–243, Washington, DC, USA, 2007. IEEE Computer Society.

[12] Tyan Vladimir, Dongwoon Jeon, Doo-Hyun Kim, Chun-Hyon Chang, and Jung-Guk Kim. Experimental feasibility analysis of roi-based hough transform for real-time line tracking in auto-landing of uav. In *ISORC Workshops*, pages 130–135. IEEE, 2012.

Appendices

A WCET Scheduling simulation code

This is the python-code used to perform the simulation described in Section 5.

```
1
2  import random
3  import numpy as np
4  import matplotlib.pyplot as plt
5  from matplotlib.ticker import FuncFormatter
6
7  def percent_format(x, pos=0):
8      return '%1.2f%%'%(100*x)
9
10 n_tasks = 5
11
12
13 dm_completed_tasks = 0
14
15 var_a_completed_tasks = 0
16 var_b_completed_tasks = 0
17 var_c_completed_tasks = 0
18
19 total_time = 0
20 dm_time = 0
21 oeta_time = 0
22
23
24 for i in range(10000):
25
26     # Generate task set
27     deadlines = []
28     exec_times = []
29     oeta_overhead = []
30     for task in range(n_tasks):
31         deadlines.append(random.uniform(5,50))
32         exec_times.append(random.uniform(1,10))
33         oeta_overhead.append(0.1)
```

```
34      deadlines.sort()
35
36      for j in range(100):
37
38          # Set execution time for current iteration
39          iteration_exec_time = exec_times[:]
40          for task in range(n_tasks):
41              random_number = random.random()
42              if random_number >= 0.99:
43                  iteration_exec_time[task] *= 3
44              elif random_number >= 0.9:
45                  iteration_exec_time[task] *= 2
46
47          # Run regular deadline monotonic
48          execution_time_sum = 0
49          for task in range(n_tasks):
50              total_time += iteration_exec_time[task]
51              execution_time_sum += iteration_exec_time[task]
52              if execution_time_sum < deadlines[task]:
53                  dm_completed_tasks += 1
54                  dm_time += iteration_exec_time[task]
55
56
57          # Run OETA scheduler
58          execution_time_sum = 0
59          for task in range(n_tasks):
60              execution_time_sum += oeta_overhead[task]
61          for task in range(n_tasks):
62              execution_time_sum += iteration_exec_time[task]
63              if execution_time_sum < deadlines[task]:
64                  var_a_completed_tasks += 1
65                  oeta_time += iteration_exec_time[task]
66              else:
67                  execution_time_sum -= (iteration_exec_time[task
                        ]/2)
68                  if execution_time_sum < deadlines[task]:
69                      var_b_completed_tasks += 1
70                      oeta_time += iteration_exec_time[task]/2
71                  else:
```

```
72                      execution_time_sum -= (iteration_exec_time[
                            task]/4)
73                      if execution_time_sum < deadlines[task]:
74                          var_c_completed_tasks += 1
75                          oeta_time += iteration_exec_time[task
                                ]/4
76
77
78
79 print "Deadline Monotonic: \t", dm_completed_tasks
80 print "Var A completed: \t", var_a_completed_tasks
81 print "Var B completed: \t", var_b_completed_tasks
82 print "Var C completed: \t", var_c_completed_tasks
83
84 width = 0.35
85
86 plt.figure(1)
87 p1 = plt.bar(0.3, dm_completed_tasks/5000000.0, width, color='
       red')
88 p2 = plt.bar(0.95, var_a_completed_tasks/5000000.0, width,
       color='red')
89 p3 = plt.bar(0.95, var_b_completed_tasks/5000000.0, width,
       color='yellow', bottom=(var_a_completed_tasks/5000000.0))
90 p4 = plt.bar(0.95, var_c_completed_tasks/5000000.0, width,
       color='green', bottom=((var_b_completed_tasks+
       var_a_completed_tasks)/5000000.0))
91
92 plt.legend(( p2[0], p3[0], p4[0]), ( 'Variant A', 'Variant B',
       'Variant C', ), loc=3)
93
94 plt.xticks([0.3+0.35/2, 2*0.3+0.35*1.5], ('Regular Deadline
       Monotonic', 'OETA Scheduling'))
95 plt.xlim(0,1.6)
96 plt.ylim(0,1)
97
98 plt.gca().yaxis.set_major_formatter(FuncFormatter(
       percent_format))
99 plt.figure(2)
100
```

```
101 plt.gca().yaxis.set_major_formatter(FuncFormatter(
        percent_format))
102
103 p1 = plt.bar(0.3, dm_time/total_time, width, color='red')
104 p2 = plt.bar(0.95, oeta_time/total_time, width, color='blue')
105
106 plt.xticks([0.3+0.35/2, 2*0.3+0.35*1.5], ('Regular Deadline
        Monotonic', 'OETA Scheduling'))
107 plt.xlim(0,1.6)
108 plt.ylim(0,1)
109
110 plt.gca().yaxis.set_major_formatter(FuncFormatter(
        percent_format))
111 plt.show()
```

B Experiment results

Standard variables: Percentage of tasks finished Deadline Monotonic: 0.8303756 Var A completed: 0.8772344 Var B completed: 0.0933094 Var C completed: 0.0213072

Time spent executing tasks that did not miss deadline Deadline Monotonic: 0.760636816703 OETA Scheduling: 0.88606699438

Overhead set to 1: Percentage of tasks finished Deadline Monotonic: 0.832656 Var A completed: 0.7752408 Var B completed: 0.1184746 Var C completed: 0.0557684

Time spent executing tasks that did not miss deadline Deadline Monotonic: 0.764510057549 OETA Scheduling: 0.794146834257

Overhead set to 1.5: Percentage of tasks finished Deadline Monotonic: 0.8308696 Var A completed: 0.7087512 Var B completed: 0.1197086 Var C completed: 0.056385

Time spent executing tasks that did not miss deadline Deadline Monotonic: 0.76196807125 OETA Scheduling: 0.729069195343

Shorter deadline: Percentage of tasks finished Deadline Monotonic: 0.6797032 Var A completed: 0.7821982 Var B completed: 0.1520732 Var C completed: 0.0489536

Time spent executing tasks that did not miss deadline Deadline Monotonic: 0.585088262426 OETA Scheduling: 0.805011664561

C The paper submitted to the WCET Workshop

Below is the paper that was written and submitted to the 2014 WCET Workshop in Madrid, Spain.

Using online WCET analysis to schedule alternative tasks in mixed criticality systems

Fredrik Bakkevig Haugli, Amund Skavhaug, and Sverre Hendseth

Department of Engineering Cybernetics - ITK
Norwegian University of Science and Technology - NTNU
Trondheim, Norway
{freha@stud, amund.skavhaug@itk, sverre.hendseth@itk}.ntnu.no

Abstract

It is well known that traditional offline WCET estimates are inherently pessimistic, leading to non-optimal design. Since most tasks have a varying execution time, planning for all tasks to use their offline computed WCET leads to the CPU being idle a large portion of the time. As not all tasks are hard real time tasks, different strategies has been used to make sure critical tasks meet their deadline, and non-critical tasks are executed on a best-effort basis. Such strategies include period transformation, requiring the application designer to split the code into several tasks or in other ways contort the task code. More recent work in mixed criticality scheduling[8, 1], lets the system designer assign criticality levels to tasks and lets the scheduler allow less critical tasks to be run as long as they do not cause a more critical task to miss a deadline.

We propose a system that uses information available online to get a better WCET estimate by running an estimator routine at each task release. This information allows the scheduler to decide if a task can be run without causing a more critical task to miss its deadline. If there is not sufficient time to do so, an alternative task may be run instead that performs the same service with lower execution time, but also with diminished quality. Resorting to alternative tasks has previously been a choice made by the application without any knowledge about the state of the system in general. Since the decision in our system is made by a scheduler with a more precise online WCET estimate for all running tasks, our system can make better decisions, as well as make the application programmers job easier. Additionally it is a bandwidth-preserving scheme, as the decision about running an alternative task can be done by the scheduler before the task starts executing, instead of having the task abort halfway through execution and thereby wasting CPU time.

A simulation has been conducted on a hypothetical task set. It showed that more tasks were able to meet their deadline in a system using online execution time analysis and alternative tasks.

Keywords and phrases Online WCET analysis, alternative tasks, mixed criticality

1 Introduction

In most real time systems not all tasks are equally important. Some tasks deal with preventing catastrophic failure or death and are very critical. Other tasks, such as data logging, might be less critical. They should be executed, but not at the expense of the high criticality tasks. This has traditionally been ensured in different ways. One way is to give tasks with high criticality a high priority, thereby ensuring that less critical tasks will be preempted in order to run the more critical ones. This can be done either by explicitly giving the task a high priority, or implicitly through for example period transformation. This method divides a highly critical task is into several tasks with shorter period in a system with a rate monotonic priority assignment scheme, thereby giving it a high priority.

These methods have some disadvantages. Firstly, a tasks temporal properties and importance are being combined into one attribute. Secondly, they might force the application

Conference/workshop/symposium title on which this volume is based on.
Editors: Billy Editor, Bill Editors; pp. 1–10

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

programmer to write the code in a sub-optimal way to ensure that the priorities correspond to the criticalities of the tasks.

Another method is to organize the system as a mixed criticality system like the one suggested in [8]. In such a system, each task is given a criticality level and the underlying runtime system aborts any task if letting it run would mean a task of higher criticality would miss its deadline. An improved version of this called Adaptive Mixed Criticality (AMC) was introduced in [1]. This allows for a better utilization of the system as, some (low criticality) tasks missing deadlines can be accepted. It also makes the formal verification of a system easier, as not all tasks must be checked.

This paper will explore the idea of using online information available at run time to get a more accurate estimate of the execution time of a single instance of a task. This online WCET will then be used to improve the utilization of systems with tasks of mixed criticality, by running one of several alternative tasks.

The rest of this paper is organized as follows:

Chapter 2 presents previous work on online WCET analysis and mixed criticality systems. In Chapter 3 the new scheduling scheme is introduced. Chapter 4 describes a simulation demonstrating the benefits of our scheme, and in Chapter 5 ideas for future work are presented. Chapter 6 concludes this paper.

2 Previous work

2.1 Online execution time analysis

One has traditionally used very pessimistic estimates for execution times, WCET, when designing real time systems and checking if they will be able to handle all tasks without missing deadlines. However, the assumption that all tasks will always execute for their WCET often results in severe under-utilization of the systems resources, since the execution time of most tasks have a certain dynamicity. They can, for example, spend merely a small percentage of their WCET most of the times they run and only in a few cases, or potentially never, spend the entire WCET.

Instead of always using the worst case execution time, a system was suggested by Sverre Hendseth and Giorgio Buttazo in [4] where the scheduler checks certain task parameters (input, program state etc.) to determine a better estimate for how long the task will need to execute in a particular release. This execution time estimate, computed online, will be called $WCET^{ON}$, and the traditional WCET determined offline will be referred to as $WCET^{OFF}$. The system utilizing the added information provided by the $WCET^{ON}$ will be referred to as OETA (Online Execution Time Analysis). In this system each task will have two entry points: the regular entry point for the actual task code and one for the OETA code. This allows the scheduler to run the analysis and get a better estimate of actual runtime required by the task than the $WCET^{ON}$ before letting the task run.

An important requirement of the estimation algorithm is that it must never give an estimate that is too low. That is, in order to be useful, it should provide a better estimate than the $WCET^{OFF}$, but it must still be sufficiently generous in order to ensure that the actual execution time does not exceed the $WCET^{ON}$. It is, after all, still a *worst* case execution time, although computed with better accuracy, due to the availability of more information.

Furthermore, the execution time of the estimation algorithm should be reasonably modest, so that it does not incur excessive overhead on the system.

The idea of computing the execution time of a task at runtime has also been suggested by Stancovic et al. [7] for use in the Spring kernel. This system allowed a tasks execution time to be "a formula that depends on various input data and/or state information". Instead of giving each task two entry points which allows for complex program code to compute the execution time, it is done with a simple mathematical expression.

2.2 Mixed criticality systems

A classic real-time system consists of a set of tasks that run together on the same hardware, and must complete execution before their respective deadlines. To assure that every task finishes before its deadline, each task is given a priority, determined by some scheduling policy. If a task with a higher priority than the currently running task is released (becomes ready to run), the lower priority task will be interrupted in order for the new task to be run, and then resumed after the high priority task is finished. These priorities does not, however, reflect the criticality of the task. In an aircraft the low criticality task that handles the cabin air conditioning may be given a higher priority than the task making sure that the landing gear is lowered on time (a very highly critical task) if the scheduling algorithm finds that this arrangement of priorities makes sure that all tasks will finish by their deadline.

This does, however, tend to become very inefficient as more tasks of varying criticality are moved onto the same hardware. This trend has become increasingly prominent, as concerns about space, energy efficiency and cost have grown [2]. The issue is that since important, potentially life-preserving, functionality is a part of this system, the other less important tasks must undergo the same rigorous tests and certification. This might lead to a severe under-utilization of the system, since making sure that all the highly critical tasks finish by their deadline requires all tasks to be given a pessimistic execution time estimate. In theory, one must allocate the WCET to each task. Finding the exact WCET is, however, not trivial [8]. The solution then, is to give each task a very generous WCET, to be "sure" that the task will not exceed it.

One way of handling different degrees of criticality is by giving highly critical tasks a high priority, and thereby ensuring that they will run without interference from the non-critical tasks.

In [8], Steve Vestal proposed a scheme for scheduling mixed criticality systems. The motivation behind this was that the different tasks of a real-time system does not necessarily need the same level of assurance in regards to certification. When a system is to be certified, only a certain number of the tasks are actually interesting to the certification authority. These are the highly critical task, e.g. those that deal with preventing damage to the system, death etc.

Several models have since been suggested, that improve the original work in [8]. One of these, called Adaptive Mixed Criticality (AMC) has been shown [1] to surpass the others in amount of task sets that are schedulable. In his 2013 review article on mixed criticality systems [2], Alan Burns writes that this is still the case[1]. The basic idea of AMC is that if a task executes for longer than its allotted time at a given criticality level, all tasks of the lowest criticality are aborted.

[1] He does, however, note that an extension of the AMC by Zhao et al. has improved the stack usage.

2.3 Alternative tasks

The use of alternative tasks is a way to give a task a better chance of completing its purpose. If the task cannot finish in time it will be aborted, and the alternative task is run instead. This alternative task will complete the job of the task faster, but with a lower quality of service. Such a system is described in [5].

The use of alternative tasks can be seen from two perspectives, it can be used as a fallback mechanism that allows a task to complete its mission, even if there is not enough time to perform the ideal task routine. Another way of looking at it is a possibility to run a routine that one would not normally consider possible if there happens to be sufficient time available at the moment.

3 OETA Scheduling Scheme

This work presents a new scheduling scheme. Its prominent features include:

- Online WCET-analysis is run at every task release and used to give the scheduler more precise timing information.
- Tasks consist of one or more task variants. If there is not enough time to run the preferred variant of a task, an alternative task variant can be run instead. This decision is made by the scheduler.
- Tasks have a criticality level associated with them to determine what tasks to prioritize over others.

3.1 Online WCET-analysis

Performing the WCET analysis online can be done in several ways. Most algorithms have an execution time that depends on their input size. One way of doing online WCET estimation is to have a lookup-table of values already computed offline, allowing for complex WCET algorithms to be used. Since it is run by the scheduler, it can also take into account system information that the application does not know about, such as cache use, the state of IO-devices etc.

An example of predicting execution time with information available online is the use of Hough transform to detect lines in an image. Several uses for this in UAV systems have been proposed, such as power line surveillance[6] and autonomous landing[9]. The algorithm consists of some preprocessing to find the edge pixels (points of high contrast) in an image, followed by looping through all edge pixels and checking if they are in a line. One would assume that the execution time of the second step is a linear function of the number of edge pixels. We have taken 100 pictures, run them through a standard Hough transform and timed each step.

Figure 1 shows the execution time of the Hough transform given the number of edge pixels in the image. The Hough transform was run 100 times on each image. We can see that the execution time behaves as expected, it is approximately proportional to the number of edge pixels in the image plus some overhead in the preprocessing. The variance in execution time for each image is most likely caused by interference from other tasks running on the system.

Figure 2 shows the portion of execution time used for the different parts of the Hough transform. We can see that the preprocessing takes a constant amount of time, and that it is the Hough transform that behaves dynamically. Also note that the OETA portion (counting the edge pixels) is very small compared to the rest of the work, and that the rest

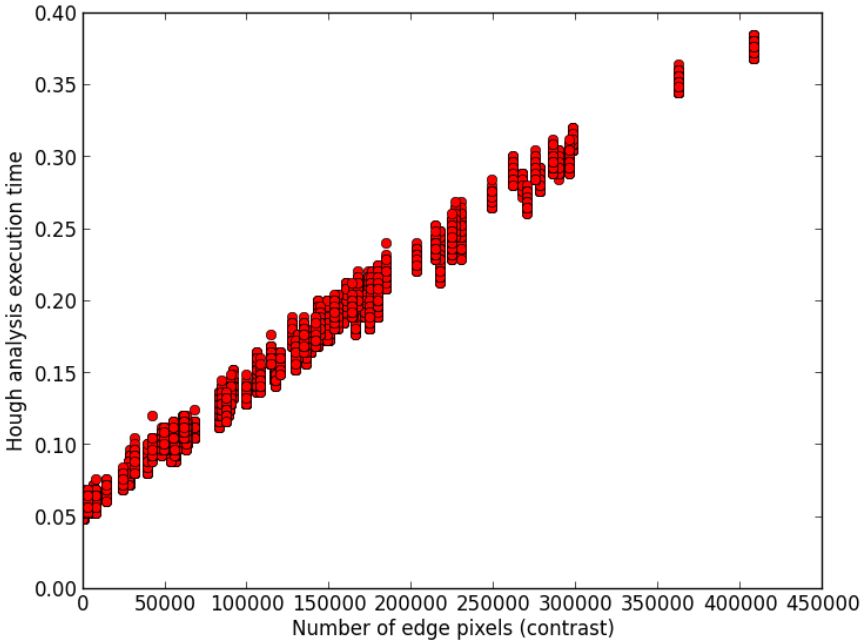


Figure 1 Execution time of line detection through Hough transform relative to the images number of edge pixels (contrast)

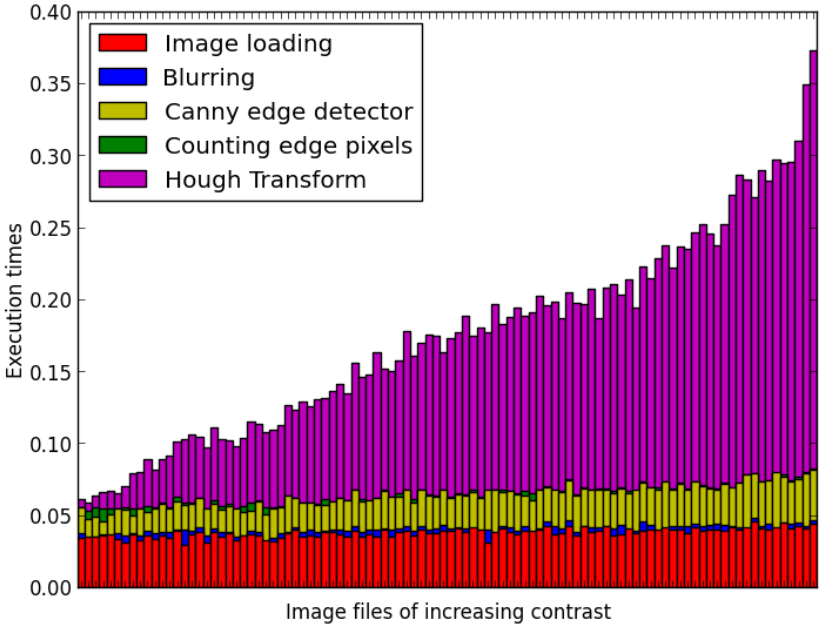


Figure 2 Average time of the different steps of the Hough transform. The images are ordered by number of edge pixels.

of the preprocessing (loading, blurring and running the canny edge detector) is not an added overhead of the OETA, and must be performed regardless.

These tests indicate that counting the number of edge pixels and using this number in a simple linear function is a good way to estimate the execution time of the Hough transform. We believe that level of contrast and other similar metrics can prove to be used to estimate the execution time of several other computer vision algorithms.

3.2 Assumptions

The OETA algorithm exists There exists some algorithm for determining a $WCET^{ON}$ for every task.

$WCET^{ON}$ is pessimistic This was also stated in the description of the OETA in section 2.1. The $WCET^{ON}$ must be such that it always is an upper bound for the execution time of the task.

3.3 Model

A task has the following information associated with it:

- A criticality: L
- A priority: P
- A deadline: D
- A period: T
- A set of task variants
- An estimator preprocessing routine

A task variant has the following information associated with it:

- An entry point for the actual task
- An entry point for the OETA code

3.4 Behavior

Two suggested behaviors are described below. It is important to note, however, that the principle of using online WCET analysis to help the scheduler make decisions as described above is not restricted to the following schemes. These represent two very different methods of scheduling tasks given the extra knowledge about WCET.

3.4.1 A greedy scheduling scheme

When a task is released, its estimator preprocessing routine is run. This routine acquires and sets up all the necessary data for computing the WCET of a variant. The OETA algorithm for that tasks first task variant is run, and the result is saved. A test is performed to check whether the addition of the new tasks will result in a deadline miss in the system. If that is not the case, all tasks are allowed to run as normal. If the test finds that some task will miss its deadline, the scheduler aborts the lowest criticality task, and checks the $WCET_{ON}$ of that tasks next task variant. If no task will miss a deadline with that task variant, the task variant is used. If not, the next task variant is checked until either the system is schedulable, or the task has no more task variants. If it has no more task variants, the task is aborted and the scheduler starts degrading the service of the second least critical task until the system is schedulable. This is illustrated in Figure 3.

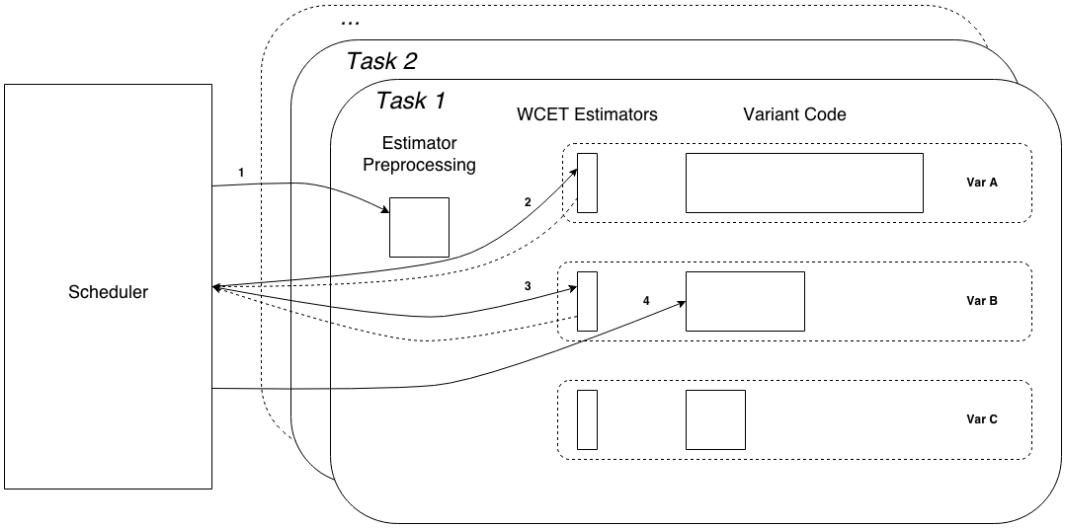


Figure 3 Deciding on a task variant. The released task is in this case the least critical task currently running. The process starts with the scheduler running the estimator preprocessing routine (1), then the WCET estimator for task variant A is run and its result returned to the scheduler (2). The scheduler deems it impossible to run task variant A, and calls the WCET estimator for task variant B (3). As the system is schedulable with the estimate for variant B, the scheduler starts running its actual code.

This scheme performs the scheduling relatively fast. It is at all times trivial for the scheduler to find the task to degrade/abort, since it always chooses the least critical. The problem with this scheme is first and foremost that it is not bandwidth-preserving if the released task is not currently the least critical task in the system. If a highly critical task enters the system, thus making the task set unable to guarantee all deadlines, a task of lower criticality will be aborted/degraded, causing it to have wasted all the CPU bandwidth spent by the task so far.

Another problem with having the scheduler aborting tasks is freeing the resources locked by that task. This requires the run-time system to keep track of every tasks locked recourses and release them when a task is aborted.

One possible extension of this scheme is giving each task variant a criticality instead of the tasks themselves. This will make the system more flexible in that it would the designer to prioritize the most bare-bones variant of a low criticality task over the most time-consuming variant of a more critical task. For example, we might say that sending a small "I am alive"-message is more important than running a complex MPC-loop with a large horizon to control a vehicles flight, even though in normal circumstances keeping the UAV in the air is more critical than sending a detailed status report to the controller.

3.4.2 A more complex scheduling scheme

One can envision a system where the goal is to not abort a task after it has started, or at least do it as rarely as possible. Instead the system should be analyzed well enough in advance that the scheduler can, at the release of a task, choose a task variant that will not be required to abort. One advantage of this is that one will not have wasted CPU-cycles running code that did not produce any result. Another is that it does not introduce the problem of releasing resources that a task has locked before it is aborted.

Instead of using criticality-levels to determine which task to degrade/abort, it uses a value function. This function can take into account a tasks criticality, how long it has been executing, how much time would be saved if it was degraded etc. to determine what tasks to degrade/abort. It will be necessary to take into account information about the other periodic tasks that are not running, but will be released during the tasks execution time, since these are the main reason that tasks needs to be aborted in the scheme described in Section 3.4.1. The problem of finding a set of tasks to degrade that will maximize the value function is clearly NP-hard. The use of heuristics is one possible way of solving this.

This system would in effect be an admission based scheduler with dynamic values as described in [3].

3.5 Setting priorities

Priorities are set using the same strategy as for the AMC system[1]. It uses a modified variant of Audsley's priority assignment algorithm. A task is identified which may be given as the lowest priority. It is then taken out of the set of tasks and the task of the second lowest priority is identified in the same manner. This continues until all the tasks have a priority assigned to it.

3.6 Testing for deadline misses

Testing for deadline misses is done with a simplified response time analysis. The general idea is that if the sum of $WCET^{ON}$ for all tasks with priority equal to or higher than a given priority is less than the time to the earliest deadline of any task of that priority, no tasks with that priority will miss a deadline. This test is then carried out for every priority level. The number of steps for this algorithm is the same as the number of tasks currently ready to run. This will run in $O(n)$ time, where n is the total number of tasks in the system. The WCET analysis is not run for every task each time, only for the task that was released. The other tasks $WCET^{ON}$ are saved by the scheduler from their release time. This can be written as

$$\sum_{\{i|p_i \geq p^*\}} WCET_i^{ON} < \min_{\{j|p_j = p^*\}} D_j \quad , \qquad \forall p* \in P \tag{1}$$

Where P is the set of priorities, and p^* is the priority level we are currently checking.

This is a simplified version of the response time analysis. The simplification is that it does not take into account periodic tasks that will be released again (for example task 1 could be released again while task 2 is executing). This is because we treat every task as a sporadic task and run the check for deadline misses at every task release.

This test is performed only on the tasks *that are currently ready to run*, not the entire task set of the system. In a system without time slicing, this test is sufficient, but not necessary. Further optimizations are likely possible.

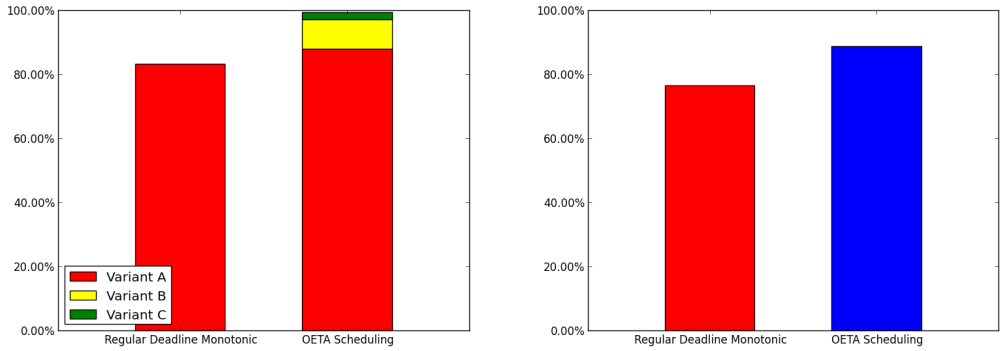
3.7 Schedulability Proof

Since the system does not require every task to finish by its deadline, making a formal proof to show that a system is schedulable is not trivial. One way to do it would be to decide on a set of tasks (or task variants) that are required to finish by their deadline for the correct function of the system (hard real time tasks), and then perform a response time analysis

for that set of tasks. These tasks should of course be given the highest criticality values to ensure that they are the ones remaining if multiple tasks must be aborted. For the response time analysis, offline WCET estimates would have to be used.

4 Simulation

A simulation has been run in order to assess the benefit of our system. A task set consisting of five tasks has been given random execution times and deadlines. Variations in execution time was simulated by randomly doubling or tripling the execution time of any task in each simulated run. The task set was run by a simulated standard deadline monotonic scheduler and a simulation of our system that added a constant overhead to represent the online WCET analysis, and also allowed a task to spend only half or 1/4 its execution time if needed to represent an alternative task being run. This was done for 10000 task sets. The results of the simulation are shown in Figure 4. As the values of execution times and deadlines are somewhat arbitrary, the qualitative observations are more interesting than the quantitative.



(a) Number of tasks meeting their deadline

(b) Amount of time spent executing tasks that met their deadline

Figure 4 Simulation results

Figure 4a shows how many tasks were able to finish by their deadline. We can observe two things from it. Firstly, the fact that more tasks were able to finish by their deadline using OETA Scheduling. This is to be expected, since a task can be swapped out with a shorter variant if there is not enough time to run it. Secondly, we can see that with OETA Scheduling more tasks of Variant A were able to complete before their deadline. Since it completes some tasks faster when needed, there is more time available in the system.

Figure 4b shows the amount of time that was not wasted by executing tasks that did not meet their deadline. Since it can switch to shorter tasks, we can see that OETA scheduling is outperforming the regular deadline monotonic scheme.

A more complex implementation allowing actual task code to be run is being developed. This will, when finished, allow for more extensive tests to be conducted.

5 Future work

A deeper analysis of a schedulability proof and priority assignment should be conducted. This is important in order to make a scheme for static verification of systems. Schemes for

choosing which tasks to degrade and how to detect a possible deadline miss should also be developed and analyzed further. If the more flexible scheme described in Section 3.4.2 is to be taken further, analysis of different heuristics to determine the task variant to run for a released task should be conducted.

A better practical runtime environment should also be produced to facilitate testing and experimentation of different scheduling schemes utilizing OETA. Tests on actual overhead and practical benefits of online WCET can then be carried out. It will also make it easier to test online estimation algorithms for real applications. This implementation has been started.

6 Conclusion

With the added information about system state, application state and program input, online WCET analysis can be used to make a more precise estimate for the execution time of a task than traditional offline methods. As real time systems grow in complexity, tasks exhibit a greater degree of dynamicity. This can for example be seen in computer vision applications that are relevant for UAV systems. This dynamicity makes traditional offline WCET analysis less applicable, since it will give an overly pessimistic result.

We propose a system that performs an online WCET estimation for every task release, taking into account task input, application state and general system state where relevant. Each task can have a set of alternative task variants that the scheduler may choose to run instead of the main task. This choice is made primarily based on the online WCET estimate. This was shown in a simulation to yield both a higher number of tasks finishing by their deadline and less time wasted on tasks that missed their deadline.

References

1 S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Proceedings of the 2011 IEEE 32Nd Real-Time Systems Symposium*, RTSS '11, pages 34–43, Washington, DC, USA, 2011. IEEE Computer Society.

2 Alan Burns and Bob Davis. Mixed criticality systems - a review, 2013.

3 Alan Burns and Andy Wellings. *Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX*. Addison-Wesley Educational Publishers Inc, USA, 4th edition, 2009.

4 Sverre Hendseth and Giorgio Buttazzo. Exploiting online WCET estimates. In *RTSOPS 2012 Final Proceedings*, pages 9–10, 2012.

5 C.M. Krishna and K.G. Shin. *Real-Time Systems*. Computer engineering. McGraw-Hill Education, 1997.

6 Zhengrong Li, Yuee Liu, Rodney Walker, Ross Hayward, and Jinglan Zhang. Towards automatic power line detection for a uav surveillance system using pulse coupled neural filter and an improved hough transform. *Mach. Vision Appl.*, 21(5):677–686, August 2010.

7 J. A. Stankovic and K. Ramamritham. The spring kernel: A new paradigm for real-time operating systems. *SIGOPS Oper. Syst. Rev.*, 23(3):54–71, July 1989.

8 Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, RTSS '07, pages 239–243, Washington, DC, USA, 2007. IEEE Computer Society.

9 Tyan Vladimir, Dongwoon Jeon, Doo-Hyun Kim, Chun-Hyon Chang, and Jung-Guk Kim. Experimental feasibility analysis of roi-based hough transform for real-time line tracking in auto-landing of uav. In *ISORC Workshops*, pages 130–135. IEEE, 2012.

D Review of the paper

The paper was not accepted by the WCET Workshop. The review is given below.

———————— REVIEW 1 ————————

PAPER: 13

TITLE: Using online WCET analysis to schedule alternative tasks in mixed criticality systems AUTHORS: Fredrik Bakkevig Haugli, Amund Skavhaug and Sverre Hendseth

OVERALL EVALUATION: 0 (borderline paper) REVIEWER'S CONFIDENCE: 4 (high) Reviewer's confidence: 4 (High) Applicability to the workshop: 3 (Fair)

——— REVIEW ———

The paper presents an online scheduling scheme where a tasks WCET estimate is refined at task release (based on some online information) and the available slack time is exploited to schedule tasks with low criticality. The paper tackles an interesting hot-topic and is well written and easy to read. The results of the evaluation are promising.

Major criticisms:

- The presentation of the used WCET analysis is missing and the paper concentrates on the scheduling part. The assumption that a WCET analysis can be done online is unrealistic. A feasible approach would be to use an offline precomputation as for instance in parametric timing analyses. This was also indicated in Section 2, but the relevant citation is missing (parametric timing analysis + dynamic scheduling has been explored in for instance in [X]) and the authors do not state if they assume such an approach or not. - The description of the evaluation is not complete and the results are thus not re-producible. I am missing already very basic information such as the range of the periods or the overall task utilization. How many tasks are considered to have a high criticality, how many a low criticality? The most important omission, however, is the runtime overhead of the presented algorithm: both the scheduler and the online WCET analysis require execution time and thus reduce the available processor time to execute other tasks. An unfortunate selection of the tasks' parameters may even result in a worse per-

formance of the proposed scheduling theme compared to the basic DM scheduling. The authors state that they assume a constant overhead to represent the online WCET analysis, but do not specify it. Technically, zero is also constant.

These drawbacks strongly reduce the quality of what otherwise would have been a valuable paper.

Minor issues:

- page 1, second to last paragraph: "This method divides a highly critical task is [sic!] into ..." - section 2.2: please state explicitly that you assume a preemptive scheduling policy as most safety-critical applications assume non-preempted execution. - section 2.2.: "These priorities does not ..." - section 3.4.2: 'A more complex scheduling scheme' is a rather unconventional name for a scheduling scheme. - section 3.4.2: I am not convinced that the finding a set of degradable tasks to maximize a value function is NP-complete. Do you have any proof or at least some indication to justify this claim?

[X] ParaScale: Exploiting Parametric Timing Analysis for Real-Time Schedulers and Dynamic Voltage Scaling. Sibin Mohan, Frank Mueller, William Hawkins, Michael Root, Christopher A. Healy, and David B. Whalley. RTSS, page 233-242. IEEE Computer Society, (2005)

———————— REVIEW 2 ————————

PAPER: 13

TITLE: Using online WCET analysis to schedule alternative tasks in mixed criticality systems AUTHORS: Fredrik Bakkevig Haugli, Amund Skavhaug and Sverre Hendseth

OVERALL EVALUATION: -1 (weak reject) REVIEWER'S CONFIDENCE: 4 (high) Reviewer's confidence: 4 (High) Applicability to the workshop: 3 (Fair)

——— REVIEW ———

The authors present a very sketchy scheme postulated on the existence of an online utility might be used to determine, presumably based on the current execution context, the band of execution time within which a program eligible to run might be allocated. On that online information (which the authors call "online WCET") the online scheduler might accept the job or else reject it in its present form and allow instead a less time-demanding variant of it. The submission includes a very simplistic simulation experiment that shows that more jobs might complete some variant of their execution is the proposed scheme was used, then it would happen if no provisions were made and tasks were free to exceed their allocated execution-time budget.

The submission seems to inspired on reference [4], which however is too shallow a basis to sanely build upon.

In spite of its prior use in citation [4], the term WCET in the connotation used in this work does not seem to be appropriate, nor is the use of "online WCET analysis" or "online execution time analysis". It would seem that the execution-time bounds on which scheduling decisions are made (for acceptance or selection of a feasible program variant) are all taken offline and then only selected at run time based on knowledge of the current execution context.

The formulation of the assumptions presented in Section 3.2 is highly questionable: not even under the authors' own hypothesis, execution time bounds are "analysed" online, rather, the runtime uses parameter monitoring to determine the band of (statically-allocated) execution time of the program(s) of interest.

The scheduling scheme presented in Section 3.4.1 seems to suggest that the whole system holds while some sort of context-aware WCET analysis is made for

a program candidate to execute: the overhead of this scheme is untenable. The solution presented in Section 3.4.2 is no less untenable, given the complexity of defining a useful and realistic "value function" to decide whether a task can be aborted or not.

All in all, the ideas on which the authors construct their argument are immature and not very convincing, in solidity and in depth.

Some remarks follow on required textual fixes. - Globally applicable real time –> real-time [when used as an adjective]; same for high/low criticality offline –> static - Section 1, page 1, para 2 a tasks temporal properties –> one task's temporal properties - Section 2.2, page 3, para 1 These priorities does not –> These priorities do not - Section 2.3, page 4, para 1 Reference [5], as given, is void and meaningless - Section 3, page 4, para 1 The dash has no place in "WCET-analysis" and should be globally removed - Section 3.1, page 4, para 1 IO-devices –> IO devices - Section 3.1, page 4, para 2 Insert a space before the citation reference

———————— REVIEW 3 ————————

PAPER: 13

TITLE: Using online WCET analysis to schedule alternative tasks in mixed criticality systems AUTHORS: Fredrik Bakkevig Haugli, Amund Skavhaug and Sverre Hendseth

OVERALL EVALUATION: -2 (reject) REVIEWER'S CONFIDENCE: 4 (high) Reviewer's confidence: 4 (High) Applicability to the workshop: 3 (Fair)

——— REVIEW ———

Summary: The authors ague that the offline WCET estimation is usually too pessimistic and is far more than the task execution time in most cases. This leads to inefficiency as idling resources are wasteful. The authors propose a new model that uses online WCET estimation and makes scheduling decisions at the runtime. With existing algorithms to calculate online WCTE estimation, the paper proposes two different scheduling schemes. The first one is to try degrading the quality of tasks starting from low to high criticality level until the system is schedulable. When a task is degraded to the lowest quality and the system is still unschedulable, the task is aborted before degrading the next task with higher criticality. The second scheme proposes to use a function to decide which task to be degraded without the trial procedure as in the first scheme.

Strong points: The idea is potentially useful in developing an efficient scheduling model for mixed criticality systems.

Weak points: - The paper fails to provide an abstract view of the proposed model. Also, there are not enough details of how this model should work. It is very difficult to follow and understand the idea of the paper. - In the first scheduling scheme, when none variant of the task with lowest criticality can be used to make the system schedulable, the task should be aborted. The scheme then considers to degrade the task with the next higher criticality. For example, there are 3 tasks: task A with 2 variants A1, A2; task B with 2 variants B1, B2; and task C with 1 variant C1. Quality of A1 is lower than A2, B1 is lower than B2. Criticality of A is lower than B, B is lower than C. The system is currently executing A2 and B2. C is released and it can not be scheduled. A is degraded to A1 and the combination (A1, B2, C) is still not schedulable. A is aborted, B

is degraded to B2 and combination (B2, C) is then schedulable. Q1: Why after aborting A, why does the scheduler degrade B immediately? Should it try first if the system is schedulable without degrading B, i.e. (B1, C)? Q2: If B is degraded to B1 and the system is schedulable, will the scheduler consider to restart A, i.e. (A1, B1, C)? Suggestion: The scheduler should consider all combinations of task variants before aborting any task. The system now examines all combinations in the following order: (A1, B2, C), (A1, B1, C), (B2, C), (B1, C), (C). The scheduler chooses the first combination that is schedulable.

- The second scheduling scheme is described very vaguely. There is no detail about the propose function.

- Organisation problems: + The introduction section fails to present the context and motivation of the paper. + The previous work section is poorly presented. For each related work, the authors list some disconnected details but do not provide an overview or concise understanding of the work.

- Unclear points: + the model supports sporadic tasks (section 3.6) but in the second scheduling scheme requires information about periodic tasks (section 3.4.2) + Task information: what is priority? how is it different from criticality? what is period T? + In section 3.5: the model uses the strategy in the AMC system to set priority for tasks. The paper needs to provide an overview how the AMC system work, not just a reference.

- Presentation problems: + Figure 2 and Figure 4: the colours are not distinguishable in gray-scale printing + Is it necessary to quote the whole phrase in reference [7] (last paragraph in section 2.1)? + Why does "sure" need to be quoted? (last sentence in the second paragraph of section 2.2)

- There are a lot of grammar problems. Here are some examples: + WCET estimate –> WCET estimation (note that "estimate" is a verb, its noun is "estimation") + "a tasks temporal properties and importance" –> "a task's ..." + "This system allowed a tasks execution time to be ... " –> "This system allows a task's execution time ..."

E Digital attachment

This thesis has a digital attachment in the form of a zip-file. It contains the source code for the scheduling simulator. The code is organized in a git repository in the same folder. To compile the module tests, use the included makefile.