Computational resource management of multi channel controller

Hodai Goldman

September 25, 2017

Contents

1 Introduction

1.1 motivation and overview

Today's computer power allows for consolidation of controllers towards systems where a single computer regulates many control loops, each with its

varying needs of computation resources. This brings two research challenges that we intend to attack in this thesis:

- How to schedule control tasks in order to achieve good performance in terms of control measures (overshoot, convergence time, etc.), within the still limited computation resources?
- What is a good interface for co-design of scheduling and control?

While it is possible to build control systems using standard operating systems, either real-time or desktop, with static or with dynamic scheduling schemes, there is an agreed opinion in the control community that these do not serve well for the purpose outlined above. For example, in [**?**], the authors say:

> 'The delay and jitter introduced by the computer system can lead to significant performance degradation. To achieve good performance in systems with limited computer resources, the constraints of the implementation platform must be taken into account at design time.'

Similar views are expressed also in other papers [**?**, **?**, **?**].

Desktop type operating systems, like Windows and Linux, schedule for computational efficiency but do not allow for worst-case performance guarantees. Real-time operating systems, on the other hand, sacrifice some efficiency for timing predictability, but the type of timing guarantees that such systems provide usually are not directly guaranteeing the control performance of the system. When using such operating systems for control, engineers usually apply controllers that work in a fixed periodic manner witch the control behavior becomes deterministic and control performance can be guaranteed. This is not efficient because resources can be better utilized if controllers act at higher frequencies only when needed.

In this work we show how to combine the efficiency of dynamic scheduling with the predictability of real-time scheduling, in a way that is more suitable for control systems then periods and deadlines. We show that applying control computations at dynamically adjustable periodic, and with variable computational demands, based on real-time information, allows for better utilization of the computational resources and therefor better control performance.

The main innovation of this work is the system architecture design. In practice most of the real-time control systems designs consists of to separated elements, (1) control and estimations tasks, and (2) a task scheduler which

divide the resources (CPU time) between all tasks. Although those two parts are constantly affected by each other they barely communicate, this lead to static (constant) scheduling regardless of the current state or needs of the system. In this thesis we develop an architecture design for control systems in witch the scheduler (2) is aware of the current state of the system, and makes *state depended* scheduling for that state. We show how allocate resources by **current** needs give better resoults than allocate resources for the **worst-case** needs.

2 Problem Statement

We are concentrate on close feedback control loop based systems as shown in Figure 1. We assume a close feedback control loop where the physical plant, *System* state (x), is monitored via an array of sensors (*Sensing*) which produce raw data (y) that represents noisy sample of some functions of the state variables. We assume uncertainty observations from the sensors so, after sensing, an entity called *State Estimator* aggregates the raw data from all the sensors and makes an educated estimation ($\hat{x}$) of the current state. Then the *Control Law* entity generates the controller outputs (u), control the actuators, to change the state of the system in order close the gap between the current estimated state and the reference desire state (*Input* r).

The current state-of-the-art computer embedded control systems, as descibed e.g. in [**?**], are developed as follows, control engineers first design control tasks (control and estimation tasks) as periodic computations, then they specify the required periodic frequency for the task, and then software engineers design a scheduler that ensures the periodic frequency requirements are met. The last step is usually done using pre-computed knowledge of the expected (maximum) duration of the tasks. Note that in order to ensure that the controlled plant is always properly maintained, for example engine temperature never exceeds maximal safe temperature, the periodic frequency must be tuned for the worst-case state that the system might be in.

We show how to achieve better performance and better resource utilization for control systems by using richer and more flexible requirements model for the tasks. Specifically, we develop tools and methodologies such that control engineers are able to specify more accurately the requirement features of their control tasks, that the scheduler will use for executing dynamic resource assignment that will, at the same time, guarantee required control performance and will be efficient in its use of computational resources.

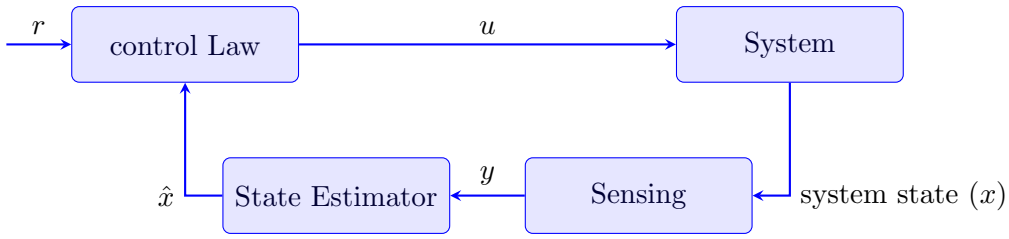

Figure 1: A typical close feedback control loop where the physical plant (*System*) is monitored via an array of sensors (*Sensing*) which produce noisy sample of the state variables y. And after sensing, *State Estimator* aggregates the raw data from all the sensors and produce estimation of the current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

3 Architecture: Automata Base Scheduler

As describe in Section 2 control systems have two main parts: *state estimation* and *controller*. All the common used techniques for controllers, e.g PID, are relatively low resources consumers and there is no real justification of optimizing it, but this is not always the case for estimation process, some times there are heavy computational tasks as part of state estimation task, this refers to both the estimator and sensing stage. The sensor can be complex computations of Global Positioning System (GPS) in small processor or even camera with heavy computer vision. Those sensing task need to run in real-time and can interfere with rest of the system tasks. In out architecture we concentrate more in those heavy sensing processes in order to improve the resource utilization and control performance. (See **??**)

The architecture of control system as we believe, is illustrated in Figure 2. The architecture, that is based on modern controller architecture where the system has multiple controlling tasks, comes to support an efficient scheduling protocols in modern control systems consisting of a processor that runs all the tasks of many independent control loops in the system. The current state of art, as described above and as we observed, e.g. in the code of APM [**?**], is that the designers of each task, control or estimation task, specify a fixed rate for invocations of the corresponding task (called *period*). In our methodology, there is a richer and more intense interaction between the *Scheduler* and the control loops. Each control loop (blue components in the Figure 2) will tell the *scheduler* of its level of certainty ($P = var(\hat{x} - x)$)

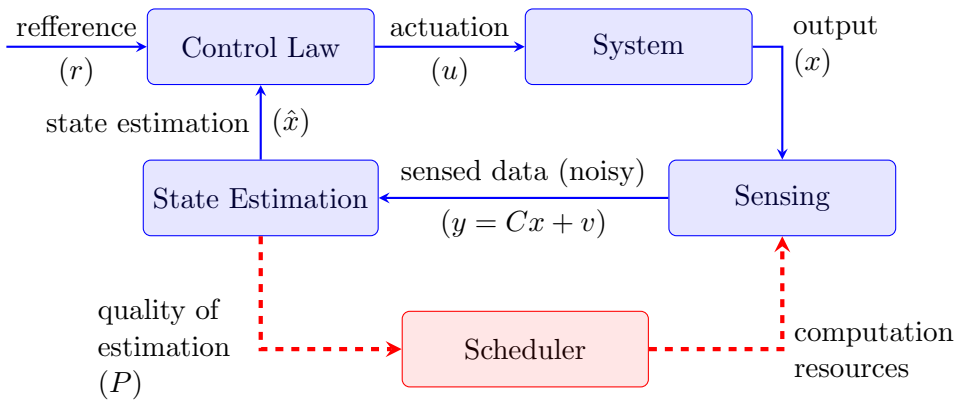

Figure 2: A general scheduling framework. Each control loop (depicted in blue) informs the resource allocator (Scheduler) of its quality of estimation and the allocator allocates accordingly the computation resources among all the control loops, in order to maintain valid estimation quality.

and the allocator will allocate resource (processor time) based on this data, meaning the *scheduler* allocate resource dynamically based on the system current needs rather than the worst case needs.

Our methodology is general and may be applicable in a wide range of applications. However, in this initial phase of the research, we focus on a specific sub-domain and in handling all technical issues in order to prove the concept. In this thesis we develop and implement a vision based controller for drone (see Section 5) and analyze the concept with it.

TODO - need to fix here!!!

Below we will dive in each part of the new architecture and explain how it should be adjusted.

3.1 Sensors

The sensing process are assumed to be periodic task. In order to allow dynamic scheduling it must be able to execute in a variable period between executions or to allow control of the execution duration time. There are two general approaches we consider for this task: use *any-time* algorithms or finite *execution-modes* based tasks. *any-time* algorithms are algorithms that always try to improve the solution quality, for sensing processes, we can define deadline for the "searching" process every execution, ??? present any-time solution for image processing [**?**]. Set of *execution-modes* based, a

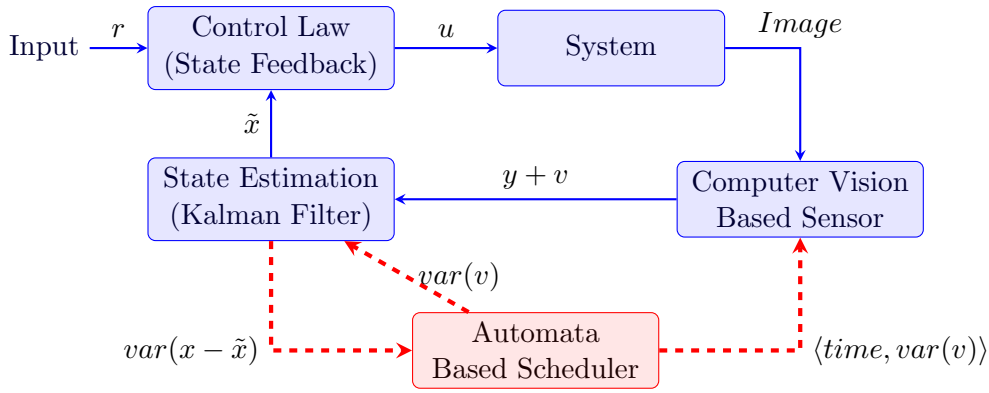

Figure 3: The controller framework we will implement, the *scheduler* will allocate CPU time ($\langle time, var(v) \rangle$) for the *Computer Vision* task base on the state estimation certainty using guarded Automata.

discrete version influence from "contract-based" presented by ??? Pant [**?**], here each sensing task have finite number of "operation modes", e.g. different algorithms to proximate the same observations value, then the scheduler can perform the apropriate mode for every cycle.

For simplicity, we only work with *execution-modes* based sensing tasks, all the modes are pre-defined and identified by a pair $\langle Duration, Variance \rangle$, which define the error $Variance$ of this mode and the run-time $Duration$ is the of this mode. We assume that longer modes produce solution with smaller error variance.

3.2 State Estimator

State Estimator (the filter) is part of the estimation process, this unit receive the measurements (y) from the *Sensors* and produce from them the state estimation required by the controller ($\hat{x}$). When the *Sensors* is accurate enough we may be able to pass it directly to the *Controll Law* ($\hat{x} = y$), but usually the measurement is noisy and the main goal of the State Estimator is to reduce this noise from the measurement (see Section 7.1).

If one consider using the optimal estimator Kalman filter (describe at Section 7.1) in the estimation process, one of the parameter that need to be consider in calculation is the covariance of sensor error (noted by R in Section 7.1). But in the new framework the sensor have diferents operation modes with variable error covariance, this means that we need to have also variable state estimators correspondingly. In order to adjust the sensor error

covariance, each cycle the scheduler will inform the state estimator about the new error covariance, and the state estimator will use corresponding parameters to make the next estimation.

3.3 Control Tasks

The control task (regulator) itself (*Control Law* in Figure 2) is responsible for closing the gap between current state estimation ($\hat{x}$) and the reference state (r), by manipulating the system actuators (e.g. changing the speed of the motors), that output to the actuators is noted by u.

The control task is usually a very low CPU consumer and been well studied [**?**, **?**]. Hence, the control task is not needed to be manipulated. For our testing we use the commonly used and well known technique from control theory called PID, A proportional-integral-derivative controller. In this technique the control output is based on the physical knowledge of the system dynamics, and have three variable parameters (P, I and D) that define the controller convergence behavior [**?**] ???.

3.4 Computation Resource Scheduler: Automata Based Scheduler

Real-time systems are mostly composed of multiple real-time tasks, tasks with time constraint, for example task that must response to an event within specified time constraints. The purpose of schedulers in such systems is to allocate the limited computational resources (CPU time) within all the task in the system. To do so, we need a well defined interface between the real-time tasks and the scheduler.

The most common way of describing the requirements of a real-time component is to specify a period, sometimes along with a deadline, which gives the frequency at which the component must execute. The designer of the component makes sure that the performance objectives are met as long as the component is executed consistent with its period. The scheduler guarantees that all components get enough resources. Specifying resource requirements using periods has advantages due to simplicity and analyzability, but has limited expressiveness, as elaborated in [**?**].

3.4.1 The Proposed Scheduling Methodology

In this thesis we develop a new methodology for allocation resources. We focus on how should engineers describe the requirements of a real-time component. To simplify, we will assume that the resource is allocated in discrete

time slots of fixed duration in the style of time-triggered architecture [**?**]. The specification framework for resource requirements is based on *Nondeterministic ω-automata* (finite automata over infinite words) [**?**]. Automata can be more expressive for describing specific requirements, and they are composable, i.e., it is easy to compose all the tasks requirements into an integrated automata, and its easy to analyze and manipulate using tools like GOAL [**?**].

Formally *task specification automata* is defined, similarly to Nondeterministic ω-automata, as tuple $A = (Q, \Sigma, \Delta, Q_0, Acc)$. In our setting, the *alphabet* of the automata is $\Sigma = T^2 \times C^2$ where C is the set of Boolean conditions variables, will be describe later, and T is the set of all tasks in the system. Each infinite word ($\alpha \in \Sigma^\omega$) of the automata define a possible tasks scheduling over a fixed period (time slots), and the automata language define all the possible scheduling. Now the scheduler only need to "walk through" the automata as follows, before every iteration (time slot) the scheduler choose state transition $(q_i, \{t_j, c_j\}, q_{i+1})$ from the current state q_i to the state q_{i+1}

Let's define $s()$

In our setting, the automata define continuous (fixed period) operation modes, and the condition of mode changing. The operation mode directly define a set of tasks that will be executed in the time slot (for example $s_{0.7}$ in Figure 4). We can stay at a single mode for some iterations, in this case the same tasks will be scheduled in each iteration. We can also take **discrete mode transition** in order to change operation mode after few iterations and schedule different set of tasks, for example, in Figure 4 the edge (m_1, m_2) says that if the previous iteration mode was m_1 and $estVar < 0.7$ we can change the mode and schedule the set $\{s_{0.2}\}$ in the next iteration.

Each infinite path on the composed hybrid automata (not necessarily with infinitely many mode transitions) represents a schedule that satisfy all the tasks requirements. In this architecture the scheduler only need to "walk through" the composed automata, this is, of course, a fast and easy computational task. In order to assure that we do not exceed the time slot duration, each task will have pre-defined maximum duration time like "deadline" in the traditional architecture. Now we can verify that every possible scheduling step can execute within a single time slot, in other words, every mode of the composed automata can be executed in a single time slot, by simply summing the "deadlines" of all the tasks in the mode tasks set. If we find a mode that goes beyond the maximum duration we can remove it from the automata so we never exceed the time slot duration.

Let us demonstrate the proposed automata based interface using the

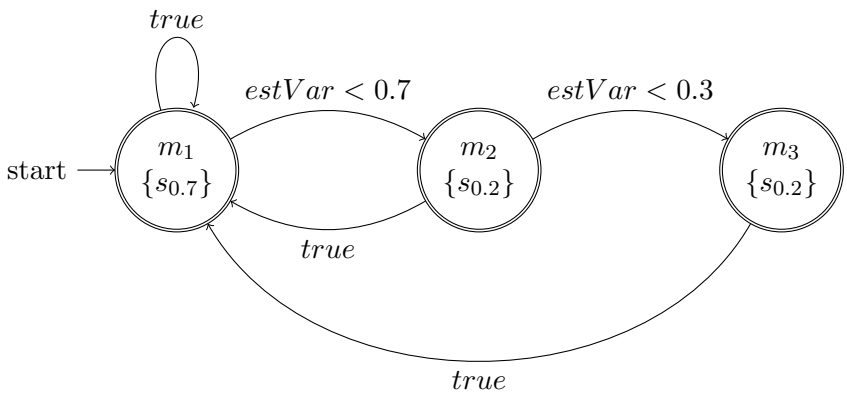

Figure 4: Example of guarded automata for our vision based sensor task, in this example the task has two operation modes, $\mathbf{s_{0.7}}$ which need 70% of the time slot but is more accurate vision computation and $\mathbf{s_{0.2}}$ which is less accurate but faster (need only 20% of the slot). The value of $estVar$ is $var(x - \tilde{x})$ of the previous iteration. Every time slot exactly one of the modes will be executed.

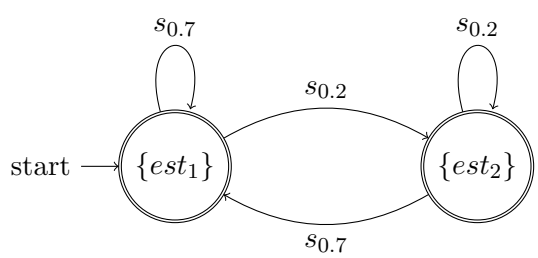

Figure 5: Example of guarded automata for our state estimator, in this example the estimator has two operation modes, $\mathbf{est_1}$ correspond to $\mathbf{s_{0.7}}$ and $\mathbf{est_2}$ correspond to $\mathbf{s_{0.2}}$ (see Figure 4).

system depicted in Figure 3. Assume we have two operation modes of the vision based sensor: (1) $s_{0.7}$ a very accurate operation mode that takes 70% of the time slot to execute, that is of course a significant amount of time, and (2) $s_{0.2}$ a less accurate operation mode but takes only 20% of the slot. In each time slot we can execute one of them and get the sensing process done, but if we use only $s_{0.7}$ every time slot we may not have enough time to execute all the tasks. On the other hand, if we use only $s_{0.2}$ we will have inferior estimates. Figure 4 shows an example of a guarded automata that guides the schedule. With this automata we can express rich specifications. In this example, execute $s_{0.7}$ is always allowed but if we need faster sensing we can use $s_{0.2}$ but only once in a row if the estimation error is not extremely bad (if $var(x - \tilde{x}) < 0.7$), or even twice in a row if the estimation error is good (if $var(x - \tilde{x}) < 0.3$).

The estimated estimation error ($estVar$ in the figure) is, in this case, the estimation error variance ($var(x - \tilde{x})$ in Figure 3).
This value ($estVar = var(x - \tilde{x})$) is calculated by the state estimator task (Section 3.2) and is passed to the scheduler as discussed before.

Each of this operation modes ($s_{0.7}$ and $s_{0.2}$) have different accuracy, specified by $var(v)$ in Figure 3, and if we want to get optimal estimation the state estimator mast be configure correspondingly, i.e., the sensing error variance should be adjusted to the correct value ($var(s_{0.7})$), an easy solution for specifying the different configurations is by the guarded automata shown in Figure 5, which defines two operation modes of the state estimator, est_1 and est_2, that correspond to $var(s_{0.7})$ and $var(s_{0.2})$. So of course if we sense in mode $s_{0.7}$ we must estimate with mode est_1 that has the correct configurations for $s_{0.7}$, and if we sense in mode $s_{0.2}$ we must estimate with mode est_2.

3.4.2 Automata Operation

3.4.3 GOAL Tool

3.4.4 Scheduler Prototype

4 Proof of Concept

4.1 Simulations

4.2 Experiment: Vision based controllers for drones

To test our concepts, we will examine the implementation of an autonomously flying quad-rotor in the context of an agriculture case study. Specifically, we

will implement a quad-rotor that flies in corridors and greenhouses by a vision based feedback. The challenge in this case study, from our perspective, is that image processing is a heavy computational task that requires careful scheduling in order to preserve the system predictability and stability.

The current state-of-the-art solution for involving heavy computational tasks such as vision in the control system is simply by adding computational power to the system (use faster processors), usually much more than needed, in order to eliminate the chance of loosing predictability. Some, more conservative, control engineers prefers to isolate the heavy computation from the core control loop by allocating one of the processor's core for that task, or even run the vision processing on a different computer (APM, for example suggests to use image processing by adding a dedicated computer board [**?**]).

In Section **??** we propose an alternative framework for such systems, to be implement in the proposed thesis, that allows to integrate the heavy computations in the control loop in an efficient way that, we believe, will allow fo cutting the costs involved in adding a dedicated boards or in dedicating a computation core.

4.2.1 Sensors

Computer Vision Based Sensor is the module that is responsible for taking a picture or a series of pictures and produce measurements of quantities such as speed and position relative to the environment. In our research we will use the *Computer Vision* in order to detect two dimensional movement in the camera surface (i.e., the drone speed). There are many such algorithms (called optical flow) differing in running times and in accuracy. Usually more invested time leads to more accuracy.

In our case we need to be able to control the *Computer Vision* running time and to have some good knowledge of the solution accuracy in order to achieve optimal state estimation (see Section 3.2). We assume that the *Computer Vision* error is distributed normally, and we will use the error variance as a measure of accuracy. We believe that this is a reasonable assumption because the estimated speed is usually computed as the avarage of of many independent random variables, as follows. Optic flow algorithms usually go by identifying similar regions in consecutive pictures and then averaging the distances that each feature "traveled". Assuming that the error in measurement of each feature is independent of the other errors, we get that the total error is the average of independent random variables. Then, by the law of large numbers, we get that the error should have a normal distribution. We will validate this assumption by experimentation

with different parameters of different algorithms.

In order to control the running time, we will use anytime based algorithms [**?**] and will mainly concentrate on "contract based" vision algorithms proposed by Pant [**?**]. This type of algorithms run until we stop them, and when we stop them they will provide a solution with accuracy that is a monotonic function of the amount of time it was running. That way we can control the solution accuracy by controlling the running time. In our implementation, in order to lower the complexity, we will pre-define few specific "operation modes" of the *Computer Vision* task, that differ by their running time and they are identified by a pair $\langle RunTime, Variance \rangle$ where $Variance$ is the error variance of running time $RunTime$.

5 Testing Environment

5.1 APM Controlled Quad-copter

5.2 Vision based Autonomous In-Door Flying

Computer Vision Based Sensor is the module that is responsible for taking a picture or a series of pictures and produce measurements of quantities such as speed and position relative to the environment. In our research we will use the *Computer Vision* in order to detect two dimensional movement in the camera surface (i.e., the drone speed). There are many such algorithms (called optical flow) differing in running times and in accuracy. Usually more invested time leads to more accuracy.

In our case we need to be able to control the *Computer Vision* running time and to have some good knowledge of the solution accuracy in order to achieve optimal state estimation (see Section 3.2). We assume that the *Computer Vision* error is distributed normally, and we will use the error variance as a measure of accuracy. We believe that this is a reasonable assumption because the estimated speed is usually computed as the avarage of of many independent random variables, as follows. Optic flow algorithms usually go by identifying similar regions in consecutive pictures and then averaging the distances that each feature "traveled". Assuming that the error in measurement of each feature is independent of the other errors, we get that the total error is the average of independent random variables. Then, by the law of large numbers, we get that the error should have a normal distribution. We will validate this assumption by experimentation with different parameters of different algorithms.

In order to control the running time, we will use anytime based algo-

rithms [**?**] and will mainly concentrate on "contract based" vision algorithms proposed by Pant [**?**]. This type of algorithms run until we stop them, and when we stop them they will provide a solution with accuracy that is a monotonic function of the amount of time it was running. That way we can control the solution accuracy by controlling the running time. In our implementation, in order to lower the complexity, we will pre-define few specific "operation modes" of the *Computer Vision* task, that differ by their running time and they are identified by a pair $\langle RunTime, Variance \rangle$ where $Variance$ is the error variance of running time $RunTime$.

5.3 Reference Measurement: Optitrack system

5.4 Test Results

6 Conclusions and Future Work

7 Apendix

7.1 Kalman Filter and State Estimation

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

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

A short brief on *complementary filter*, this is basically combination of two filters (witch are complement each other), in control field it usually refers to *Low-Pass filter* and *High-Pass filter*. Low-pass filter allows the low frequencies signals to pass and filter out the high frequencies signals, is used for signals with high frequency error, e.g accelerometer signal. high-pass filter is its complementary, meaning it filters out the low frequencies signals, is used for signals like gyroscope that have continuous accumulated error.

General complementary filter notation: $\hat{f} = \alpha \cdot f_h + (1-\alpha) \cdot f_l$ s.t. $\alpha \in [0,1]$ if α is significant high, f_l is considered *low-pass*, as rapid value changes less significant and d slow value changes can be accumulated over time, and f_h is considered *high-pass*. In vision base measurements we use the only Low-pass filter: $\hat{x}_k = \hat{x}_{k-1} + \alpha \cdot (y_k - \hat{x}_{k-1})$.

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

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

assumes the true state at time k is evolved from the state at ($k1$) according to:

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

and the measurement at time K is:

$$y_k = Cx_k + v_k$$

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

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

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

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

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

$$\hat{x}_k = \hat{x}_k^- + K_k(y_k - C\hat{x}_k^-)$$

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

$$K_k = \frac{P_k^- C^T}{CP_k^- C^T + R}$$

And the Kalman estimation covariance is:

$$P_k = (I - K_k C)P_k^-$$

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

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

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

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

7.2 TODO

References