High-Assurance Quasi-Synchronous Systems

Robin Larrieu – École Polytechnique
Natarajan Shankar – SRI International

Abstract

The design of a complex cyber-physical system is centered around one or more models of computation (MoCs). These models define the semantic framework within which a network of sensors, controllers, and actuators operate and interact with each other. In this paper, we examine the foundations of a quasi-synchronous model of computation Our version of the quasi-synchronous model is inspired by the Robot Operating System (ROS). It consists of nodes that encapsulate computation and topic channels that are used for communicating between nodes. The nodes execute with a fixed period with possible jitter due to local clock drift and scheduling uncertainties, but are not otherwise synchronized. The channels are implemented through a mailbox semantics. In each execution step, a node reads its incoming mailboxes, applies a next-step operation to update its local state, and writes to all its outgoing mailboxes. The underlying transport mechanism implements the mailbox-to-mailbox data transfer with some bounded latency. Messages can be lost if a mailbox is over-written before it is read. We prove a number of basic theorems that are useful for designing robust high-assurance cyber-physical systems using this simple model of computation. We show that depending on the relative rates of the sender and receiver, there is a bound on the number of consecutive messages that can be lost. By increasing the mailbox queue size to a given bound, message loss can be eliminated. We demonstrate that there is a bounded latency between the time a message is sent and the time that it is processed, which in turn can be used to bound the end-to-end sense-control-actuate latency. We illustrate how these theorems are useful in designing and verifying a correct thermostat-based heating system. Our proofs have been mechanically verified using the Prototype Verification System (PVS).

Contents

1 Introduction

The HACMS[1] project aims to give proved reliability and security for systems based on a ROS (Robot Operating System) architecture. In ROS, we use the abstraction of *nodes* and *topics* to describe the different components and communication channels. At initialization, each node is declared subscriber to some topics and publisher on others. At each step, a node reads messages from the subscribed topics, execute a computation and publish on the topics it is in charge.

Each node is set to execute at a given frequency, but the actual rate to publish messages may vary because of clock bias, or a computing duration depending on the inputs. Since several nodes may execute physically on the same processor, scheduling randomness adds to this imprecision. When a message is published on a topic, it may need a certain delay before it is received by the subscribers. This delay is caused by the nature of the communication link (network/shared memory) and the underlying mailbox system. For these reasons, we cannot ensure a subscriber will have new messages at each step, or that sent messages will be processed by the subscriber (they may be overridden before that). Figure 1 gives examples for these problems.

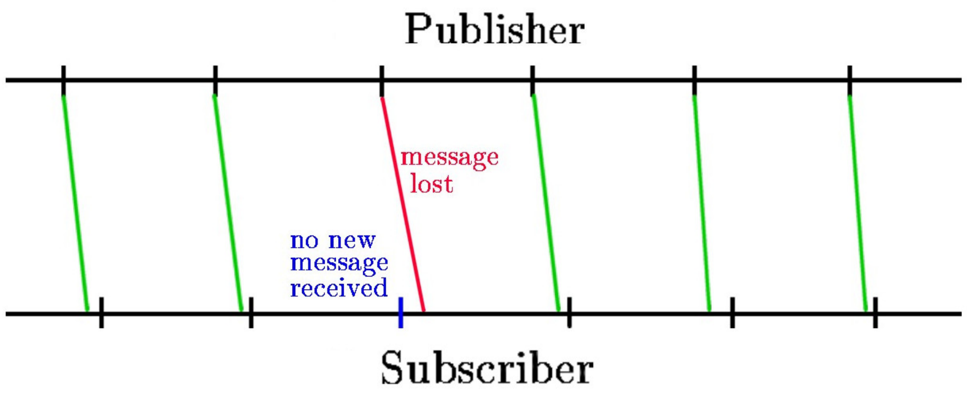

Figure 1: Uncertainty for the messaging system

In this report, we will establish a worst case scenario for this messaging system. For example, knowing that a node will always have available at least one message with a bounded age can be sufficient to assert a control loop maintains a state precisely enough around the desired value. This can also be used for monitoring purposes: if a node detects a behavior outside this worst case scenario, it can raise an alert to a supervisor. By collecting these alerts, the supervisor could detect a compromised/crashed node, or a broken link, and decide how to fix the problem.

In Section 2, we give more details about the ROS architecture and the model used to represent it. In Section 3, we prove low-level properties for the messaging system. In Section 4, we give assurance claims for a simple end-to-end property that a basic control loop maintains a state around a set value.

[1] stands for High Assurance Cyber-Military System

2 The ROS architecture

2.1 Overview of ROS functioning

The aim here is not to give a complete description of ROS (the interested reader could refer to [1]). However, a few notions about ROS abstraction could be needed to understand following sections.

The main purpose of ROS is to provide a framework for distributed computing. Rather than writing one complex program that runs the whole robot, designers prefer writing several simpler programs (potentially running on different computers) in charge of a specific task: one program that uses sensors to scan the environment, one responsible to interact with the operator, another one controlling the wheels ...

Each of these programs is represented by a ROS *node* that can send messages to other nodes. *Topics* are declared to group messages with the same function, and each node can be publisher or subscriber to one or more topics. The ROS master gives information to publishers (IP addresses, shared buffers ...), so when publishing a message on a topic, they can send messages to all subscribers to this topic. Messages received by a subscriber are stored in a queue until they are computed. If the queue is full when a new message is received, the oldest message is thrown away.

2.2 Modeling assumptions

Topic assumptions: To avoid jamming, we require at most one publisher per topic[2]; this is secured in HACMS by message authentication and firewall rules. For simplification purpose, we assume in this report that there is only one subscriber on each topic. This assumption does not lead to a loss of generality: a topic with n subscribers could be seen as n topics with one subscriber. Finally, we assume that the transmission delay for a given message is bounded.

Then, a topic can be characterized by two nodes (the publisher and the subscriber), the maximum transmission delay and the queue size for the subscriber.

Node assumptions: For less CPU usage, HACMS nodes don't run continuously, but are discrete simple tasks executed regularly. Each node is supposed to execute at a given frequency, but the actual rate may vary. The execution could be non periodic, but we assume that the time between two consecutive steps is inside some known interval $[minT, maxT]$ with $minT > 0$. The typical example is when the frequency is known with a certain drift ρ, and the actual instantaneous frequency is in $[(1-\rho).f, (1+\rho).f]$.

We also do the simplifying assumption that the task at each step is executed instantaneously. Actually, in the case of a node that is only a publisher (resp. subscriber)[3], we can use as the execution time the time when it sends (resp. reads) the message, which are atomic operations. In the case of an intermediate node, that is both subscriber and publisher on different topics, the computed duration between these two operations could just be added to the delay of the published topic.

Then a node can be characterized by the upper and lower bound for the pseudo-period between consecutive steps.

[2] no publisher on a topic generates a warning, but is considered non-critical
[3] for example a sensor (resp. actuator)

Notations: Events are described by functions from $\mathbb{N}$ (the system runs forever) to $\mathbb{R}^+$ that give the time when this event happens. For example, if e is the function used to describe the executions of some node, $e(n)$ is the time when the nth execution occurs.

Definition 2.1. : ***Node execution***

Each node N is defined by its minimal $minT(N)$ and maximal $maxT(N)$ pseudo-period. Then an execution of N is any function e such that

$$e(0) = 0 \quad \text{and} \quad \forall n \in \mathbb{N},\, e(n) + minT(N) \leq e(n+1) \leq e(n) + maxT(N)$$

The assumption $e(0) = 0$ could seem too strong because it implies all nodes start exactly at the same time (which is quite unrealistic). First, having a common origin simplifies inductions, that are widely used in proofs. Second, we could prove that for any initial time shift Δ, with an interval $[minT(N), maxT(N)]$ as small as desired, there exist executions for N and N' such that $e(n) - e'(m) = \Delta$ for some n and m.

Definition 2.2. : ***Message reception***

Let T be a topic with a maximum transmission delay of D, and let s be an execution of its publisher S (associated to the event a message is sent*). A reception of these messages is any function r such that*

$$\forall n \in \mathbb{N},\, s(n) \leq r(n) \leq s(n) + D \quad \text{and} \quad r \textit{ is injective}$$

The hypothesis that r is injective means that two different messages cannot be received exactly at the same time. This is linked to hardware limitations, and is important to represent the queue: if messages m and n are received exactly at the same time, which one comes before the other in the queue?

Note that this definition does not assume the channel conserves the order of messages. We could have $r(k) > r(k+1)$ and therefore, the message received just after message n is not necessarily message $n+1$. However, $\{n \in \mathbb{N},\, r(n) \leq t\}$ is finite, so we can define a function *next* such that

$$\forall n \in \mathbb{N}, \quad r(n) < r(next(n)) \text{ and } (\forall k, r(k) \leq r(n) \text{ or } r(k) \geq r(next(n)))$$

and the iterated function $Nth\_next(N, n)$.

Definition 2.3. : ***Processed message***

Let T a topic with a subscribers queue size of L. Let s and c be an execution of its publisher and subscriber and r be a reception of sent messages.

By definition, we have $\left|\{k, r(n) \leq r(k) < r(Nth\_next(L, n))\}\right| = L$. Then, message n is in the queue at time t iff[4] $r(n) \leq t < r(Nth\_next(L, n))$

A message n is said to be processed *if and only if*

$$\exists k \in \mathbb{N}, \quad r(n) \leq c(k) < r(Nth\_next(L, n))$$

The message is said to be *lost* or *dropped* otherwise

[4] Actually, depending on the implementation, the message could be removed from the queue during a computation, but before $r(Nth\_next(L, n))$. This does not affect the definition.

3 Low-level messaging system

In this section, we consider a topic T, characterized by a publisher P, a subscriber S, a maximum transmission delay D, and a queue length L.

For $n \in \mathbb{N}$, we note $s(n)$ and $r(n)$ respectively the time when the n'th message is sent and received. We also note $c(n)$ the time when the subscriber executes its n'th computation.

3.1 Latency

For a processed message, we define its latency as the duration between the time it is sent and the time it is computed for the first time (see Figure 2):

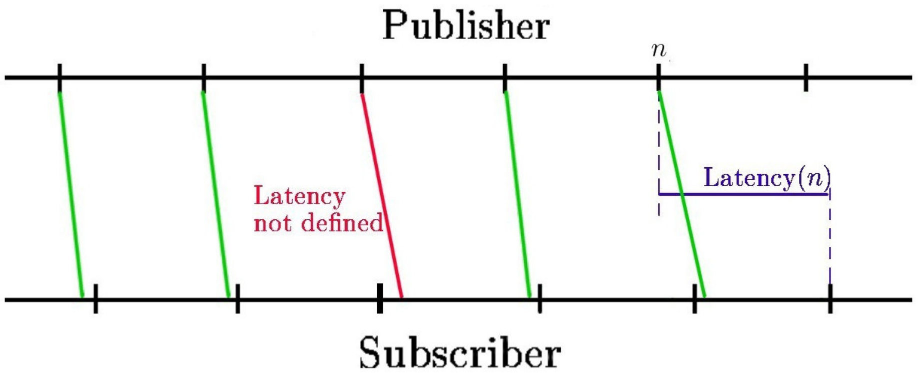

Figure 2: Latency definition

Definition 3.1. : ***Latency***

$$Latency(n) = c\Big(min(\{k \in \mathbb{N} | r(n) \leq c(k)\})\Big) - s(n)$$

Since $\forall k, c(k+1) \leq c(k) + maxT(P)$, we have $c\big(min(\{k \in \mathbb{N} | r(n) \leq c(k)\})\big) - r(n) \leq maxT(P)$. Therefore,

Theorem 3.1. : ***Latency bound***

$$Latency(n) \leq maxT(P) + D$$

The latency is only defined for a processed message, so this gives few information for assurance properties. However, it can be used to detect a clock failure or a certain type of attack: assume for example that the subscriber reads a new message, but the delay since the messages timestamps exceed the latency bound. It could be that the clock of one of the nodes drifted significantly from real time, or it may mean that the message had been recorded and was resent later by an attacker.

3.2 Overtaking

The model defined in section 2.2, allows an older message to be delivered after a more recent one. We may want to avoid this even if the mailbox system alone doesn't assert that this property holds. We give here a simple condition that asserts that messages are received in the order they are sent.

This may give better bounds to assert other properties are satisfied. By choosing good nodes parameters, a developer can also secure the order of received messages and rely on this to prove correctness of an application.

We have $r(n) \leq s(n) + D$, $s(n) + minT(P) \leq s(n+1)$ and $s(n+1) \leq r(n+1)$. Therefore,

Theorem 3.2. : ***Overtaking***

$$D < minT(P) \quad \Longrightarrow \quad \forall n \in \mathbb{N}, \quad r(n) < r(n+1)$$

3.3 Number of lost messages

As we saw before, messages can be overwritten in the queue before they are actually computed by the subscriber, which means these messages are never processed. However, we can ensure the subscriber don't misses too many consecutive messages

3.3.1 Upper bound for the number of consecutive lost messages

Definition 3.2. : ***Consecutive lost messages***
Formally, we can define the property "the subscriber never misses N consecutive messages" by

$$\forall k \in \mathbb{N}, \exists l < N, \quad \text{message } k + l \text{ is processed}$$

First, we have this fundamental lemma:

Lemma 3.3.1. *If $c(n) \geq t + D + maxT(P)$, there exists a message k with $t < r(k) \leq c(n)$ that is processed.*

Proof. $r(0) \leq D < c(n)$. Let m be the maximum of the $l \in \mathbb{N}$ such that $r(l) \leq c(n)$. Since $r(m+1) > c(n)$, we have $t < r(m) \leq c(n)$.

The set $S = \{l \in \mathbb{N}, t < r(l) \leq c(n)\}$ is finite and nonempty. There exists a $k \in S$ such that $\forall l \in S, r(l) \leq r(k)$. By construction, this k solves the problem because no message is received between $r(k)$ and the next execution of the subscriber. □

Basic inequalities transformations give this result:

Lemma 3.3.2.

$$r(m) > s(n) + D - minT(P) \implies n \leq m$$
$$r(m) < s(n) + minT(P) \implies m \leq n$$

Theorem 3.3. : ***Consecutive lost messages***

Assume $N.minT(P) > 2.D + maxT(S) + maxT(P) - minT(P)$. Then the subscriber never misses N consecutive messages.

Proof. According to definition 3.2, given $k \in \mathbb{N}$, we prove there exists a $l < N$ such that message $k + l$ is processed.

The subscriber executes at least once in every interval of time of length $maxT(S)$. In particular,

$$\exists n \in \mathbb{N}, \quad \begin{cases} s(k) + 2.D + maxT(P) - minT(P) < c(n) \\ c(n) \leq s(k) + 2.D + maxT(P) - minT(P) + maxT(S) \end{cases}$$

Lemma 3.3.1 gives $\exists l \in \mathbb{N}, s(k) + D - minT(P) < r(l) \leq c(n)$ such that the message l is processed. Since $s(k + N - 1) \geq (N-1).minT(P)$, with the given condition on N, $c(n) < s(k + N - 1) + minT(P)$.

With lemma 3.3.2, we have $k \leq l \leq k + N - 1$, and l is processed by construction. □

Case without overtaking : Assume now that $\forall k \in \mathbb{N},\ r(k) < r(k+1)$. In that case, with $m = max(k \in nat, r(k) \leq c(n))$ we have $r(k) \leq c(n) \implies r(k) \leq r(m)$ which ensures message m is processed.

With $N.minT(P) > D + maxT(S)$, we have $\exists n \in \mathbb{N}, s(k) + D \leq c(n) < s(k+N)$, then $r(k) \leq c(n) < r(k+N)$. We deduce message m is processed with $k \leq m < k+N$.

Theorem 3.4. : ***Consecutive lost messages – no overtaking***
Assume $N.minT(P) > D + maxT(S)$ and $\forall k \in \mathbb{N},\ r(k) < r(k+1)$. Then the subscriber never misses N consecutive messages.

3.3.2 Influence of the queue length

As one can expect, increasing the message queue size leads to a lower message loss rate. For more comprehension, we first prove this in the case without overtaking. Then, we give an idea of the proof in the general case.

Assume we have $\forall k \in \mathbb{N},\ r(k) < r(k+1)$ and we never drop N $(N > 1)$ messages with a queue length of L. Given $k \in \mathbb{N}$, there exists $l < N$ such that message $k+l$ is processed. If $l < N-1$, the result is proved. If $l = N-1$, it means the message $k+N-1$ was in the queue when some computation $c(n)$ occurred. With a queue length of $L+1$, the message $k+N-2$ was in the queue when $c(n)$ occurred, which means it was processed.

In the general case, we assume the property "never miss N consecutive messages" holds whatever r is as soon as the constraints with s and D are respected (theorem 3.3 gives this assurance). Like in the previous proof, we assume $k+N-1$ is processed and none of the $k+l$ with $l < N-1$ is. Among non processed messages received before $r(k+N-1)$, let m be the one received last. With a queue length of $L+1$, m is processed with the same argument as before. The difficult part is to prove that this m exists and must be one of the $k+l$, $l < N-1$. For this, we construct an other r', consistent with the constraints on s and D but with N consecutive messages lost (which is a contradiction).

By a simple induction, we then get the following result:

Theorem 3.5. *Let N satisfies conditions of either theorem 3.3 or theorem 3.4. Let $m < N$ and assume $L > m$. Then the subscriber never miss $N - m$ consecutive messages.*
In particular, if $L \geq N$, no message is lost.

3.4 Age of processed messages

In this section, we prove that at each computation, the subscriber gets (from a newly received message or with a backup from previous computation) a reasonably recent message.

This is essentially this bound on the age of processed messages that will be used in next sections to prove end-to-end properties. It is also helpful to detect errors: when the latest available message to the subscriber is older than the bound, it can raise a timeout flag. By collecting these flags, a monitor could guess whether the publisher node crashed or there is a network failure.

3.4.1 Definition and basic properties

The aim is to model the following behavior (see Figure 3): At each computation, if messages are available in the queue, the node chooses the most recent one and saves it to give a backup solution for next computation. If not, it uses the backup given by previous step (while no message has been received, and no backup is available, a default value – set at initialization time – is used)

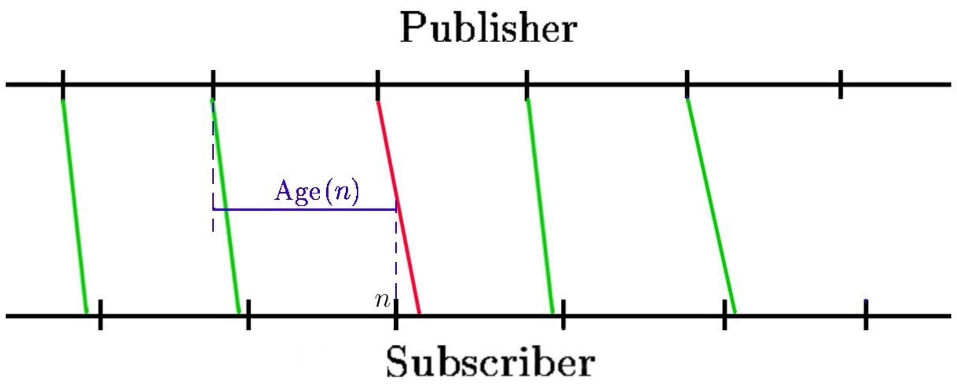

Figure 3: Age definition

Definition 3.3. : ***Age for the most recent available message***
For $n \neq 0$, we define the (finite, but maybe empty) set of messages computed at step n by

$$PL(n) = \left\{k \in \mathbb{N}, c(n-1) < r(k) \leq c(n) \wedge \text{message } k \text{ is processed}\right\}$$

Then, the age of the most recent available message can be defined by the recursive function

```
Age($n$) = {
   if $n = 0$ then 0
   else {if $PL(n) = \emptyset$
      then {Age($n-1$)+$c(n) - c(n-1)$}
      else {$c(n) - s(max(PL(n))$}
}}
```

Lemma 3.4.1. *A simple induction over n gives* Age$(n)$$\leq c(n)$.
Also, with the same argument as in the proof of lemma 3.3.1, we have:

$$PL(n) = \emptyset \iff \{k \in \mathbb{N}, c(n-1) < r(k) \leq c(n)\} = \emptyset$$

3.4.2 Upper bound in worst case scenario

Theorem 3.6. : ***Age bound***

$$\texttt{Age}(n) < 2.D + maxT(P)$$

Proof. First, we see by a simple induction over n that:

$$r(k) \leq c(n) \implies \texttt{Age}(n) \leq c(n) - r(k) + D$$

(if $PL(n) = \emptyset$, lemma 3.4.1 gives $r(k) \leq c(n-1)$. Otherwise, using the definition of a processed message, we either have $r(k) \leq r(max(PL(n)))$ or $k \leq max(PL(n))$. In both cases, $s(max(PL(n))) \geq r(k) - D$)

In the case $c(n) \leq D + maxT(P)$, lemma 3.4.1 gives the result. Otherwise, by lemma 3.3.1, there exists a message k such that $c(n) - D - maxT(P) < r(k) \leq c(n)$, which proves the theorem with the property above. $\square$

When messages are delivered in the same order they were sent, we get a better bound:

Theorem 3.7. : ***Age bound – no overtaking***

$$\forall k \in \mathbb{N},\ r(k) < r(k+1) \quad \Longrightarrow \quad \texttt{Age}(n) < D + maxT(P)$$

Proof. When no overtaking is possible, the message $m(n) = max(\{k \in \mathbb{N}, r(k) \leq c(n)\})$ (assuming the considered set is non empty) is processed.

Since $r(m(n)+1) \leq s(m(n)) + D + maxT(P)$, we have

$$c(n) - D - maxT(P) < s(m(n)) \leq c(n)$$

$r(m(n)) \leq c(n-1) \implies m(n) = m(n-1)$. Therefore, we get by induction

$$\texttt{Age}(n) = \begin{cases} c(n) - m(n) & \text{if } r(0) \leq c(n) \\ c(n) & \text{if } r(0) > c(n) \text{, which means } c(n) < D \end{cases}$$

$\square$

4 Assurance claim for the plant controller

4.1 Physical model and hypothesis

In this section, we look at a control loop with a plant (external state we wish to maintain), a sensor which measures the plant state, a controller and an actuator. The actuator can be *on* or *off*, which gives two different progression modes for the plant. The controller sets the actuator *on* or *off* according to the input from the sensor and the settings from an operator. Said operator can decide to enable or disable control, and can set preferences for the value to maintain the state around. This is summarized in Figure 4. For now, we assume the operator enables the control (no interest otherwise) and doesn't change the settings.

We consider the plant state as a real function of the time. The aim for the plant controller is to bring this state between two bounds and to maintain it there. A first example could be a thermostat: we want to maintain a room temperature around a comfortable value by switching a heater *on/off* when needed. We can also imagine maintaining a correct speed for a vehicle by giving or not power to the engine.

For better comprehension, we will use the vocabulary of the thermostat example in the rest of this section (but the model is actually more general). We name $\theta(t)$ the temperature at time t, and τ and Γ the upper and lower bounds in which we want to maintain the temperature. We consider that the room leaks energy, and that the heater is powerful enough to increase the room temperature. Typically, while the heater is off, the room temperature decreases with a rate of at most μ, and it increases with a rate of at least $\nu - \mu$ ($\nu > \mu$) while the heater is on. More precisely, considering the temperature is piecewise differentiable,

$$\begin{cases} 0 < \nu - \mu < \dot{\theta} < \Omega & \text{If the heater is } \textit{on} \\ -\mu < \dot{\theta} < -\omega < 0 & \text{If the heater is } \textit{off} \end{cases}$$

Assuming the outside temperature is lower than the inside temperature (who needs a heater otherwise?), this model is quite realistic, at least in a reasonable inside temperature range.

To represent real-life devices, we allow the sensor to be slightly inaccurate. Therefore, if the actual temperature is θ, the sensed temperature will be σ with $\theta - \epsilon \leq \sigma \leq \theta + \epsilon$

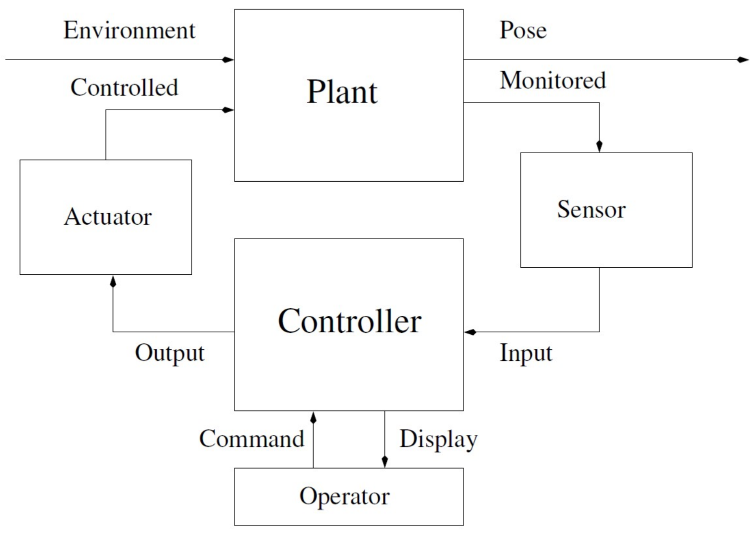

Figure 4: The plant controller model

4.2 Controller strategy

The intuitive way is to set the heater *on* when the temperature is too low, and set it *off* when the temperature is too high. We define two trigger temperatures Δ_1 and Δ_2 with $\tau < \Delta_1 \leq \Delta_2 < \Gamma$. When the measured temperature is below Δ_1, the controller switches the heater to *on*, and when the measured temperature is above Δ_2, the controller switches the heater to *off*. The behavior when the temperature is between Δ_1 and Δ_2 doesn't really matter, so we decide for example to resend previous command.

Of course, we do not use τ and Γ as triggers because of the sensor inaccuracy and because of the potential message latency.

This strategy could be implemented as follows: The controller keeps a binary setting, initialized to some default value. At each step, if a new message have been received, the controller chooses the most recent one and look at the sensed temperature σ inside. If $\sigma < \Delta_1$, the setting is turned to *on*. If $\sigma > \Delta_2$, the setting is set to *off*. Otherwise, no change is applied to the setting. After this test, the current setting is sent to the actuator. (using the previous setting also in case no new message have been received is equivalent to use a backup from the previous computation)

4.3 Correctness proof

To prove correctness of the system, we need to apply this physical model to our logical architecture. We name S, C and A the nodes for the sensor, the controller and the actuator respectively. We note $s(n)$, $c(n)$ and $a(n)$ the time when their respective nth computation occurs. Let *Input* and *Output* be the topics for the sensed temperature and control commands respectively (that is, S publishes to *Input*, C subscribes to *Input* and publish to *Output*, and A subscribes to *Output*). Let finally $r_1(n)$ and $r_2(n)$ be the times when the nth message in these topics is delivered.

Definition 4.1. : ***Plant controller characteristic time***

For a topic T, we note $MA(T)$ some bound for the age of latest available message[5] *at each step (given for example by theorem 3.6 or theorem 3.7).*

The characteristic time for the plant controller is the quantity

$$\lambda = MA(Input) + MA(Output) + maxT(A)$$

Theorem 4.1. : ***State stability***

Assume $\theta(t) \geq \Delta_1 - \epsilon \geq \tau + \lambda.\mu$ for some time t. Then, $\theta(t)$ never drops below τ after that: $\forall t' \geq t$, $\theta(t') > \tau$

Symmetrically, assume $\theta(t) \leq \Delta_2 + \epsilon \leq \Gamma - \lambda.\Omega$ for some time t. Then, $\theta(t)$ never becomes above Γ after that: $\forall t' \geq t$, $\theta(t') < \Gamma$

Proof. Let $t' \leq t$ and $g = glb(\{u \in \mathbb{R}, t \leq u \leq t'$ and $\theta(u) \leq \Delta_1 - \epsilon\})$. By definition, $\theta(g) = \Delta_1 - \epsilon$ and $g < u \leq t' \implies \theta(u) < \Delta_1 - \epsilon$.

Given $\dot{\theta} > -\mu$, if $t' \leq g + \lambda$, the result is proved. Otherwise, we prove $\theta(t') \geq \theta(g + \lambda) > \tau$, because the actuator is always *on* in the time interval $[g + \lambda, t']$:

Let $u \geq g + \lambda$. The actuator state at time u was set at the latest computation $a(n)$ of A. $a(n + 1) \leq a(n) + maxT(A)$ so we have $a(n) > g + MA(Input) + MA(Output)$. The corresponding command from the thermostat was issued at some $c(k)$ and by definition of $MA(Output)$, we have $c(k) > g + MA(Input)$. This command was computed by comparing a measured temperature with Δ_1 and Δ_2. Let $s(l)$ the execution that measured this temperature. We have $s(l) > g$ (by definition of $MA(Input)$), which means $\theta(s(l)) < \Delta_1 - \epsilon$. Therefore, the sensed temperature was $\sigma(s(l)) < \Delta_1$ □

Theorem 4.2. : ***State convergence***

Assume $\theta(t) < \Delta_1 - \epsilon$. Then we have $\theta(t') \geq \Delta_1 - \epsilon$ for some t' with

$$t' < t + \lambda + \frac{\Delta_1 - \epsilon - \theta(t) + \lambda.\mu}{\nu - \mu}$$

Symmetrically, assume $\theta(t) > \Delta_2 + \epsilon$. Then we have $\theta(t') \leq \Delta_2 + \epsilon$ for some t' with

$$t' < t + \lambda + \frac{\theta(t) + \lambda.\Omega - \Delta_2 - \epsilon}{\omega}$$

Proof. Assume for all u with $t \leq u \leq T = t+\lambda+\frac{\Delta_1-\epsilon-\theta(t)+\lambda.\mu}{\nu-\mu}$, we have $\theta(u) < \Delta_1 - \epsilon$. With the same argument as in previous theorem, the heater is always on in the time interval $[t + \lambda, T]$.

During this interval, the temperature increases at a minimum rate of $\nu - \mu$. Therefore, $\theta(T) > \theta(t+\lambda)+(\Delta_1-\epsilon-\theta(t)+\lambda.\mu)$. But $\theta(t+\lambda) > \theta(t)-\lambda.\mu$, which leads to a contradiction □

References

[1] Jason M. O'Kane. *A Gentle Introduction to ROS*. CreateSpace Independent Publishing Platform, 10 2013.

[5] see definition 3.3