15-424: Foundations of Cyber-Physical Systems

Lecture Notes on Reactions & Delays

André Platzer

Carnegie Mellon University
Lecture 9

1 Introduction

Lecture 7 on Control Loops & Invariants explained the central proof principle for loops based on induction using invariants. Lecture 8 on Events & Responses studied the important feedback mechanism of event-driven control and made crucial use of invariants for rigorously reasoning about event-driven control loops. Those invariants uncovered important subtleties with events that could be easily missed. In Lecture 8 on Events & Responses, we, in fact, already noticed these subtleties thanks to our "safety first" approach to CPS design, which guided us to exercise the scrutiny of Cartesian Doubt on the CPS model before even beginning a proof.

But, even if the final answer for the event-driven controller for ping pong balls was rather clear and systematic, event-driven control had an unpleasantly large number of modeling subtleties in store for us. Furthermore, event-driven control has a rather high level of abstraction, because it assumes that all events would be detected perfectly and right away in continuous sensing. However, the event-driven model had $x \leq 5$ as a hard limit in the evolution domain constraint to ensure that the event $4 \leq x \leq 5$ would never be missed as the ball is rushing upwards.

As soon as we want to implement such an event detection, it becomes clear that real controller implementations can only perform discrete sensing, i.e. checking sensor data every once in a while at certain discrete points in time, whenever the measurement comes from the sensor and the controller has a chance to run. Most controller implementations would, thus, only end up checking for an event every once in a while, whenever the controller happens to run, rather than permanently as event-driven controllers pretend.

Today's lecture focuses on the second important paradigm for making cyber interface with physics to form cyber-physical systems. The paradigm of *time-triggered control*,

which uses periodic actions to affect the behavior of the system at certain frequencies. This is to be contrasted with the paradigm from Lecture 8 on Events & Responses of *event-driven control*, where responses to events dominate the behavior of the system and an action is taken whenever one of the events is observed. Both paradigms play an equally important role in classical embedded systems and both paradigms fall out naturally from an understanding of the hybrid program principle for CPS.

These lecture notes are loosely based on [Pla12, Pla10].

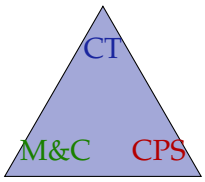
Based on the understanding of loops from Lecture 7 on Loops & Invariants, the most important learning goals of this lecture are:

Modeling and Control: Today's lecture provides a number of crucial lessons for modeling CPS and designing their controls. We develop an understanding of time-triggered control, which is an important design paradigm for control loops in CPS. This lecture studies ways of developing models and controls corresponding to this feedback mechanism, which will turn out to be surprisingly subtle to control. Knowing and contrasting both event-driven and time-triggered feedback mechanisms helps with identifying relevant dynamical aspects in CPS coming from events and reaction delays. Today's lecture focuses on CPS models assuming discrete sensing, i.e. sensing at (nondeterministic) discrete points in time.

Computational Thinking: This lecture uses the rigorous reasoning approach from Lecture 5 on Dynamical Systems & Dynamic Axioms and Lecture 7 on Loops & Control to study CPS models with time-triggered control. As a running example, the lecture continues to develop the extension from bouncing balls to ping pong balls, now using time-triggered control. We again add control decisions to the bouncing ball, turning it into a ping pong ball, which retains the intuitive simplicity of the bouncing ball, while enabling us to develop generalizable lessons about how to design time-triggered control systems correctly. The lecture will also crucially study invariants and show a development of the powerful technique of design-by-invariant in a concrete example. While the lecture could hardly claim showing how to verify CPS models of appropriate scale, the basics laid in this lecture definitely carry significance for numerous practical applications.

CPS Skills: This lecture develops an understanding for the semantics of time-triggered control. This understanding of the semantics will also guide our intuition of the operational effects of time-triggered control and especially the impact it has on finding correct control constraints. Finally, the lecture studies some aspects of higher-level model-predictive control, which will be followed up on later in the course.

using loop invariants
design time-triggered control
design-by-invariant



modeling CPS
designing controls
time-triggered control
reaction delays
discrete sensing

semantics of time-triggered control
operational effect
finding control constraints
model-predictive control

2 Delays in Control

Event-driven control is a useful and intuitive model matching our expectation of having controllers react in response to certain critical conditions or events that necessitate intervention by the controller. Yet, one of its difficulties is that event-driven control with its continuous sensing assumption can be hard or impossible to implement in reality. On a higher level of abstraction, it is very intuitive to design controllers that react to certain events and change the control actuation in response to what events have happened. Closer to the implementation, this turns out to be difficult, because actual computer control algorithms do not actually run all the time, only sporadically every once in a while, albeit sometimes very often. Implementing event-driven control faithfully would, in principle, requires permanent continuous monitoring of the state to check whether an event has happened. That is not particularly realistic, because fresh sensor data will only be available every once in a while, and controller implementations will only run at certain discrete points in time causing delays in processing, and because actuators may sometimes take quite some time to get going. Think of the reaction time it takes you to turn the insight "I want to hit this ping pong ball there" into action so that your ping pong paddle will actually hit the ping pong ball.

Back to the drawing desk. Let us reconsider the original $\mathsf{d}\mathcal{L}$ formula (1) for the ping pong ball (Fig. 1) that we started out from for designing the event-driven version in Lecture 8 on Events & Responses.

$$\begin{aligned} &0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\ &\quad \big[(x' = v, v' = -g \,\&\, x \geq 0; \\ &\qquad \texttt{if}(x = 0)\, v := -cv \,\texttt{else}\, \texttt{if}(4 \leq x \leq 5)\, v := -fv)^*\big](0 \leq x \leq 5) \end{aligned} \tag{1}$$

This simplistic formula (1) turned out not to be valid, because its differential equation was not guaranteed to be interrupted when the event $4 \leq x \leq 5$ happens. Consequently, (1) needs some other evolution domain constraint to make sure all continuous

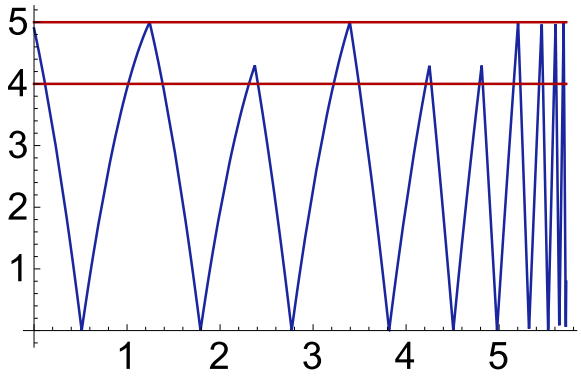


Figure 1: Sample trajectory of a ping pong ball (plotted as position over time) with the indicated ping pong paddle actuation range, sometimes actuating early, sometimes late

evolutions are stopped at some point for the control to have a chance to react to situation changes. Yet, it should not be something like $\ldots \& x \leq 5$ as in Lecture 8 on Events & Responses, because continuously monitoring for $x \leq 5$ requires permanent continuous sensing of the height, which is difficult to implement.

Note 1 (Physical versus controller events)**.** *Observe that the event $x = 0$ in the (physics) controller as well as the (physics) evolution domain constraint $x \geq 0$ for detecting the event $x = 0$ are perfectly justified in bouncing ball and ping pong ball models, because both represent physics. And physics is very well capable of keeping a ball above the ground, no matter how much checking for $x = 0$ it takes to make that happen. It is just in our controller code that we need to exercise care when modeling events and their reactions, because the controller implementations will not have the privilege that physics possesses of running all the time. Cyber happens every once in a while (even if it may run quickly), while physics happens all the time.*

How else could the continuous evolution of physics be interrupted to make sure the controller actually runs? By bounding the amount of time that physics is allowed to evolve before running the controller again. Before we can talk about time, the model needs to be changed to include a variable, say t, that reflects the progress of time with a differential equation $t' = 1$.

$$\begin{aligned} &0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\ &\quad \big[(x' = v, v' = -g, \boldsymbol{t' = 1} \,\&\, x \geq 0 \wedge \boldsymbol{t \leq 1}; \\ &\qquad \texttt{if}(x = 0)\, v := -cv \,\texttt{else}\, \texttt{if}(4 \leq x \leq 5)\, v := -fv\big)^*\big](0 \leq x \leq 5) \end{aligned} \tag{2}$$

In order to bound time by 1, the evolution domain now includes $\ldots \& t \leq 1$ and declares that the clock variable t evolves with time as $t' = 1$. Oops, that does not actually quite do it, because the HP in (2) restricts the evolution of the system so that it will never ever

evolve beyond time 1, no matter how often the loop repeats. That is not what we meant to say. Rather we wanted the duration of each individual continuous evolution limited to at most one second. The trick is to reset the clock t to zero before the continuous evolution starts:

$$\begin{aligned}0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\ \big[\big(\boldsymbol{t := 0}; (x' = v, v' = -g, t' = 1 \,\&\, x \geq 0 \wedge t \leq 1); \\ \texttt{if}(x = 0)\, v := -cv \,\texttt{else}\, \texttt{if}(4 \leq x \leq 5)\, v := -fv\big)^*\big](0 \leq x \leq 5)\end{aligned} \tag{3}$$

In order to bound time by 1, the evolution domain now includes $\ldots \,\&\, t \leq 1$ and the variable t is reset to 0 by $t := 0$ right before the differential equation. Hence, t represents a local clock measuring how long the evolution of the differential equation was. Its bound of 1 ensures that physics gives the controller a chance to react at least once per second. The system could very well stop the continuous evolution more often and earlier, because there is no lower bound on t in (3). Also see Exercise 1.

Before going any further, let's take a step back to notice an annoyance in the way the control in (3) was written. It is written in the style that the original bouncing ball and the event-driven ping pong ball were phrased: continuous dynamics followed by control. That has the unfortunate effect that (3) lets physics happen before control does anything, which is not a very safe start. In other words, the initial condition would have to be modified to assume the initial control was fine. That is a nuisance duplicating part of the control into the assumptions on the initial state. Instead, let's switch the statements around to make sure control always happens before physics does anything.

$$\begin{aligned}0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge g > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\ \big[\big(\texttt{if}(x = 0)\, v := -cv \,\texttt{else}\, \texttt{if}(4 \leq x \leq 5)\, v := -fv; \\ t := 0; (x' = v, v' = -g, t' = 1 \,\&\, x \geq 0 \wedge t \leq 1)\big)^*\big](0 \leq x \leq 5)\end{aligned} \tag{4}$$

Now that dℒ formula (4) has an upper bound on the time it takes between two subsequent control actions, is it valid? If so, which invariant can be used to prove it? If not, which counterexample shows its invalidity?

Before you read on, see if you can find the answer for yourself.

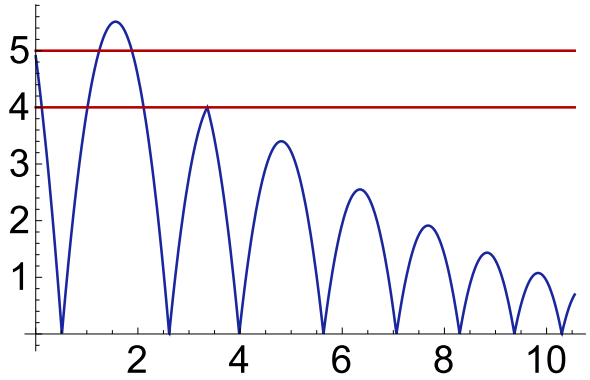


Figure 2: Sample trajectory of a time-triggered ping pong ball (as position over time), missing the first event

Even though (4) ensures a bound on how long it may take at most until the controller inspects the state and reacts, there is still a fundamental issue with (4). We can try to prove (4) and inspect the non-provable cases in the proof to find out what the issue is. The controller of (4) runs at least after one second (hence at least once per second) and then checks whether $4 \leq x \leq 5$. But if $4 \leq x \leq 5$ was not true when the controller ran last, there is no guarantee that it will be true when the controller runs next. In fact, the ball might very well have been at $x = 3$ at the last controller run, then evolved continuously to $x = 6$ within a second and so missed the event $4 \leq x \leq 5$ that it was supposed to detect (Exercise 2); see Fig. 2. Worse than that, the ping pong ball has then already become unsafe.

For illustration, driving a car would be similarly unsafe if you would only open your eyes once a second and monitor whether there is a car right in front of you. Too many things could have happened in between that should have prompted you to brake.

Note 2 (Delays may miss events)**.** *Delays in controller reactions may cause events to be missed that they were supposed to monitor. When that happens, there is a discrepancy between an event-driven understanding of a CPS and the real time-triggered implementation. That happens especially for slow controllers monitoring small regions of a fast moving system. This relationship deserves special attention to make sure the impact of delays on a system controller cannot make it unsafe.*

It is often a good idea to first understand and verify an event-driven design of a CPS controller and then refine it to a time-triggered controller to analyze and verify that CPS in light of its reaction time. Discrepancies in this analysis hint at problems that event-driven designs will likely experience at runtime and they indicate a poor event abstraction.

How can this problem of (4) be solved? How can the CPS model make sure the controller does not miss its time to take action? Waiting until $4 \leq x \leq 5$ holds true is not guaranteed to be the right course of action for the controller.

Before you read on, see if you can find the answer for yourself.

The problem with (4) is that its controller is unaware of its own delay. It does not take into account how the ping pong ball could have moved further before it gets a chance to react next. If the ball is already close to the ping pong paddle's intended range of actuation, then the controller had better take action already if it is not sure whether next time will still be fine.

The controller would be in trouble if $x > 5$ might already hold in its next control cycle after the continuous evolution, which will be outside the operating range of the ping pong paddle (and already unsafe). Due to the evolution domain constraint, the continuous evolution can take at most 1 time unit, after which the ball will be at position $x + v - \frac{g}{2}$ as Lecture 4 already showed by solving the differential equation. Choosing gravity $g = 1$ to simplify the math, the controller would be in trouble in the next control cycle after 1 second which would take the ball to position $x + v - \frac{1}{2} > 5$ if $x > 5\frac{1}{2} - v$ holds now.

The idea is to make the controller now act based on how it estimates the state might have evolved until the next control cycle (this is a very simple example of model-predictive control). Lecture 8 on Events & Responses already discovered for the event-driven case that the controller only wants to trigger action if the ball is flying up, not if it is already flying down. Thus, making (4) aware of the future in this way leads to:

$$\begin{array}{l} 0 \leq x \wedge x \leq 5 \wedge v \leq 0 \wedge \boldsymbol{g = 1} > 0 \wedge 1 \geq c \geq 0 \wedge f \geq 0 \rightarrow \\ \quad \big[\big(\texttt{if}(x = 0)\, v := -cv\, \texttt{else if}((\boldsymbol{x > 5\frac{1}{2} - v}) \wedge \boldsymbol{v \geq 0})\, v := -fv; \\ \qquad t := 0; (x' = v, v' = -g, t' = 1 \,\&\, x \geq 0 \wedge t \leq 1)\big)^*\big](0 \leq x \leq 5) \end{array} \tag{5}$$

Is conjecture (5) about its future-aware controller valid? If so, which invariant can be used to prove it? If not, which counterexample shows its invalidity?

Before you read on, see if you can find the answer for yourself.

The controller in formula (5) has been designed based on the prediction that the future may evolve for 1 time unit. If an action will no longer be possible in 1 time unit, because the event $x \leq 5$ has passed in that future time instant, then the controller in (5) takes action right now already. The issue with that, however, is that there is no guarantee at all that the ping pong ball will fly for exactly 1 time unit before the controller is asked to act again (and the postcondition is checked). The controller in (5) checks whether the ping pong ball could be too far up after one time unit and does not intervene unless that is the case. Yet, what if the ball only flies for $\frac{1}{2}$ time units? Clearly, if the ball will be safe after 1 time unit, which is what the controller in (5) checks, it will also be safe after just $\frac{1}{2}$ time unit, right?

Before you read on, see if you can find the answer for yourself.

Wrong! The ball may well be below 5 after 1 time unit but still could have been above 5 in between the current point of time and the time that is 1 time unit from now. Then the safety of the controller will be a mere rope of sand, because controller will have a false sense of safety after having checked what happens 1 time unit from now, ignoring whether it was safe until then. Such trajectories are shown in Fig. 3 from the same initial state and the same controller with different sampling periods. Such a bouncing ball would not be safe if it has been above 5 in between two sampling points.

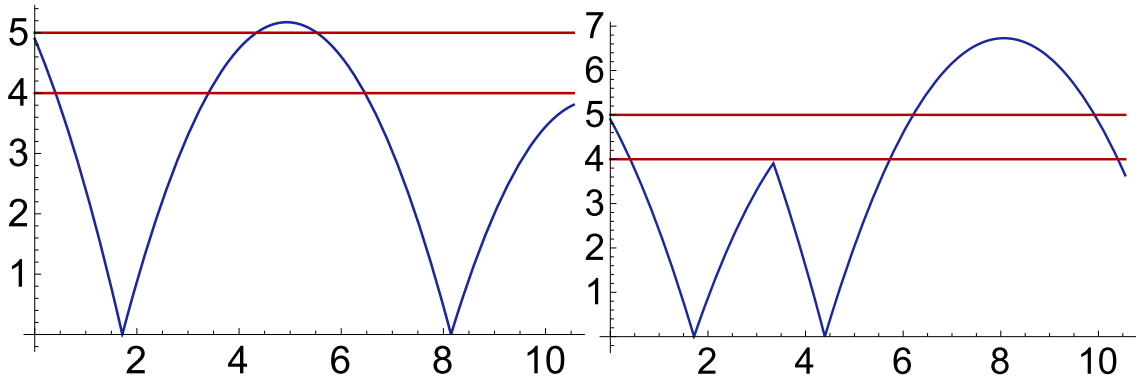


Figure 3: Sample trajectory of a time-triggered ping pong ball (as position over time), missing different events with different sampling periods

In order to get to the bottom of this, recall the invariant for the bouncing ball identified in Lecture 4 on Safety & Contracts and then used in Lecture 7 on Loops & Invariants to prove safety of the bouncing ball:

$$2gx = 2gH - v^2 \land x \geq 0 \land c = 1 \land g > 0 \tag{6}$$

This formula was proved to be an invariant of the bouncing ball, which means it holds true always while the bouncing ball is bouncing around. Invariants are the most crucial information about the behavior of a system that we can rely on all the time. Since (6) is only an invariant of the bouncing dynamics not the ping pong ball, it, of course, only holds until the ping pong paddle hits, which changes the control. But until the ping pong paddle is used, (6) summarizes concisely what we know about the state of the bouncing ball at all times. Of course, (6) is an invariant of the bouncing ball, but it still needs to be true initially. The easiest way to make that happen is to assume (6) in the beginning of the ping pong ball's life.[1] Because Lecture 7 only conducted the proof of the bouncing ball invariant (6) for the case $c = 1$ to simplify the arithmetic, the ping pong ball now adopts this assumption as well. To simplify the arithmetic and arguments, also adopt the assumption $f = 1$ in addition to $c = 1 \land g = 1$ for the proofs.

[1]Note that H is a variable that does not need to coincide with the upper height limit 5 like it did in the case of the bouncing ball, because the ping pong ball has more control at its fingertips. In fact, the most interesting case is if $H > 5$ in which case the ping pong ball will only stay safe because of its control. One way to think of H is as an indicator for the energy of the ball showing how high it might jump up if it would not be for all its interaction with the ground and the ping pong paddle.

Substituting the safety-critical height 5 for H in the invariant (6) for this instance of parameter choices leads to the following condition

$$2x > 2 \cdot 5 - v^2 \tag{7}$$

as an indicator for the fact that the ball might end up climbing too high, because its energy would allow it to. Adding this condition (7) to the controller (5) leads to:

$$\begin{aligned}&\boldsymbol{2x = 2H - v^2} \land 0 \leq x \land x \leq 5 \land v \leq 0 \land g = 1 > 0 \land \mathbf{1} = \boldsymbol{c} \geq 0 \land \mathbf{1} = \boldsymbol{f} \geq 0 \rightarrow \\ &\quad \big[\big(\texttt{if}(x = 0)\, v := -cv \,\texttt{else}\, \texttt{if}((x > 5\frac{1}{2} - v \lor \boldsymbol{2x > 2 \cdot 5 - v^2}) \land v \geq 0)\, v := -fv; \\ &\quad\; t := 0; (x' = v, v' = -g, t' = 1 \,\&\, x \geq 0 \land t \leq 1)\big)^*\big](0 \leq x \leq 5)\end{aligned} \tag{8}$$

Recall that the bouncing ball invariant (6) is now assumed to hold in the initial state.

Is $\mathsf{d}\mathcal{L}$ formula (8) about its time-triggered controller valid? As usual, use an invariant or a counterexample for justification.

Before you read on, see if you can find the answer for yourself.

Formula (8) is "almost valid". But it is still not valid for a very subtle reason. It is great to have proof to catch those subtle issues. The controller in (8) takes action for two different conditions on the height x. However, the ping pong paddle controller actually only runs in (8) if the ball is not at height $x = 0$, for otherwise ground control takes action of reversing the direction of the ball. Now, if the ball is flat on the floor already ($x = 0$) yet its velocity so incredibly high that it will rush past height 5 in less than 1 time unit, then the ping pong paddle controller will not even have had a chance to react before it is too late, because it does not execute on the ground according to (8); see Fig. 4.

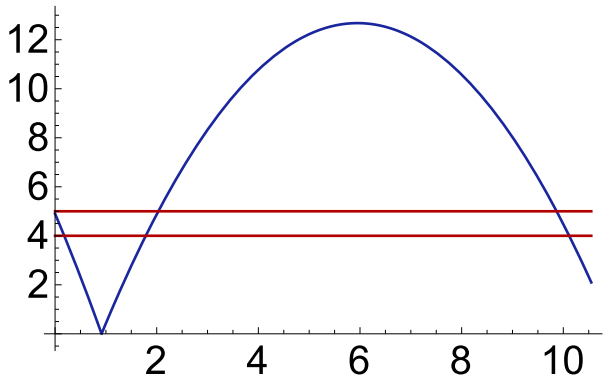


Figure 4: Sample trajectory of a time-triggered ping pong ball (as position over time), failing to control on the ground

Fortunately, these thoughts already indicate how that problem can be fixed. By turning the nested `if-then-else` cascade into a sequential composition of two separate `if-then` that will ensure the ping pong paddle controller to run for sure even if the bouncing ball is still on the ground (Exercise 3).

$$\begin{aligned} &2x = 2H - v^2 \land 0 \leq x \land x \leq 5 \land v \leq 0 \land g = 1 > 0 \land 1 = c \geq 0 \land 1 = f \geq 0 \rightarrow \\ &\quad \big[\big(\texttt{if}(x = 0)\, v := -cv \texttt{ ; } \texttt{if}((x > 5\frac{1}{2} - v \lor 2x > 2 \cdot 5 - v^2) \land v \geq 0)\, v := -fv; \\ &\quad\quad t := 0; (x' = v, v' = -g, t' = 1 \,\&\, x \geq 0 \land t \leq 1)\big)^*\big](0 \leq x \leq 5) \end{aligned} \tag{9}$$

Now, is formula (9) finally valid, please? If so, using which invariant? Otherwise, show a counterexample.

Before you read on, see if you can find the answer for yourself.

Yes, formula (9) is valid. What invariant can be used to prove formula (9)?

Formula (9) is valid, which, for the case $g = c = f = 1$, can be proved with the following invariant:

$$2x = 2H - v^2 \land x \geq 0 \land x \leq 5 \tag{10}$$

This invariant instantiates (6) for the present case of parameter choices and augments it with the desired safety constraint $x \leq 5$.

Yet, is the controller in (9) useful? That is where the problem lies now. The condition (7) that is the second disjunct in the controller of (9) checks whether the ping pong ball could possibly ever fly up to height 5. If this is ever true, it might very well be true long before the bouncing ball even approaches the critical control cycle where a ping pong paddle action needs to be taken. In fact, if (7) is ever true, it will also be true in the very beginning. After all, the formula (6), from which condition (7) derived, is an invariant, so always true for the bouncing ball. What would that mean?

That would cause the controller in (9) to take action right away at the mere prospects of the ball ever being able to climb way up high, even if the ping pong ball is still close to the ground and pretty far away from the last triggering height 5. That would make the ping pong ball safe, after all (9) is a valid formula. But it would also make it rather conservative and would not allow the ping pong ball to bounce around nearly as much as it would have loved to. It would basically make the bouncing ball lie flat on the ground, because of an overly anxious ping pong paddle. That would be a horrendously acrophobic bouncing ball if it never even started bouncing around in the first place. And the model would even require the (model) world to end, because there can be no progress beyond the point in time where the ball gets stuck on the ground. How can the controller in (9) be modified to resolve this problem?

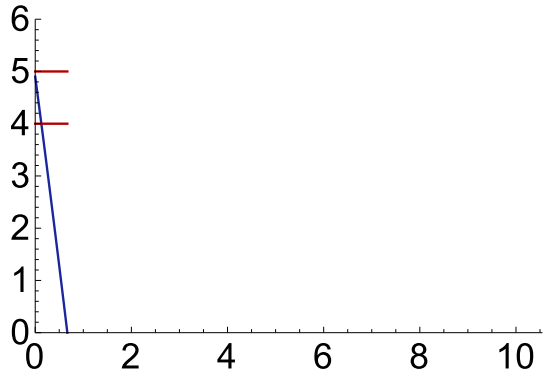
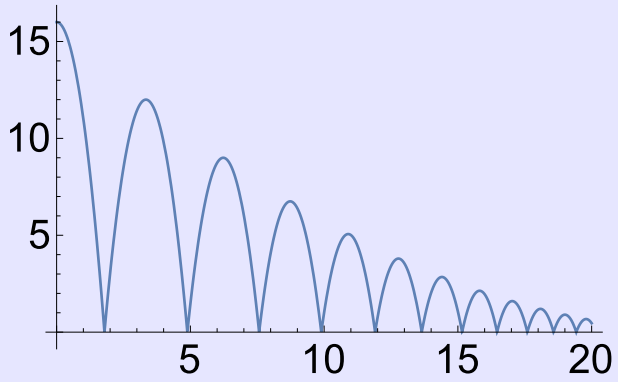


Figure 5: Sample trajectory of a time-triggered ping pong ball (as position over time), stuck on the ground

Before you read on, see if you can find the answer for yourself.

Note 3 (Zeno paradox). *There is something quite surprising about how (9) may cause the time to freeze. But, come to think of it, time did already freeze in mere bouncing balls.*



The duration between two hops on the ground in a bouncing ball keeps on decreasing rapidly. If, for simplicity, the respective durations are $1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$*, then these durations sum to*

$$\sum_{i=0}^{\infty} \frac{1}{2^i} = \frac{1}{1-\frac{1}{2}} = 2$$

which shows that the bouncing ball model will make the (model) world stop to never reach time 2 nor any time after. Hence, the bouncing ball model disobeys what is called divergence of time, i.e. that the real time keeps diverges to ∞*. The reason this happens is that the bouncing ball keeps on switching directions on the ground more and more frequently. This is very natural for bouncing balls, but can cause subtleties and issues in other systems if they switch infinitely often in finite time.*

The name Zeno paradox comes from the Greek philosopher Zeno (ca. 490–430 BC) who found a paradox when fast runner Achilles gives the slow Tortoise a head start of 100 meters in a race: In a race, the quickest runner can never overtake the slowest, since the pursuer must first reach the point whence the pursued started, so that the slower must always hold a lead. – recounted by Aristotle, Physics VI:9, 239b15

Pragmatic solutions for the Zeno paradox in bouncing balls add a statement that make the ball stop when the remaining velocity on the ground is too small. For example:

$$\texttt{if}(x = 0 \land -0.1 < v < -0.1)\, v := 0; x' = 0$$

The idea is to restrict the use of the second `if-then` disjunct (7) in (9) to slow velocities in order to make sure it only applies to the occasions that the first controller disjunct $x > 5\frac{1}{2} - v$ misses, because the ball will have been above height 5 in between. Only with slow velocities will the ball ever move so slowly that it is near its turning point to begin its descent and start falling down again before 1 time unit. And only then could the first condition miss out on the ball being able to evolve above 5 before 1 time unit. When is a velocity slow in this respect?

For the ball to turn around and descend, it first needs to reach velocity $v=0$ by continuity (during the flying phase) on account of the mean-value theorem. In gravity $g=1$ the ball can reach velocity 0 within 1 time unit exactly when its velocity was $v<1$ before the differential equation, because the velocity changes according to $v(t)=v-gt$. Consequently, adding a conjunct $v<1$ to the second disjunct in the controller makes sure that the controller only checks for turnaround when it might actually happen during the next control cycle.

$$\begin{aligned}&2x=2H-v^2 \land 0\leq x\land x\leq 5\land v\leq 0\land g=1>0\land 1=c\geq 0\land 1=f\geq 0\rightarrow\\ &\quad\big[\big(\texttt{if}(x=0)\,v:=-cv;\texttt{if}((x>5\frac{1}{2}-v\lor 2x>2\cdot 5-v^2\land \boldsymbol{v<1})\land v\geq 0)\,v:=-fv;\\ &\qquad t:=0;(x'=v,v'=-g,t'=1\,\&\,x\geq 0\land t\leq 1)\big)^*\big](0\leq x\leq 5)\end{aligned} \tag{11}$$

This dℒ formula is valid and provable with the same invariant (10) that was already used to prove (9). It has a much more aggressive controller than (9), though, so it is more fun for the ping pong ball to bounce around with it.

The easiest way of proving that dℒ formula (11) is valid using invariant (10) is to show that the invariant (10) holds after every line of code. Formally, this reasoning by lines corresponds to a number of uses of the generalization proof rule MR to show that the invariant (11) remains true after each line if it was true before. The first statement $\texttt{if}(x=0)\,v:=-cv$ does not change the truth-value of (10), i.e.

$$2x=2H-v^2\land x\geq 0\land x\leq 5\rightarrow[\texttt{if}(x=0)\,v:=-cv](2x=2H-v^2\land x\geq 0\land x\leq 5)$$

is valid, because, when $c=1$, the statement can only change the sign of v and (10) is independent of signs, because the only occurrence of v satisfies $(-v)^2=v^2$. Likewise, the second statement $\texttt{if}((x>5\frac{1}{2}-v\lor 2x>2\cdot 5-v^2\land \boldsymbol{v<1})\land v\geq 0)\,v:=-fv$ does not change the truth-value of (10), i.e.

$$\begin{aligned}&2x=2H-v^2\land x\geq 0\land x\leq 5\rightarrow\\ &[\texttt{if}((x>5\frac{1}{2}-v\lor 2x>2\cdot 5-v^2\land \boldsymbol{v<1})\land v\geq 0)\,v:=-fv](2x=2H-v^2\land x\geq 0\land x\leq 5)\end{aligned}$$

is valid, because, for $f=1$, the second statement can also only change the sign of v, which is irrelevant for the truth-value of (10). Finally, the relevant parts of (10) are a special case of (6), which has already been shown to be an invariant for the bouncing ball differential equation in Lecture 7 on Loops & Invariants and, thus, continues to be an invariant when adding a clock $t'=1\,\&\,t\leq 1$, which does not occur in (10). The additional invariant $x\leq 5$ that (10) has compared to (6) is easily taken care off using the corresponding knowledge about H.

Recall that (global) invariants need to be augmented with the usual trivial assumptions about the unchanged variables: $g=1\land 1=c\land 1=f$.

See ≪Time-triggered ping pong ball KeYmaera model≫

Note 4 (Time-triggered control). *One common paradigm for designing controllers is time-triggered control, in which controllers run periodically or pseudo-periodically with certain frequencies to inspect the state of the system. Time-triggered systems are closer to implementation than event-driven control. They can be harder to build, however, because they invariably require the designer to understand the impact of delay on control decisions. That impact is important in reality, however, and, thus, effort invested in understanding the impact of time delays usually pays off in designing a safer system that is robust to bounded time delays.*

Partitioning the hybrid program in (11) into the parts that come from physics (typographically marked like physics) and the parts that come from control (typographically marked like control) leads to:

$$2x = 2H - v^2 \land 0 \leq x \land x \leq 5 \land v \leq 0 \land g = 1 > 0 \land 1 = c \geq 0 \land 1 = f \geq 0 \rightarrow$$
$$\big[\big(\texttt{if}(x = 0)\, v := -cv; \texttt{if}((x > 5\frac{1}{2} - v \lor 2x > 2 \cdot 5 - v^2 \land \boldsymbol{v < 1}) \land v \geq 0)\, v := -fv;$$
$$t := 0; (x' = v, v' = -g, t' = 1 \,\&\, x \geq 0 \land t \leq 1)\big)^*\big](0 \leq x \leq 5) \tag{11}$$

Note how part of the differential equation, namely $t' = 1$, comes from the controller, because it corresponds to putting a clock or on the controller and running it with at least the sampling frequency 1 (coming from the evolution domain constraint $t \leq 1$).

3 Summary

This lecture studied time-triggered control, which, together with event-driven control from Lecture 8 on Events & Responses, is an important principle for designing feedback mechanisms in CPS and embedded systems. The lecture illustrated the most important aspects for a running example of a ping pong ball. Despite or maybe even because of its simplicity, the ping pong ball was an instructive source for the most important subtleties involved with time-triggered control decisions. Getting time-triggered controllers correct requires predictions about how the system state might evolve over short periods of time (one control cycle). The effects and subtleties of time-triggered actions in control were sufficiently subtle to merit focusing on a simple intuitive case.

Unlike event-drive control, which assumes continuous sensing, time-triggered control is more realistic by only assuming the availability and processing of sensor data at discrete instants of time (discrete sensing). Time-triggered system models avoid the modeling subtleties that events tend to cause for the detection of events. It is, thus, often much easier to get the models right for time-triggered systems than it is for event-driven control. The price is that the burden of event-detection is then brought to the attention of the CPS programmer, whose time-triggered controller will now have to ensure it predicts and detects events early enough before it is too late to react to them. That is what makes time-triggered controllers more difficult to get correct, but is also

crucial because important aspects of reliable event detection may otherwise be brushed under the rug, which does not exactly help the final CPS become any safer either.

CPS design often begin by pretending the idealized world of event-driven control (if the controller is not even safe when events are checked and responded to continuously, it is broken already) and then subsequently morphing the event-driven controller into a time-triggered controller. This second step then often indicates additional subtleties that were missed in the event-driven designs. The additional insights gained in time-triggered controllers are crucial whenever the system reacts slowly or whenever it reacts fast but needs a high precision to remain safe. For example, the reaction time for ground control decisions to reach a rover on Mars are so prohibitively large that they could hardly be ignored. Reaction times in a surgical robotics system that is running at, say, 55Hz, are still crucial even if the system is moving slow and reacting fast, because the required precision of the system is in the sub-millimeter range [KRPK13].

Overall, the biggest issues with event-driven control, besides sometimes being hard to implement, is the subtleties involved in properly modeling event detection without accidentally defying the laws of physics in pursuit of an event. But controlling event-driven systems is reasonably straight-forward as long as the events are chosen well. Finding a model is comparably canonical in time-triggered control, but identifying the controller constraints takes a lot more thought, leading, however, to important insights about the system at hand.

Exercises

Exercise 1. The HP in (4) imposes an upper bound on the duration of a continuous evolution. How can you impose an upper bound 1 and a lower bound 0.5?

Exercise 2. Give an initial state for which the controller in (4) would skip over the event without noticing it.

Exercise 3. What would happen if the controller in (9) uses the ping pong paddle while the ball is still on the ground? To what physical phenomenon does that correspond?

Exercise 4. The formula (11) with the time-triggered controller of reaction time at most 1 time unit is valid. Yet, if a ball is let loose a wee bit above ground with a very fast negative velocity, couldn't it possibly bounce back and exceed the safe height 5 faster than the reaction time of 1 time unit? Does that mean the formula ought to have been falsifiable? No! Identify why and give a physical interpretation.

Exercise 5. The controller in (11) ran at least once a second. How can you change the model and controller so that it runs at least twice a second? What changes can you do in the controller to reflect that increased frequency? How do you need to change (11) if the controller only runs at least once every two seconds?

Exercise 6. Conduct a sequent proof proving the validity of $\mathsf{d}\mathcal{L}$ formula (10). Is it easier to follow a direct proof or is it easier to use the generalization rule **??** for the proof?

Exercise 7. The event-driven controller we designed in Lecture 8 on Events & Responses monitored the event $4 \leq x \leq 5$. The time-triggered controller in Sect. 2, however, ultimately only took the upper bound 5 into account. How and under which circumstances can you modify the controller so that it really only reacts for the event $4 \leq x \leq 5$ rather than under all circumstances where the ball is in danger of exceeding 5?

Exercise 8. Devise a controller that reacts if the height changes by 1 when comparing the height before the continuous evolution to the height after. Can you make it safe? Can you implement it? Is it an event-driven or a time-triggered controller? How does it compare to the controllers developed in this lecture?

Exercise 9. The ping pong ball proof relied on the parameter assumptions $g = c = f = 1$ for mere convenience of the resulting arithmetic. Develop a time-triggered model, controller, and proof for the general ping pong ball.

Exercise 10. Show that the ping pong ball (11) can also be proved safe using just the invariant $0 \leq x \leq 5$ (possibly including assumptions on constants such as $g > 0$). Which assumptions on the initial state does this proof crucially depend on?

Exercise 11 (*). Design a variation of the time-triggered controller for the ping pong ball that is allowed to use the ping pong paddle within height $4 \leq x \leq 5$ but has a relaxed safety condition that accepts $0 \leq x \leq 2 \cdot 5$. Make sure to only force the use of the ping pong paddle when necessary. Find an invariant and conduct a proof.

References

[KRPK13] Yanni Kouskoulas, David W. Renshaw, André Platzer, and Peter Kazanzides. Certifying the safe design of a virtual fixture control algorithm for a surgical robot. In Calin Belta and Franjo Ivancic, editors, *HSCC*, pages 263–272. ACM, 2013. doi:10.1145/2461328.2461369.

[Pla10] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. doi:10.1007/978-3-642-14509-4.

[Pla12] André Platzer. Dynamic logics of dynamical systems. *CoRR*, abs/1205.4788, 2012. arXiv:1205.4788.