Christoph Lüth

Entwicklung von Systemen hoher Sicherheit und Qualität

Wintersemester 2025/26

Lecture Notes

Last revision: December 3, 2025.

1 Lecture 5: Hybrid Systems

1.1 Introducing Our Running Example

Our running will be an (autonomous) car. The most basic safety requirement for a car, autonomous or not, is:

- It should not run into obstacles.

There is also a liveness requirement, because a car which is safe in that sense is one which never moves:

- It should run.

That may mean, it should run wherever the driver wants it to go (unless there is an obstacle in the way), or it should eventually travel to its destination unless there is no path leading there. Conceptually, it makes sense to consider the upper layers (navigation, route finding, *etc.*) of an autonomous car, and the driver in a non-autonomous car, to be the same — and we will model it in that sense below.

A car moves about in a physical environment. This environment is *continuous*, as opposed to the discrete nature of digital systems. The question is, how do we model a digital system with a discrete time rate in a fundamentally continuous environment? Let's look at how to model continuous systems first.

A very simple car is one which can just go forwards and backwards. It has a position, $s \in \mathbb{R}$, given by its distance to some origin, and moves about with a velocity $v \in \mathbb{R}$ and an acceleration $a \in \mathbb{R}$. All of these depend on each other; by setting one, we determine the other. In the simple car, if we set s that corresponds physical to just put the car somewhere. Then $v = \dot{s}$ and $a = \dot{v}$ give us the velocity and acceleration of this action. However, this is not realistic; in practice (depending a bit on the drive technology), we set the acceleration a and get s and v:

$$s = \int v\,dt \quad v = \int a\,dt$$

If a is constant, we get

$$v = at + v_0 \tag{1}$$
$$s = \frac{a}{2}t^2 + v_0 t + s_0 \tag{2}$$
$$\tag{3}$$

(where $v_0 = v(0)$ and $s_0 = s(0)$, the initial values for s and v). In summary, our system has a *state* given by real-valued variables s, v, a depending on the time. Some of these are *control variables* which mean they determine the behaviour of the system (and are thus indeterminate), others (such as v and s) are derived from these.

1.2 Hybrid Systems

If a computer is controlling the car, it will run in discrete time "ticks", where every some Milliseconds inputs are read, outputs are computed and the control variables are set. At this moment the behaviour of the system is not continuous (e.g. the acceleration is set arbitrarily). We have already seen how finite state machines are the most basic model of a computer; the concept of a *hybrid system* models the combination of finite discrete-state machines with a continous environment. (This definition is from [1], which also contains the definitions elided here.)

Definition 1 (Hybrid System) *A* hybrid system *is given by*

$$\mathscr{H} = (V_D, Q, E, \mu_1, \mu_2, \mu_3)$$

where

- V_D *is a set of n real-valued* data variables *valued in* $\Sigma_D = \mathbb{R}^n$*;*
- Q *is a set of* locations *(discrete states);*
- E *is a* transition relation $E \subseteq Q \times Q$*;*
- *For each* $q \in Q$*,* $\mu_1(q)$ *is a set of* activities *(for each data variable) specified by* differential equations *and* $\mu_2(q) : \Sigma \to \mathbb{B}$ *is an* invariant *predicate for each data variable*
- *For each* $(p,q) \in E$*,* $\mu_3(p,q) \subseteq \Sigma \times \Sigma$ *is a transition relation between data variables.*

Intuitively, the system starts in a state $q \in Q$ with data variables set to $\sigma \in \Sigma_D$ such that $\mu_2(q,\sigma)$ is true. (Hybrid systems can be made into hybrid automata by adding an initial and acceptance condition, see [1].) The continuous behaviour of the data variables is specified by $\mu_1(q)$. As soon as the predicate $\mu_2(q,\sigma)$ becomes false, the automaton transitions into a state $p \in Q$ where $(q,p) \in E$; the values of the data variables may then change into $\tau \in \Sigma_D$ such that $\mu_3(p,q,\sigma,\tau)$ is true. Thus, the state can change in two ways: by an *instantaneous* transition that changes the entire state according to the successor relation, or by elapse of time that changes only the values of data variables in a *continuous manner* according to the activities of the current location.

This is made precise by the concept of a *run* (the analog of traces). Formally, given a hybrid system $\mathscr{H}$ as above, a run of $\mathscr{H}$ is given by (finite or infinite) sequence

$$\longrightarrow_{\sigma_0} (l_0, I_0, f_0)_{\sigma'_0} \longrightarrow_{\sigma_1} (l_1, I_1, f_1)_{\sigma'_1} \longrightarrow_{\sigma_1} (l_2, I_2, f_2)_{\sigma'_2} \longrightarrow \ldots$$

with $\sigma_i, sigma'_i \in \Sigma_D, l_i \in Q$, intervals I_i and activities f_i such that

(i) for $i \geq 0$, $(l_{i+1}, sigma_{i+1})$ is a successor of (l_i, σ'_i) (i.e. $(sigma'_i, \sigma_{i+1}) \in \mu_3(l_i, l_{i+1})$);

(ii) $I_0, I_1, I_2, \ldots$ is an interval sequence (i.e. a sequence of non-overlapping intervals the union of which is equal to $\mathbb{R}^+$);

(iii) for $i \geq 0$, f_i is in $\mu_1(l_i)$ and $f_i(0) = \sigma_i$, $f_i(r_{I_i}) = \sigma'_i$, and for all $t \in I_i, f_i(t - l_{I_i}) \notin \mu_2(l_i)$.

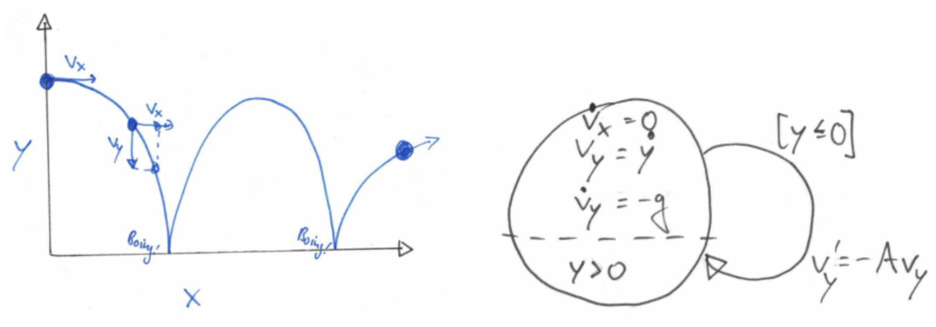

Figure 1: A bouncing ball (left), modelled as a hybrid system (right).

1.3 Examples

1.3.1 The Bouncing Ball

When we write examples, we use the following graphical notation, where the states are nodes containing the activities' equations and the invariant, and the transition relations are written as arrows. Here is an example: A bouncing ball starts at height y with a horizontal velocity v_x and a vertical velocity $v_y = 0$. The ball falls down towards the floor (Figure 1, left). When it reaches the floor (height $y = 0$), the ball bounces; the velocity is inverted (and reduced by a factor $0 < A \leq 1$). This is a non-continuous state change, but it returns to the same state. Thus, the state space is $\Sigma = \langle x, y, v_x, v_y \rangle$, and the hybrid system is given as in Figure 1 right.

1.3.2 A Very Simple Car

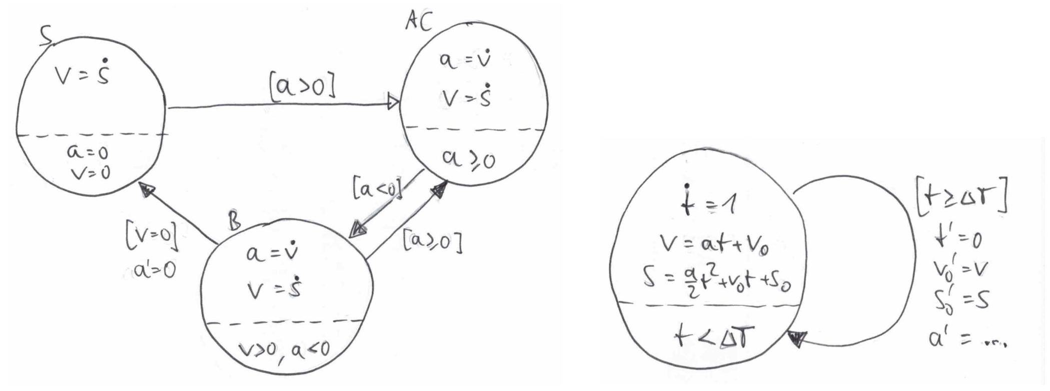

Figure 2: Hybrid systems modelling the very simple car. Uncontrolled acceleration (left) vs. controlled acceleration (right).

Let us now reconsider the very simple car from before. Here, the state space is given as $\Sigma = \langle s, v, a, s_0, v_0 \rangle$. Let us say we have three states: standing still (S), accelerating or cruising (AC), and braking B; see Figure 2, left. But here it is unclear where the acceleration a comes from.In hybrid systems, data variables

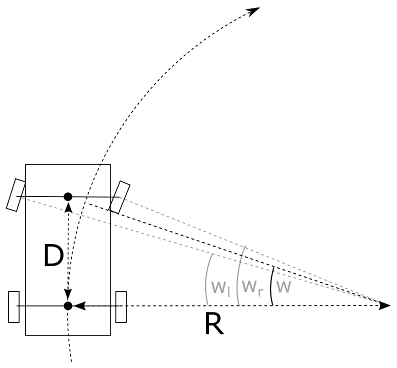

Figure 3: Ackermann steering geometry

are not control variables which can be set; the car could be driven by a person[1]or it could be controlled by a computer. In the second case, we can given an alternative model which makes the control structure more precise: every ΔT seconds, the system reads the input and calculates a response (in the form of an acceleration a). (We assume it does so in zero time; we can model the response time of the controller as well, but let us be simple for the time being.)

The model is given in Figure 2 on the right. It uses a system state $\Sigma = \langle s, v, a, s_0, v_0, t \rangle$, and the trick to make the time explicit, i.e. part of the system state Σ. We require t to be a data variable with $\dot{t} = 1$ (because the derivative of the identity function is 1). The system just has one location, and a non-discrete state change every ΔT seconds; this state change can set the acceleration a.

The model is very rudimentary. For example, it does not brake to a standstill, it will just start to move backwards unless one very accurately calculates the acceleration needed to come to a complete standstill.

2 Lecture 6: Foundations

2.1 A Real Car

Most cars nowadays have two steerable front wheels, and a fixed back axle; when keeping the steering angles fixed, the car travels in a circle. This is called the Ackermann geometry[2], and as everybody trying to reverse park into a tight spot can attest, it can be tricky to manoeuvre.

Our car has a position $P = \langle x, y \rangle$, a velocity $v \in \mathbb{R}$, an orientation $\phi \in \mathbb{R}$, a steering angle $\omega \in \mathbb{R}$, and acceleration $a \in \mathbb{R}$. This gives a system state $\mathscr{X} = \langle x, y, v, \phi, \omega, a \rangle$. We assume a and ω are control variables, so they can be set by the controlling algorithm. (The steering angle ω is actually two steering angles ω_l and ω_r for the left and right front leg, but we do not need that here.) The car also has parameters: it is a rectangle of width W and length L, with the distance between the axles D. (We will use capital letters for all values which are not time-dependent.) In this state, the car travels along a circle $R = \frac{D}{\tan(\omega)}$, with R going to infinity if $\omega = 0$ (going straight ahead). While doing so, it changes orientation by (approximately) $v \cdot \frac{\omega}{D}$.

With the acceleration, we get the scalar velocity $v = \int a\, dt$. Together with the orientation ϕ, the velocity vector is $\vec{v} = (\phi, v)$ given in polar coordinates. The position then is the vector integral $s = \int \vec{v}\, dt$.

[1]Who would lead a rather boring life just going back and forth.
[2]See `https://en.wikipedia.org/wiki/Ackermann_steering_geometry`

2.2 Safety Requirements Revisited

We now want to formalize the safety requirement:

- The car should not run into obstacles.

Firstly, the car in state $\mathscr{X}$ always covers an area $A(\mathscr{X})$ which is a set of points (in the plane):

$$A(\langle x,y,v,\phi,\omega,a\rangle) = \{(p,q) \mid |p-x| \leq \frac{L}{2}, |q-y| \leq \frac{W}{2}\}$$

Now, the obstacles are just arbitrary point clouds $Obs \in \mathscr{P}(()\mathbb{R})$. Note that they do not depend on time, so they are static. The requirement above formalizes as

$$A(\mathscr{X}) \cap Obs = \emptyset \tag{4}$$

To show that this always holds (at all points in time) we need to show that all points we are able to brake to standstill without running into obstacles. To formulate this, we first need the *time* for braking to a standstill: if we travel with speed v_0 and accelerate with $-a_{brk}$, then our speed at time t will always be

$$v(t) = v_0 - a_{brk} \cdot t \tag{5}$$

Setting $v(t) := 0$ and solving for t easily gives the breaking time

$$t_{brk} = \frac{v_0}{a_{brk}} \tag{6}$$

Now the area covered while breaking A_{brk} is given by the integral

$$A_{brk} = \int_0^{t_{brk}} A(\mathscr{X})\,dt = \cup_{t\in[0,t_{brk}]} A(\mathscr{X}) \tag{7}$$

Note $\mathscr{X}$ and all its components depend on t; we should make this explicit. In order to check whether this area is free of obstacles we need some way of *sensing* obstacle.[3]

2.3 Sensor Input

There are a lot sensors for mobile robots to detect obstacles. The most basic ones are bumpers, which when running into an obstacle trigger a physical contact and initiate braking. They are comparatively cheap, reliable, and utterly useless in this case (primary use cases are slow-moving industrial autonomous robots e.g. found in hospitals such as Klinikum Bremen-Mitte). Remote-sensing sensors include ultrasonic, radar and lidar sensors. These send out signals and measure the time a reflection needs to calculate the distance. Ultrasonics work with ultrasonic sound waves; they are cheap, but fairly unreliable and in particular not directed (they just give the distance, not the direction, of an obstacle; I do not know how bats can use them to find prey). Lidar and radar sensors work with electromagnetic waves; lidar in particular use laser beams and a rotating mirror to measure both distance and direction of possible obstacles. They are more expensive, but very reliable and precise. Abstractly, such a sensor returns a sequence of n distance measurements

$$L = \langle l_0, \ldots, l_n \rangle_{n\in\mathbb{N}} \tag{8}$$

[3]Looking them up on Google Maps is hardly safety-directed.

which cover an angle from 0 to N_0 (with $N_0 = \pi$ or even $N_0 = 2\pi$), and n in the range of a couple of hundred measurements.

So we now want to check whether the area given by equation (7) is free from obstacles as given by measurements as given in equation (8). Recall from above that abstractly we treated the obstacles as a point cloud $Obs \in \mathscr{P}(\mathbb{R}^2)$. We need to show (mathematically) that the measurements from (8) detect obstacles; however, on this level of abstraction this is impossible, as there may always be points between the measurements. (We could model obstacles as circles, rectangles, or polygons, but then we would also need to model the sensing more finely as a sequence of rays emanating from a central point, and intersecting with the obstacle.) But we also need to show that our measurements cover the whole of the area defined by 7; so not only need we reason about the presence of obstacles, but also their absence.

For lidar and radar scanners, this sort of argument is part of the certification process; it needs to be demonstrated that they (physically) detect obstacles which are present. (This is not a given; if the obstacles does not reflect much light, the sensor needs to work even with that. The relevant norm states that a lidar sensor must detect black velvet at the given distance, which is not trivial.) In any case, this is not really our concern here, it just demonstrates the width of arguments one needs for comprehensive safety in the real world.

2.4 Proving Safety

In order to keep our running example clear without being too simple, we assume that the measurements given in 8 really detect all relevant obstacles, and that we are safe if we can always brake within a *distance* s_{brk} which is less than the *smallest* distance measurement.

We measure s_{brk} and the distances in L from the origin of our car; S is the size (radius of the covering circle) of car; for the rectangular car of width W and length L, this is $S = sqr\frac{W^2+L^2}{2}$.[4]Then we need that the smallest distance measurement is always greater than the braking distance plus S:

$$S + s_{brk} < \min(L) \tag{9}$$

Note that we also abstracted over the steering geometry, overapproximating in the process: we assume that the car can possibly move in any direction at any time, though when breaking only with speed as given by (5). To calculate s_{brk}, we first need the time to brake. In time t we travel

$$s(t) = \frac{a}{2}t^2 + v_0 \cdot t$$

[4]S could also contain a safety margin if e.g. we always want to brake 10cm before an obstacle. This is useful if the distance measurements are from a lidar sensor mounted at ankle height, and the obstacle is leg, then we do not want to run over the foot belonging to the leg while braking.

so with equation (6) and $v_0 := v(t)$ we get the breaking distance s_{brk} at time t as $s(t_{brk})$:

$$\begin{aligned}
s_{brk}(t) &= \frac{-a_{brk}}{2}\left(\frac{v(t)}{a_{brk}}\right)^2 + v(t)\cdot\frac{v(t)}{a_{brk}} \\
&= \frac{-v(t)^2}{2\cdot a_{brk}} + \frac{v(t)^2}{a_{brk}} \\
&= \frac{-v(t)^2 + 2\cdot v(t)^2}{2\cdot a_{brk}} \\
&= \frac{v(t)^2}{2\cdot a_{brk}}
\end{aligned}$$

With this, (9) becomes

$$S + \frac{v^2(t)}{2\cdot a_{brk}} < \min(L) \tag{10}$$

We now need to show that this equation holds at all times. This means that for the discrete system we have to that

(i) it holds in all initial states, and

(ii) if it holds at time t, and the acceleration is set as given by the control program, it will hold at time $t+\Delta T$ as well.

2.5 The Control Program

We first specify the control program in an abstract setting, before coding it as a C program. The control program could be modelled a hybrid automaton with two states: one where the car is crusing (driving regularly), and one where it is in emergency braking mode. It starts off cruising, and enters emergency braking mode if at time $t+\Delta T$ the safety requirement would not hold anymore. But this is not satisfactorily: the detection of the invariant violation of the hybrid automaton (i.e. that (10) does not hold) should be anticipated by the control program; in the continuous time evolution part of the hybrid automaton, we cannot change the acceleration any more.

So we model the control program by a discrete FSM with the two state cruising (C) and emergency braking (E) as given. The state of the program is given by

$$\mathcal{X} = \langle x, y, v, \phi, \omega, a, L\rangle$$

All of these depend on the time in the sense that they change over time; however, for the control program, time is implicit. In other words, these are data that are kept in the memory of the controller (or read from sensors), and they may change from time $t_1, t_2, t_3, \ldots$, but at any given point t_i in time when the control program is run the values are fixed.

References

[1] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Robert L. Grossman,

Anil Nerode, Anders P. Ravn, and Hans Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science, pages 209–229, Berlin, Heidelberg, 1993. Springer.