INFO-F410:
Electric-Bike Project

April 2022

Abstract

This note describes the project for INFO-F410. The project is to be made on a strict individual basis.[1]

Report. The structure of the report folder is the following:

```
your_name/
  lustre/
    lustre_report.pdf
    main.lus
  strix/
    strix_report.pdf
    <strix_project_files>
  uppaal/
    uppaal_report.pdf
    uppaal_project.xml
    uppaal_project.q
  xcos/
    xcos_report.pdf
    <xcos_project_files>
```

1 Introduction

You are the chief safety engineer at SuperVolt Corp., and lead a project on developing the world-safest electric bike. Only you can stop the company from releasing the bike with critical safety issues, saving it from billion-euro fines, and protect people lives. Your aids are: formal verification, synthesis, testing, and simulation. ;-)

The main goal of this project is to allow you to experience the use of formal models supported by verification and synthesis and simulation techniques as aid to the design of a piece of reactive software. This reactive software is part of a representative embedded system. We ask you to design and verify a part of a motor controller of an electric bike,

[1]As a consequence any breach of this rule will trigger the application of the general rules of the university to combat plagiarism, which a.o. include exclusion. Those rules will apply both for the copying side as well as the source side.

on different abstraction levels. You will synthesise automatically and design manually models, and write logical specifications, from the informal descriptions given in this note.

The project consists of four parts. In the part on synthesis, you will automatically synthesise a part of the motor controller using Strix [4]. In the part on verification of timed automata, you will design and verify a timed model using UppAal [1]. In the part on reactive programming, you will implement the controller manually in the synchronous data-flow language Lustre [2]. Finally, the part on simulation requires designing a physical dynamics model of a bike and then simulating it in Xcos [3].

2 High-level description of the bike components

The electric bike is schematically depicted in Fig. 2.

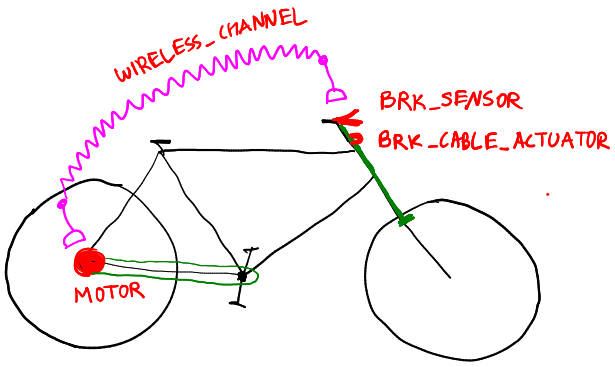


Figure 1: Components of the bike: motor controller, brake sensor, brake-cable actuator, wireless channel.

The bike comes equipped with an electric motor and a braking system. The braking system consists of a brake sensor and actuator. The brake sensor `BRK_SENSOR` outputs a signal whenever the rider presses on the brake lever. There are two press levels of the braking: soft braking and hard braking (and, of course, no braking at all). This signal is read by the brake actuator `BRK_ACT`. The braking level is also sent wirelessly to the motor. The wireless communication is not reliable. When the connection is active, on receiving the soft or hard braking command, the motor controller should engage the regenerative braking using the motor. Such a regenerative braking charges the battery. On the other hand, when the communication between the braking sensor and motor controller is lost, the regenerative braking should not be engaged. In this case, the braking actuator should clinch the brake cable leading to the rim brake on the front wheel, engaging the manual brake. The actuator should also clinch the cable and engage the manual braking when the rider presses hard on brakes.

The motor is also used to assist the driver. To this end, the motor controller reads the signal from the power sensor measuring the current effort by the rider, the current speed, and the battery level. Finally, note that the motor should not assist the driver on speeds above 25 km/h nor when the braking is active nor when the battery level is 0. The motor should not engage in assisting nor braking on losing communication with the braking sensor.

These are a subset of operational rules of the braking system and the motor controller. We now look at them in more detail.

Model decomposition

Figure 2 depicts the components that compose the system and their interaction. Those pieces of equipment communicate with each other by emitting and reading events[2]. We adopt here the vocabulary of the synchronous approach to reactive systems as used in Lustre [2]. In the following description, we assume that the *execution* of the system can be represented as a time line of linearly ordered logical time points $t_0, t_1, \ldots, t_n, \ldots$. In each time point, each signal (Boolean or valued) can be present or absent. If a valued signal is present, then it carries a value. For example, if there are two signals: one Boolean $A : \mathbb{B}$, and one integer-valued $B : \mathbb{Z}$, then the following sequence:

$$(t_0, \{A, B(1)\}), (t_1, \{B(2)\}), (t_2, \{A\}), \ldots$$

denotes a prefix of an infinite execution. In the initial logical time point, the Boolean signal A is *true*, the integer signal B carries the value 1. In the second logical time point, the Boolean signal A is *false* and the integer signal B carries the value 2. Finally, in the third logical time point, the Boolean signal A is *true* and the integer signal B is not active.

We now describe how the components interact in detail.

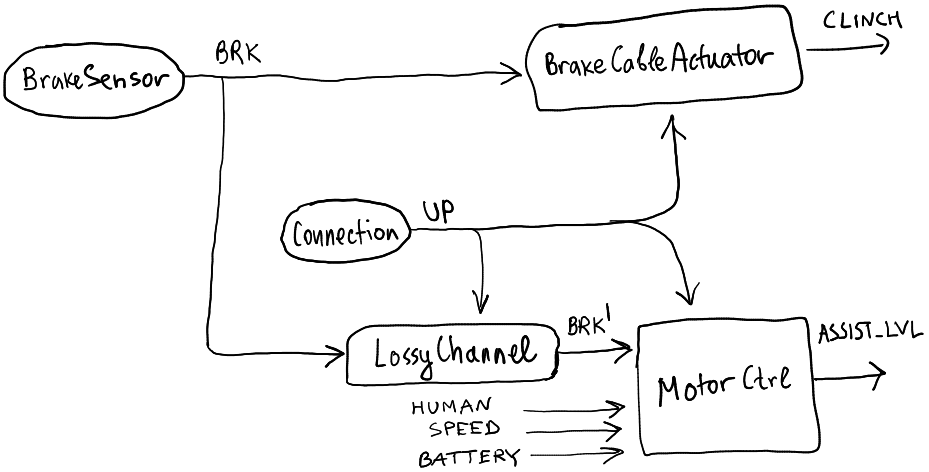


Figure 2: Interaction of the components.

Brake sensor. The brake sensor models human braking behaviour by outputting signal $\mathsf{BRK} : \mathbb{Q}_{\geq 0}$. We distinguish three cases: when $\mathsf{BRK} = 0$ no braking happens, when $\mathsf{BRK} \in (0, BRK\_SOFT]$ the soft braking mode is active, and when $\mathsf{BRK} > BRK\_SOFT$ the hard braking mode is active, where BRK_SOFT is a certain constant. As this component models the environment (human) behaviour, it is nondeterministic.

[2]Throughout this document, we use both the terms *event* and *signal* interchangeably. The term *event* is an intuitive notion that we have used in the formal verification course while *signal* is more used in the context of control or Lustre.

Lossy channel. This component models a wireless transmission of data. It reads the values from the braking sensor and the current status UP of the channel. When UP is true, the component truthfully relays the value on its input to the output. When UP is false, the output may be anything. This component is nondeterministic.

Brake cable actuator. The behaviour of the actuator depends on BRK and on status UP of the connection with the motor. First, consider the case when the connection is up.

- When the human brakes softly, the brake cable actuator does not engage (thus CLINCH is false). At the same time, the motor should activate its regenerative braking. In the regenerative braking mode, the motor uses the kinetic energy of the bicycle to charge the battery.
- When the cyclist applies hard braking, the cable actuator engages the rim brake (CLINCH becomes true). Note that the motor should still apply the regenerative braking.

Now consider the case when the connection is down (UP is false). As there is no reliable connection with the motor, the brake cable actuator should activate the rim brakes to ensure manual braking. This should happen in both cases of soft and hard braking.

Motor controller. The motor controller is responsible for regenerative braking and for assisting the cyclist. The component reads the braking signal through the wireless channel, the amount of force the human applies, the current speed, the battery level, and outputs the amount of assistance $\mathsf{ASST\_LVL} : \mathbb{Q}$ that the motor should apply. When $\mathsf{ASST\_LVL} > 0$, we say that the motor should assist the driver, when $\mathsf{ASST\_LVL} < 0$, we say that the motor should apply the regenerative braking, and when $\mathsf{ASST\_LVL} = 0$, the motor should not be engaged. The motor controller behaves as follows. When the connection with the braking sensor is down, the motor should not be engaged. Consider the case when the connection is up. If the braking occurs, the motor controller activates the regenerative braking whose level is proportionate to the amount of braking BRK'. Otherwise, if the speed is below 25 km/h, the motor controller should provide the assistance that equals the current human force. When the speed is above 25 km/h, the motor should not provide assistance. On top of that, the motor should not provide assistance if the battery level is 0 (but the regenerative braking can still occur).

In the next sections, we design and verify (some of) these components on different levels of abstraction.

3 LUSTRE: design and verification of discrete-time systems

In this part we focus on the discrete-time abstraction of the system. Your task is to implement a few components in language LUSTRE and verify them using model checker KIND2. This model checker is capable of verifying safety properties of systems working

with integer and real values, and although such a model checking problem is undecidable in general, the tool is capable of verifying many interesting cases.

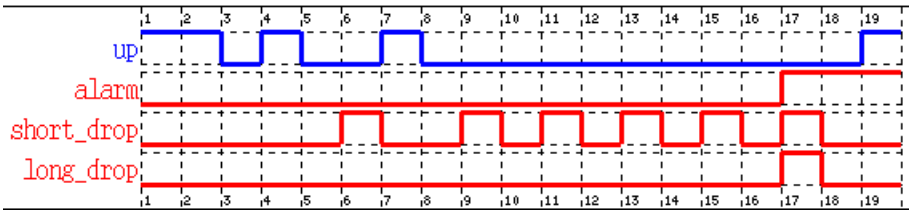
You need to implement the components `MotorController`, `BrakeCableActuator`, `LossyChannel`, and `ConnectionAlarm`.

Task L1: ConnectionAlarm

Note that the this module is *not* shown on the interaction scheme in Figure 2. This module reads the connection status UP, and raises the ALARM flag when the number of connection errors reaches a certain limit, signalling that the connection hardware is probably faulty and should be replaced. Its declaration in LUSTRE is

```
node ConnectionAlarm(up:bool) returns (alarm:bool);
```

We now explain its behaviour using the diagram below.



The module sets ALARM to true when one of the following happens:

- The number of "short drops" is ≥ 7. A short drop is a loss of communication (UP is false) lasting ≥ 2 steps but less than 10 steps.
- A "long drop" happens. A long drop is an absence of UP lasting ≥ 10 steps.

Note that a long drop is counted as a short drop as well. In the diagram above at step 6 we observe a short drop because UP is absent for 2 consecutive steps (at moments 5 and 6). For the same reason we observe a short drop at time steps 9, 11, 13, 15, 17. In step 17, we observe a long drop, since UP is absent for 10 steps (since moment 8). Observing a long drop, the module raises ALARM. Note that once the alarm is raised, it should stay high from now on forever (see steps 18, 19).

Task. Implement the module. Verify it using KIND2 against the property "once the alarm is set, it stays set forever". Your report should contain:

- the code of that module;
- the diagrams[3] illustrating the short and the long drops, and the alarms caused (a) by too-many short drops, and (b) by a long drop;
- how would you write that property in LTL? Add your LTL formula into the report;
- the outcome of the verification step[4].

[3] To be able to see `short_drop` and `long_drop` boolean flags (or whatever you name them), create a copy of module `ConnectionAlarm` which has those flags in the output.

[4] For instance, provide a screenshot.

Task L2

Implement the component

```
node BrakeCableActuator(brk:real; up:bool) returns (clinch:bool);
```

The component is inspired by the description in the intro, but slightly more involved. It should monitor the connection using `ConnectionAlarm` that you've implemented in the previous task. When the cyclist presses the brakes hard, CLINCH should be engaged. It should also be engaged when the cyclist brakes softly and there is a connection alarm. Note that CLINCH might be not engaged when UP is false for a short duration: we assume that such short durations do not affect the driving.

Task. Implement the module. Verify the following properties using KIND2:

- whenever the cyclists brakes hard, the clinch should happen;
- whenever the clinch happens, BRK is not zero.

(Feel free to add more properties.) Add to your report the code of the module and the outcome of the verification steps.

Task L3

Implement the component

```
node LossyChannel(noise,brk:real; up:bool) returns (brk_rcvd:real);
```

It is used to model the nondeterministic behaviour of the communication channel when the connection drops. When UP is false, the value BRK sent by the brake sensor can be damaged during the transmission, thus BRK_RCVD can be anything. We model this using a new input signal NOISE. The component relays BRK to the output when UP is true, otherwise it outputs anything (NOISE).

Task L4

Implement the component

```
node MotorController(brk_rcvd:real; up:bool; human,speed,battery:real)
returns (asst_lvl:real);
```

The controller should monitor the connection status using the module `ConnectionAlarm`. Note that when UP is false, the controller cannot trust the value of BRK_RCVD. The behaviour of the controller was described in the introduction, but here we will use a slightly different logic, which depends not only on the connection status but also on the connection alarm. First, if the connection alarm is set, the controller should not engage the motor neither in braking nor in assisting. Then, when the alarm is low, the controller should satisfy the following properties.

- If the connection is down, the motor cannot be used for braking (but can be used for assistance).

- There is no assistance on speeds above 7 m/s.
- There is no assistance when the battery is 0 (but regenerative braking can occur).
- When regenerative braking is engaged, ASST_LVL equals $-$BRK_RCVD.
- When assistance is engaged, ASST_LVL equals HUMAN.

Task. Implement a controller satisfying these properties. If you find that certain cases are not described, make a reasonable design choice. Include into report the code and explanation for your design choices (if any).

Task L5

We provide an implementation of the component

```
node System(up:bool; noise,brk,human,speed,battery:real)
returns (clinch:bool; asst_lvl:real);
```

This component represents the whole system, with inputs coming from the cyclist (HUMAN, BRK) and the environment (NOISE, UP, SPEED, BATTERY).

Task. Add and verify the following properties:

- if the connection alarm is true and the braking happens, brake-cable clinch must happen (i.e., CLINCH must be true);
- when the connection alarm is true, the regenerative braking cannot happen;
- regenerative braking occurs only if the braking is requested by the human;
- when the speed is above 7 m/s, the assistance is negative or zero;
- the assistance level is always equal to one of the following values: 0, -BRK, HUMAN.
- if the braking and assisting are both requested at the same time and the connection is up and there is no alarm, then the motor should engage the regenerative braking.

Your report should contain the code with these properties added and the outcome of the verification step.

4 Strix: synthesis of discrete-time systems

Coming soon...

5 UppAal TiGa: verification and systhesis of timed automata

Coming soon...

6 Xcos: simulation of dynamical systems

Your goal is to simulate in Xcos a bike trip from Parc Ten Reuken to Parc de la Héronnière. Figure 3 shows a map with its elevation profile.

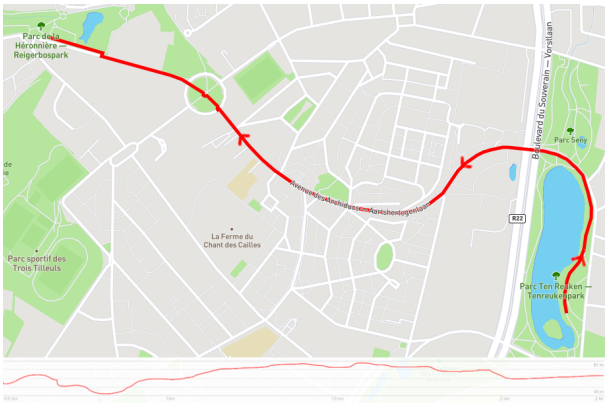


Figure 3: Track from Parc Ten Reuken to Parc de la Héronnière; total length 1800m.

In the folder on Xcos you will find a project template. The template contains all the necessary components, some already implemented and some are left for you to implement. The main components are: battery, human, electric assistance, and bike dynamics. You need to implement the last two components.

The bike dynamics is described by the following equation (assuming the speed $v > 0$):

$$m\frac{dv}{dt} = F_e + F_h - mg\sin\alpha - C_{rr}mg\cos\alpha - kv^2,$$

where F_e is an electric assistance, F_h is a human effort, $mg\sin(\alpha)$ accounts for the road gradient, and the last two components are the rolling-resistance force and the air-resistance force. We use the following values: $k = 0.25$, $m = 70$, $C_{rr} = 0.003$.

Task X0. Answer, with explanations, the following questions.

(a) Is the function for $\frac{dv}{dt}$ Lipschitz-continuous? (When answering the question (a), you can drop the multipliers m, mg, $C_{rr}mg$, k in the function.) Here we assume that F_e, F_h, α are inputs. Start your answer with the definition of the Lipschitz-continuity from Rajeev's book. Then state what do you really want to show (continuous or not continuous?), and show it from the definition. Hint: you can rely on the following claim: if a function $f(x, y) = g(x) + h(y)$ can be expressed as a sum of two (or more) functions over individual variables, and one of those functions is not Lipschitz-continuous, then $f(x, y)$ is not Lipschitz-continuous.

(b) Let us assume that $F_e = 0$, $\alpha = 0$, and $F_h = 275exp(-0.5v)$. Using Xcos, draw a plot "derivative vs. speed" (speed on the X axis and derivative on the Y axis); save your work into `x0.zcos`. Find an equilibrium speed of the bike using the diagram. (Do not forget the component $C_{rr}mg\cos\alpha$.) (Hint: it is a very strong rider.) Are these equilibrium speeds stable? Asymptotically stable? Justify the answers using your diagram.

Your report should contain the answers together with the plots, and the file `x0.zcos`.

Task X1. Implement the behavioural model of a rider. The rider pedals on speeds below 10m/s and brakes on higher speeds. See the block "human behaviour" in the template for more details. Draw the diagram "effort vs. speed" with assistance and braking on the Y axis and speed on the X axis. To this end, create the file `x1.zcos` containing only the rider model and the scopes necessary for the diagram. Your report should contain the diagram "effort vs. speed" and the file `x1.zcos`.

Task X2. Implement the dynamics model of the bike, as described by the differential equation above. Simulate your model, observe the diagrams. How much time does the cyclists need to reach the destination? By dividing the total distance (1800m) by the total time, calculate the average speed of the bike without the electrical assistance. Add your answers and the diagrams to the report. Save your solution into `x2.zcos`.

Task X3. (Copy the file `x2.zcos` into `x3.zcos` and work on it.) Implement the basic model of electrical assistance. It repeats the human effort ($F_e = F_h$) when the battery level is above 0.4 (40%), and otherwise provides $F_e = 0.75F_h$. The assistance should be turned off on speeds above 7 m/s. Additionally, the regenerative braking should be activated when the cyclist requests it, and the braking overrides the assistance (has a higher priority). What is the new journey time and average speed? Add your answers and simulation diagrams to the report. Save the solution into `x3.zcos`.

Task X4. (Copy the file `x3.zcos` into `x4.zcos` and work on it.) Implement the advanced model of electrical assistance, which additionally has a hysteresis behaviour around 6...7 m/s. This means: if the assistance is 0 and the speed is below 7 m/s, then we turn on the assistance. Once the speed reaches 7 m/s, turn off the assistance until the speed falls below 6 m/s, then the assistance turns on again, and so on. You might want to use the block `hysteresis` to implement this behaviour. How do diagrams for tasks X3 and X4 differ? Is the travel time/speed are affected? Add the answers and simulation diagrams to your report. Save your solution into `x4.zcos`.

References

[1] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. Developing UPPAAL over 15 years. *Softw. Pract. Exp.*, 41(2):133–142, 2011. URL: https://uppaal.org/.

[2] Nicholas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. The synchronous data flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, 1991. URL: https://www-verimag.imag.fr/The-Lustre-Programming-Language-and.

[3] https://scilab.org. Scilab and Xcos. URL: https://scilab.org.

[4] Michael Luttenberger, Philipp J. Meyer, and Salomon Sickert. Practical synthesis of reactive systems from LTL specifications via parity games. *Acta Informatica*, 57(1-2):3–36, 2020. URL: https://strix.model.in.tum.de/, doi:10.1007/s00236-019-00349-3.