Christian Schilling
Jakob Ø. Hansen
Kim G. Larsen
Martijn Goorden
Max Tschaikowski
Milad Samim

Models and Tools for Cyber-Physical Systems
Exercise sheet 10
WITH SOLUTIONS

Exercise 1: Cruise Control

Consider the car model from the course

$$\frac{d}{dt}x(t) = v(t) \qquad\qquad \frac{d}{dt}v(t) = -\frac{k}{m}v(t) + \frac{1}{m}F(t),$$

where x is the position, v the velocity and F the force applied in case of mass m and friction coefficient k. Starting at position $x(0) = 0$, we wish to accelerate the car to the speed of 20 (m/s) using a proportional controller with appropriate gain K_p for $m = 1000$ (kg) and $k = 50$.

(a) Assuming that we have no feedforward control and measure only the velocity of the car, provide a continuous-time component in block diagram form.

(b) Create a continuous-time component of the proportional controller in block diagram form and indicate how the controller component is connected to the car model component. Finally, combine them together into a single continuous-time component.

(c) Using your numerical solver from Exercise sheet 8, tune your controller, i.e., find an appropriate value for K_p, such that the car attains the speed of $v = 20$ (m/s) within $t = 100$ (s). Describe the quality of your controller in terms of steady state error, settling time, overshoot and rise time.

(d) Based on your observations from the pure proportional controller, design a PI controller that improves the results. Tune your chosen PI architecture.

Assume now that there is a car ahead whose position at t (s) is given by $y(t) = 150 + 15t$ (m). We wish to extend the foregoing continuous-time component to a hybrid component which

- stops accelerating as soon as the distance to the car ahead falls below 50 (m).
- activates the previous controller once the car ahead is more than 100 (m) away.

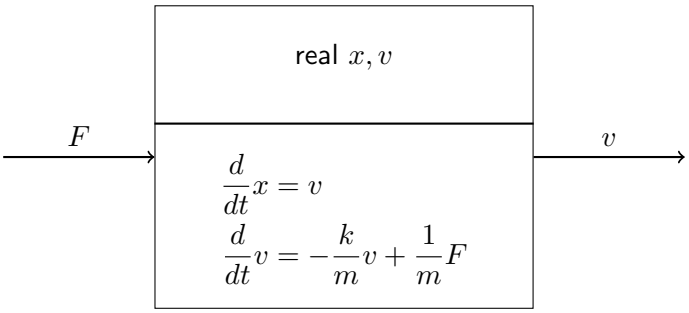
(e) Create the aforementioned hybrid component in block diagram form.

(f) Implement the hybrid controller by extending the numerical solver from Exercise sheet 8. By running simulations, decide whether the performance of your controller is good. If not, argue why not and try to improve. [Hint: May it be beneficial to reset the integral error for certain mode changes?]

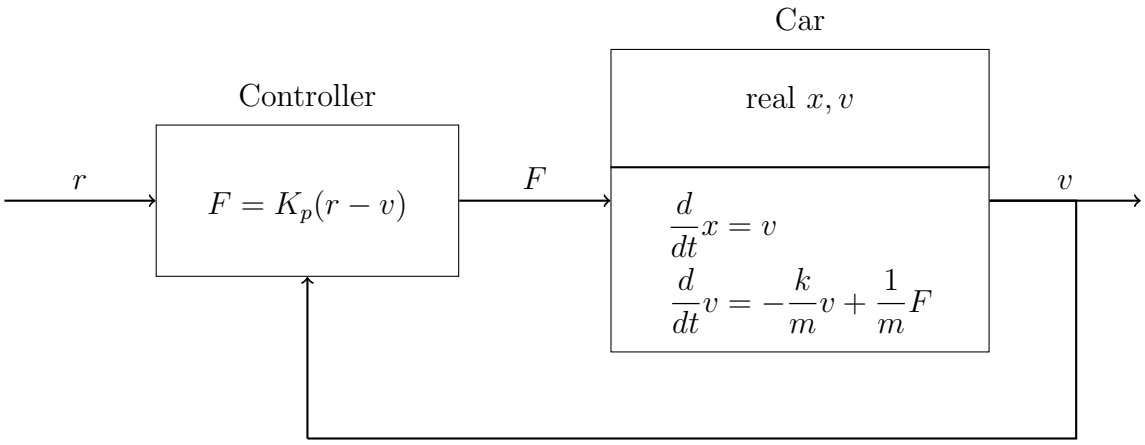
(g) Assume now that the controller does not stop accelerating the car once the distance to the car ahead falls below 50 (m). What is the maximal force F_h with which we could accelerate the car before a crash occurs?

Solution sketch

(a) The model is already given as a set of first-order differential equations, so we do not have to do any rewriting. The continuous-time component of the car is shown below.



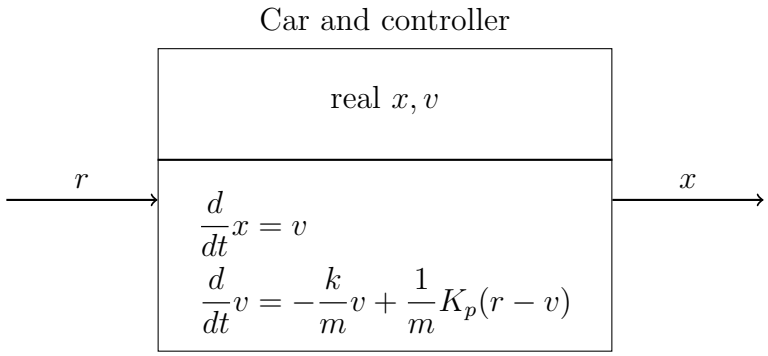
(b) The continuous-time component of the controller is shown below. Note that a proportional controller does not have an internal state.



The combined continuous-time component of the car and the controller is shown below.

(c) You can use the solution code from Exercise sheet 8 where you define function `f` to be as follows.

```
1  def f(x, t):
2    m = 1000
3    k = 50
4    Kp = 300
5    r = 20
```




```
6   der_x = x[1]
7   der_v = -k / m * x[1] + Kp * (r - x[1]) / m
8   return [der_x, der_v]
9
```

Figure 1 shows a plot of the velocity v (blue) for a gain of $K_p = 300$.

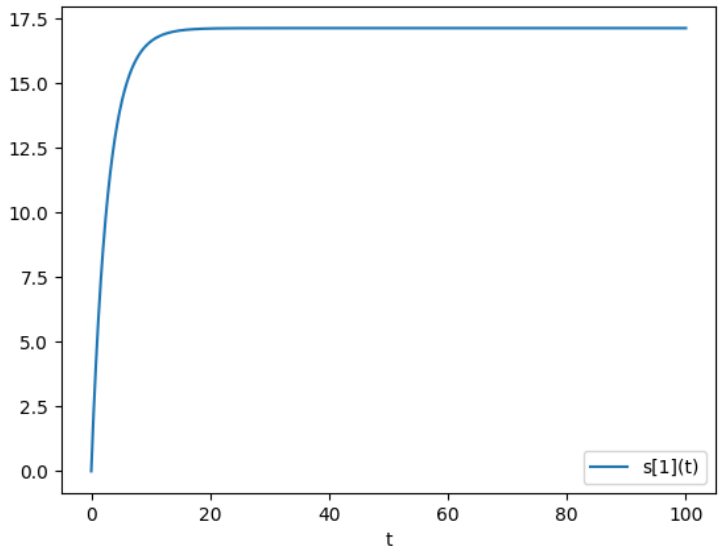


Figure 1: Car model state trajectory with only proportional controller $K_p = 300$.

One can observe that the proportional controller results in no overshoot but a significant steady-state error. The magnitude of the gain K_p determines the settling time (higher values of K_p result in shorter settling times) and the steady-state error (higher values of K_p result in smaller steady-state errors) of the controller system. In theory, one needs to have $K_p = \infty$ to have no steady-state error, but that is unrealistic for actual implementation.

(d) The steady-state error observed in the previous question can be solved by introducing the integral controller. To implement the integral controller, we need to integrate the error. We

can use the following observation to implement this integral in numerical solvers.

$$\int e\ dt = I \qquad \text{need to calculate } I$$
$$\frac{d}{dt}\int e\ dt = \frac{d}{dt}I \qquad \text{differentiate both sides}$$
$$e = \frac{d}{dt}I \qquad \text{simplifying the equation}$$

Observe that we effectively have created an additional internal state variable (for the controller), see also the block diagram in Figure 6.15 in the book of Alur.

You can use the solution code from Exercise sheet 8 where you define function `f` to be as follows. Note that our initial state now becomes `x0 = [0, 0, 0]`.

```
def f(x, t):
  m = 1000
  k = 50
  Kp = 100
  Ki = 5  # Note that the integral controller needs a third state
  r = 20
  F = Kp * (r - x[1]) + Ki * x[2]
  der_x = x[1]
  der_v = -k / m * x[1] + F / m
  der_I = (r - x[1])
  return [der_x, der_v, der_I]
```

Figure 2 shows a plot of the velocity v with gains of $K_p = 100, K_I = 5$. Observe that the PI controller no longer has a steady-state error (which was our intention with introducing the integral controller).

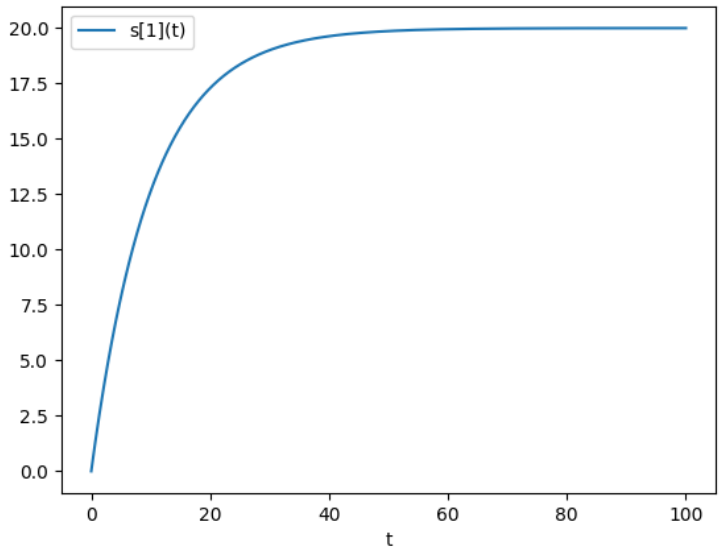


Figure 2: Car model state trajectory with PI controller $K_p = 100, K_I = 5$.

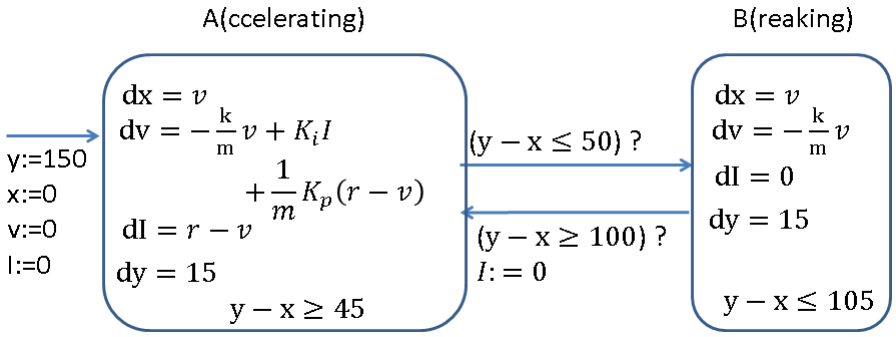


Figure 3: Hybrid cruise controller component.

(e) The hybrid component is visualized in Figure 3. Note that we reset the integral error each time the PI controller is re-activated. Moreover, the guards are triggered by the values 50 and 100. As invariants, instead, we used 45 and 105 to ensure that numerical algorithms like the Runge-Kutta method can be used to generate valid executions of the hybrid component. [For the mathematically inclined: Can you argue why?]

(f) To implement this hybrid controller, first note that we cannot identify the mode with just the continuous states. For example, the state where $y(t) - x(t) = 75$ is a valid state for both `Mode A` as well as `Mode B`. Hence we introduce a global variable `mode` outside `f`[1]. A first implementation of the hybrid controller could be as follows, where we implement the hybrid behavior in the function `f`. Denoting by F_h the force with which the car is being accelerated when the PI controller is inactive, the code below implements the desired hybrid cruise controller when $F_h = 0$.

```
mode = 1

def f(x, t):
  global mode

  m = 1000
  k = 50
  Kp = 100
  Ki = 5  # Note that the integral controller needs a third state.
  r = 20
  Fh = 0

  y_front = 150 + 15 * t

  # Check whether we have to switch mode.
  if mode == 1 and y_front - x[0] < 50:
    mode = 2
  if mode == 2 and y_front - x[0] > 100:
    mode = 1

  # Get the force depending on the mode.
  if mode == 1:
```

[1]We don't pass the value of mode as input and put of the `f` function so we can (hopefully) keep the implementation of the numerical solver unchanged.

```
    F = Kp * (r - x[1]) + Ki * x[2]
  else:
    F = Fh

  der_x = x[1]
  der_v = -k / m * x[1] + F / m
  der_I = (r - x[1])
  return [der_x, der_v, der_I]
```

Figure 4 shows a plot of the velocity v with gains of $K_p = 100, K_I = 5$ (same as before) and $F_h = 0$. Observe that the controlled behavior is unstable! Note that for any other value of F_h the system remains unstable.

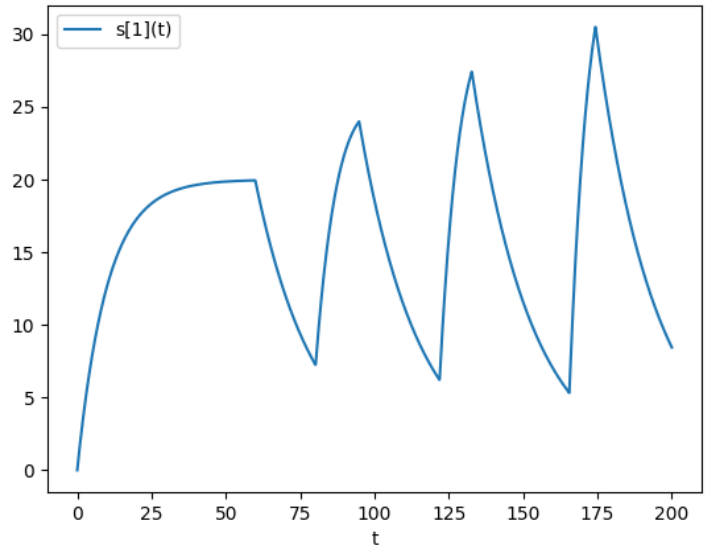


Figure 4: Hybrid car model state trajectory v with PI controller $K_p = 100, K_I = 5$ and $F_h = 0$.

The main cause of the instability is that the integral error $I = \int e \, dt$ keeps accumulating while the PI controller mode is not active. Ideally, we want to reset I every time we switch back to the PI mode. We can achieve this by separating the discrete mode switching from the continuous dynamics in `f`. Hence the numerical solver needs to be adjusted slightly. An implementation could be as follows. We use `switch=None` to keep the numerical solver backwards compatible for pure dynamical systems.

```
def euler_method(f, x0, t0, h, N, switch=None):
  t = [t0]
  result = [x0]
  for i in range(N):
    xi = result[-1]

    # Check for mode switching.
    if switch is not None:
      xi = switching(xi, t[-1])

```

```
    der = f(xi, t[-1])
    t.append(t[-1] + h)
    step = []
    for j in range(len(der)):
      step.append(xi[j] + h * der[j])
    result.append(step)
  return t, result

def f(x, t):
  global mode
  m = 1000
  k = 50
  Kp = 100
  Ki = 5  # Note that the integral controller needs a third state.
  Fh = 0
  r = 20

  # Get the force depending on the mode.
  if mode == 1:
    F = Kp * (r - x[1]) + Ki * x[2]
  else:
    F = Fh

  der_x = x[1]
  der_v = -k / m * x[1] + F / m
  der_I = (r - x[1])
  return [der_x, der_v, der_I]


def switching(x, t):
  global mode
  y_front = 150 + 15 * t

  # Check whether we have to switch mode.
  if mode == 1 and y_front - x[0] <= 50:
    mode = 2
  if mode == 2 and y_front - x[0] >= 100:
    mode = 1
    # We switch back to PI controller, so we reset the integral
                                        term.
    x[2] = 0
  return x
```

Figure 5 (left) shows a plot of the velocity v with gains of $K_p = 100, K_I = 5$ and $F_h = 0$ but now resetting the integral term each time the car switches to the PI mode. As can be seen, this now ensures that the hybrid system is stable. Don't forget that a good cruise controller does not collide in with the car in front of it. Therefore, Figure 5 (right) shows a plot of the position x of the car and y of the car in front of it. As can be seen, the blue line never actually touches the red line, hence the cars do not collide. Once can also confirm this by plotting the distance between the front car and the car with the hybrid cruise controller $y(t) - x(t)$ as shown in Figure 6.

(g) By considering different simulations for different values of F_h, it can be noted that we bump into the car ahead once F_h is around 650 or higher. Figure 7 shows the relative distance

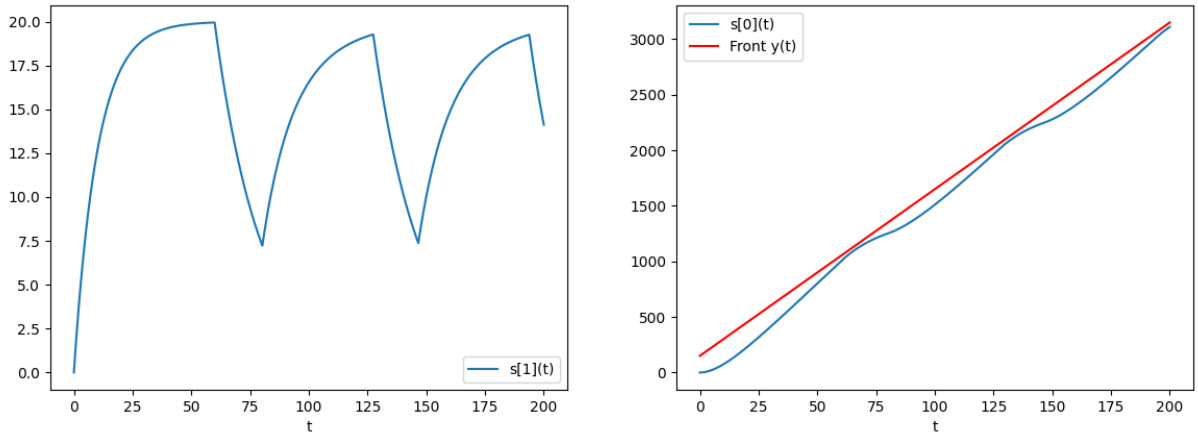


Figure 5: Hybrid car model state trajectory v (left) and the positions x and y (right), both with PI controller $K_p = 100, K_I = 5$ and $F_h = 0$ including the reset of the integral error term.

$y(t) - x(t)$.

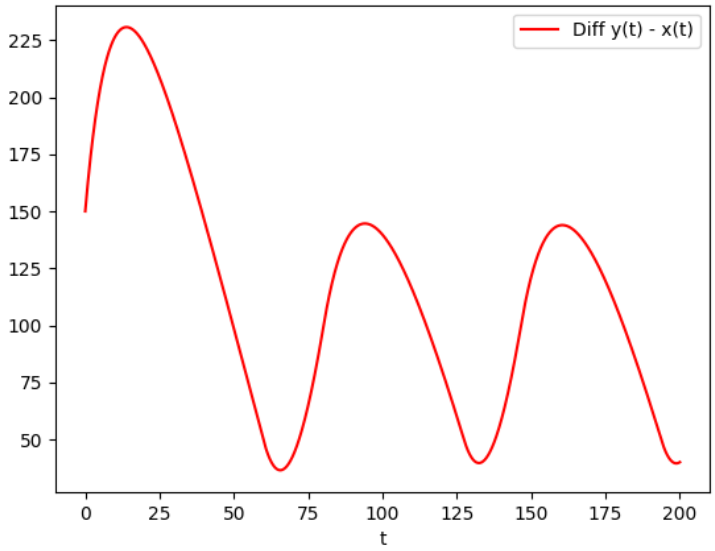


Figure 6: Distance between the car with the hybrid cruise controller and the car in front of it.

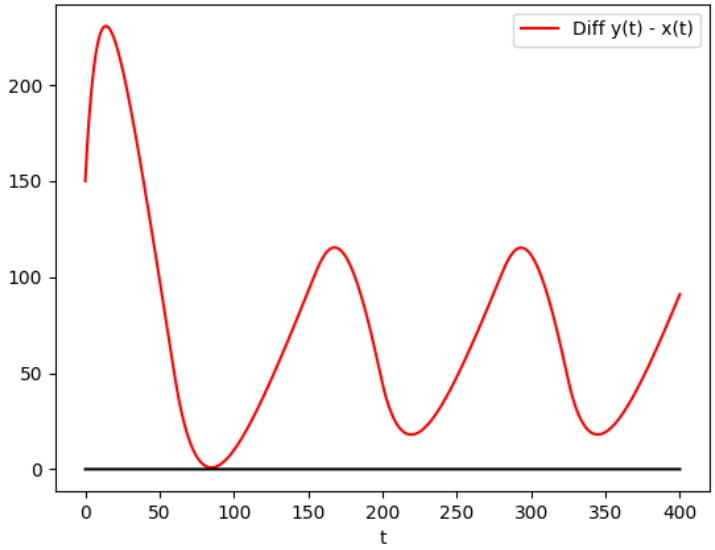


Figure 7: Relative distance $y(t) - x(t)$ between the cruise-controlled car and the car in front of it for a force $F_h = 650$ (N).