A Hybrid Event-B Study of Lane Centering

Richard Banach and Michael Butler

Abstract A case study on automotive lane centering control is examined in Hybrid Event-B (an extension of Event-B that includes provision for continuously varying behaviour as well as the usual discrete changes of state). This allows aspects beyond the reach of a discrete Event-B treatment to be more deeply investigated. Lane centering offers particular challenges concerning how the monitoring of continuously varying continuous functions is handled and how this interacts with discrete mode-level decision making.

1 Introduction

These days, an ever increasing proportion of the equipment we interact with all the time involves digital control of analogue phenomena. The repercussions of this trend include an increasing number of devices that could do real harm if they malfunctioned. Besides this, the capabilities of digital control bring with it ever increasing complexity, with the risks that this inevitably brings.

It is by now well accepted that formal techniques, appropriately deployed, can help with both of these issues. However, in the main, formal techniques are strongly focused on purely discrete reasoning, dealing poorly with continuous behaviours. The *hybrid* and *cyberphysical* systems we speak of (see, e.g. [30, 33, 2, 32, 12]) are rather poorly served by conventional formal techniques. See, for example, the extensive review in [14], which covers a large number of approaches and their accompanying tool systems, including: Charon [5, 6], CheckMate [29], HSolver [25],

Richard Banach
School of Computer Science, University of Manchester, Oxford Road, Manchester, M13 9PL, U.K., e-mail: banach@cs.man.ac.uk

Michael Butler
School of Electronics and Computer Science, University of Southampton, Highfield, Southampton, SO17 1BJ, U.K., e-mail: mjb@ecs.soton.ac.uk

HyTech [7, 17], Modelica [23] and Simulink [22], among others. Although these techniques have enjoyed success over the years, most of them are either limited in their expressivity (typically driven by a desire to achieve decidability for a given language fragment), or they lack rigour by comparison with most discrete techniques (for example by employing simulation as a strategy for verification).

An exception to this general trend is KeYmaera (see [1, 24]). This is a system that combines formal proof of a quality commensurate with contemporary formal techniques, with the kind of continuous behaviour that is needed in the description of genuine physical systems. KeYmaera concentrates on the verification of properties of a defined system model. In this sense its focus is different from the refinement-based approach of the B-Method —our concern in this paper— since, although verification of some properties can often be seen as a kind of refinement, KeYmaera does not emphasise refinement as a *development approach* in the way that the B-Method emphatically does.

The increasing interest in hybrid and cyberphysical systems just noted, and the desire to achieve high dependability in their development, has led to attempts to use Event-B [3] for their development, for example during the DEPLOY project [16]. However it is fair to say that the absence of the ability to deal with continuous phenomena directly within the discrete Event-B framework during such work is keenly felt.

This need to deal with continuous phenomena within Event-B has prompted the development of an extension, Hybrid Event-B [11], treating discrete and continuous behaviour on the same footing. In this paper, we apply Hybrid Event-B to a case study previously done in discrete Event-B: the modelling of an automotive lane centering system first investigated during a collaboration between the second author and GM Research. Among other things, such revisiting of the case study can confirm the suitabilty of the Hybrid Event-B formalism to adequately deal with facets of the original discrete Event-B treatment that were less than ideal.

The lane centering case study explored in this paper forms a natural accompaniment to another automotive-based case study, on cruise control, done by the present authors in [10]. The contrast between the two is instructive. In cruise control, the problem can be seen as completely self-contained. The desired speed is set, and the car can easily monitor progress towards it, and how well it is being maintained. This gives rise to a situation in which convincing invariants can be established. In lane centering though, the road ahead is unpredictable in principle. So the problem is not self-contained any more. Progress towards the desired goal is constantly dependent on external information, which impacts the kind of invariants one can establish. These aspects are discussed in detail in Section 4.

In contrast to KeYmaera, there is at present no dedicated tool support for Hybrid Event-B. So a further benefit of case studies like this one is to confirm that Hybrid Event-B contains the right collection of ingredients for industrial scale modelling, before the serious investment in extending the Rodin Tool [4, 26, 27] is made.

The rest of this paper is as follows. Section 2 overviews the lane centering system, including how our case study differs in detail from the requirements tackled in the previous version. Section 3 then gives a pure mode based model for the lane cen-

tering control system, while Section 4 refines it to a model where both modes and continuous behaviour are fully defined. These models are related to one another using a suitable refinement. Section 5 discusses various issues raised by the preceding material, and concludes.

2 Lane Centering Controller Overview

A lane centering controller (LCC) is a software system which automatically keeps a car correctly aligned with respect to the centre of the lane in which it is travelling. It does this based primarily on information received from a path generator unit, which calculates the target and predicted paths of the car for a short period into the future, based in turn on information from an image processing unit (which looks for lane markings in the road) and other data available from the engine management system. The car's driver can engage the LCC, which will then attempt to discharge its lane centering obligations, but the LCC is an assistance system rather than a safety system, so when it is unable to perform its function, it issues a warning, at which point the driver must resume responsibility for steering the car. In addition, it must disengage upon request of the driver (or in case of a system fault). All of these are safety properties.

The left hand side of Fig. 1 gives a diagrammatic view of the LCC architecture, taken from [34]. We see the image processing unit, which feeds information about the car's lateral position in the lane, and information about the road curvature, to the path generator unit. This then receives further information regarding the yaw angle and rate, the steering angle, the lateral and longitudinal speed, and the driver-selected offset. From all this information, the target path, the predicted actual path and a safety margin, are calculated and fed to the LCC itself, which generates the actuated steering angle for keeping the car on target.

The right hand side of Fig. 1 illustrates how some of these parameters relate to the movement of the car. In reality, the LCC only works when an adaptive cruise control system is actively controlling the speed of the car, leading to a coupling between the two systems. As noted in the Introduction, we have considered cruise control earlier in [10], but for simplicity, in this paper we neglect the coupling between it and the LCC. In the same vein, we neglect the lateral offset parameter, and a number of more subtle considerations concerning circumstances under which the LCC has to curtail its activity because the information it has is insufficient, or is of insufficient quality. We cater for all such cases by switching the LCC off or by taking an error transition, offering some further discussion in the Conclusions.

With these caveats in place, in Fig. 2 we see the state transition diagram for the LCC at an intermediate level of description, comparable to, but simplified from, the LCC description in [34].

The LCC starts in the *OFF* state, from where it can be made to *Sw*itch*On*, putting it into the *STANDBY* state. In this state the LCC can be made to *Sw*itch*Off*. Alternatively, the driver may try to engage the LCC. If the motion of the car is too

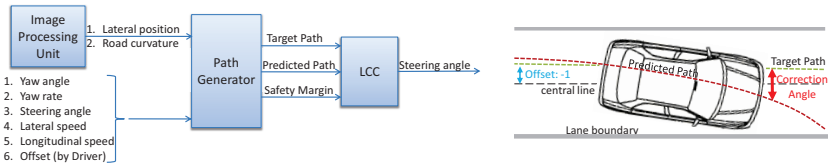

Fig. 1 Architecture of a lane centering controller (LCC) on the left. On the right, a schematic illustration of the geometric elements that figure in LCC computations.

*UnAl*igned with the lane markings for the LCC to take over, the state remains *STANDBY*, but if the motion is adequately *Aligned*, the the LCC goes into the *ACTIVE* state, in which it actively steers the car.

Normal LCC working can be overridden by putting the *Ind*icator*On*, which signifies that the driver will shortly turn out of the lane. Alternatively, the driver may try to steer the car manually by forcing the steering wheel out of the orientation determined by the LCC. In either of these cases the state becomes *OVERRIDE*.

Putting the *Ind*icator*Off*, or ceasing to try to steer the car manually causes the LCC to try to *Resume* normal working in the *ACTIVE* state — provided the car is adequately aligned; if it is not then the *O*ut*O*f*Al*ignment transition switches the LCC off.

In addition to these ways of working, the driver can *Sw*itch*Off* the LCC in the *ACTIVE* and *OVERRIDE* states (as well as *STANDBY*), and when in any of these three states, the LCC can undergo an *Error* transition into the *ERROR* state, for example if the path generator unit or image processing unit undergo some failure, or their information is too low quality to be relied on safely.

We take it for granted that the state transition diagram in Fig. 2 does an effective job of modelling the top level requirements of the LCC system. For example, it addresses the requirement that when it is switched on, the LCC does not start working immediately, but waits in the *STANDBY* state until instructed to take control of the steering. Likewise, the transitions between the *ACTIVE* and *OVERRIDE* states cor-

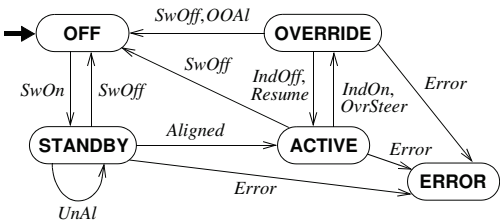

Fig. 2 The state transition diagram for a simplified LCC.

respond to requirements embodying assumptions about the appropriateness of the use of the accelerator during LCC working. And so on.

In the following sections, we will develop a series of Hybrid Event-B machines to capture this design. We explain the technicalities of Hybrid Event-B as we go.

3 Lane Centering — Top Level Mode Oriented Control

In this section we start the development of the LCC system. Fig. 3 contains a straightforward translation of the state transition diagram in Fig. 2 to the discrete, mode oriented part of Hybrid Event-B. The nature of the translation is rather self-evident, so we restrict ourselves to a few essential comments.

The state of the system is recorded in the *mode* variable, which is restricted to the values we mentioned above (this observation constituting the sole invariant at this level of abstraction), and is initialised to *OFF*. Beyond this, the events of the model simply record the state changes permitted by Fig. 2 in a relatively obvious way.

The events merit three further technical observations at this point. Firstly, the design is *aggressive*, in the sense that although each event is guarded by a condition that permits its sensible execution, the states are not protected against inappropriate requests from the environment. In each state, only those events can be executed that make sense in our model, whereas in the real world, the driver, could, if a little unwisely, engage controls that one would not expect to see engaged there. At a higher level of abstraction than we model in this paper, the system would have to be protected against such stimuli.

Secondly, although we only appear to have discrete transitions, these *mode* events each have an input, whose value is essentially just the event's name (except for *UnAl* and *Aligned* where there is a genuinely nondeterministic choice), which is furthermore never used. The explanation for this is that Hybrid Event-B models real time behaviour, which therefore means that the occurrence times of mode events (which execute instantaneously) must be defined. For convenient modelling, Hybrid Event-B stipulates that unscheduled stimuli from the environment —such as the otherwise unspecified timings of occurrences of our mode events— are modelled by the arrival of inputs to the relevant mode events.

Thirdly, and following on from the previous point, there is also another, *pliant* event, *PliTrue*. Its nature is signalled by the 'STATUS pliant' tag. Unlike mode events, pliant events describe periods of continuous behaviour, taking place over nonempty intervals of time, in between mode events (occurrences of which are thus isolated in time). The semantics of Hybrid Event-B stipulates that a mode event can prempt a running pliant event as soon as the mode event becomes enabled, and upon the completion of the mode event, some pliant event should have become enabled and should be selected for execution to take forward the continuous evolution of the system.[1]

[1] This *eager* semantics for preemption of pliant events by mode events gives another reason why an *asynchronous arrival* semantics for inputs to mode events is needed. In discrete Event-B as soon

```
MACHINE LCC_0
VARIABLES mode
INVARIANTS
  mode ∈ {OFF, STANDBY, ACTIVE,
          OVERRIDE, ERROR}
EVENTS
  INITIALISATION
    STATUS ordinary
    BEGIN
      mode := OFF
    END
  SwOn
    STATUS ordinary
    ANY in?
    WHERE in? = swOn ∧ mode = OFF
    THEN mode := STANDBY
    END
  SwOff
    STATUS ordinary
    ANY in?
    WHERE in? = swOff ∧ mode ∈
      {STANDBY, ACTIVE, OVERRIDE}
    THEN mode := OFF
    END
  UnAl
    STATUS ordinary
    ANY in?
    WHERE in? = tryAct ∧
      mode = STANDBY
    THEN skip
    END
  Aligned
    STATUS ordinary
    ANY in?
    WHERE in? = tryAct ∧
      mode = STANDBY
    THEN mode := ACTIVE
    END
  IndOn
    STATUS ordinary
    ANY in?
    WHERE in? = indOn ∧
      mode = ACTIVE
    THEN mode := OVERRIDE
    END
  IndOff
    STATUS ordinary
    ANY in?
    WHERE in? = indOff ∧
      mode = OVERRIDE
    THEN mode := ACTIVE
    END
  OvrSteer
    STATUS ordinary
    ANY in?
    WHERE in? = ovrSteer ∧
      mode = ACTIVE
    THEN mode := OVERRIDE
    END
  Resume
    STATUS ordinary
    ANY in?
    WHERE in? = resume ∧
      mode = OVERRIDE
    THEN mode := ACTIVE
    END
  OOAl
    STATUS ordinary
    ANY in?
    WHERE in? = oOAl ∧
      mode = OVERRIDE
    THEN mode := OFF
    END
  Error
    STATUS ordinary
    ANY in?
    WHERE in? = error ∧ mode ∈
      {STANDBY, ACTIVE, OVERRIDE}
    THEN mode := ERROR
    END
  PliTrue
    STATUS pliant
    COMPLY INVARIANTS
    END
END
```

Fig. 3 Mode level description of lane centering control.

as an event has completed, its successor is enabled, since the state does not change in beteen event occurrences (which are assumed separated in time as a matter of *interpretation*). In Hybrid Event-B, real time figures explicitly, and a second mode event is not permitted to execute immediately

In our case, *PliTrue* stipulates in the COMPLY *INVARIANTS* clause merely that the continuous behaviour in between mode event occurrences should obey the invariants. With its help, the behaviour of the model is defined for all times after the initialisation point, since, having a trivial guard, whenever a mode event runs, *PliTrue* is re-enabled immediately afterwards.

4 Lane Centering — From Mode Control to Continuous Control

In this section we go beyond the pure mode oriented model of Section 3, to include a description of the continuous behaviour in the periods between occurrences of mode events, that a more complete model demands.

Our enhanced model appears in Fig. 4, completed in Fig. 5. We go through this element by element, giving, as we go, appropriate commentary, not only on the technical background of the case study itself, but also on relevant aspects of Hybrid Event-B and its semantics, especially when this is at odds with corresponding aspects of discrete Event-B, or raises interesting issues in its own right.

Fig. 4 starts with the INTERFACE *LCC_PG_IF* block. The INTERFACE construct is the Hybrid Event-B syntactic construct that enables different machines to be coupled to each other to form a larger system, working together in an integrated way, under the control of the required invariants. In our case study, we have taken the view that the LCC will 'own' those variables whose values it can control, whereas, variables which are set externally, or inferred indirectly from the car's environment via the image processing unit, will not be owned by the LCC.

The *LCC_PG_IF* interface contains these 'externally owned' variables, that need to be shared with the LCC. The first is a measure of the torque applied by the driver to the steering wheel, *trq*. The other two concern the target path and deviation from the lane centre, both of which need to be calculated from the visual inputs to the overall system.

For lack of space in this paper, we simplify matters considerably compared with the system architecture of Fig. 1. Thus the target path variable (for which one can imagine a large number of quite detailed formulations), is assumed to have already been converted to a target steering angle, θ_T. What we intend by this is the following: *if it were the case that the car was already exactly in the middle of the lane, then steering at θ_T would keep it exactly where the overall lane control system would want the car to go*. We accept that this amounts to a rather specific treatment of the requirements of an LCC system, since in reality, such requirements would speak in more detail about what constitutes the kind of road that the system is expected to be able to cope with, and what kind of behaviour would be expected under those conditions. A further simplification in our model is the assumption that if the path generator is sending the value of θ_T to the LCC and has not called an *Error* transition, then it is safe for the car to proceed down the path defined by θ_T at its current

after a first mode event, despite being enabled then. Giving the second mode event an asynchronous input gets round the problem.

velocity (e.g. the car has not driven into a muddy field that the image processing unit and path generator could not cope with).

The last variable in the *LCC_PG_IF* interface is d, which represents the measured deviation of the car from the lane centre. All of *trq*, θ_T and d are declared PLIANT, which means that they are subject to continuous change during the pliant transitions that interleave occurrences of the mode events at runtime.

The *LCC_PG_IF* interface also contains the invariants that these variables must obey. These say that *trq*, θ_T and d are all real-valued, and that they remain within statically determined bounds. Finally, the *LCC_PG_IF* interface also contains these variables' initialisations.

We come to the *LCC_*1 machine itself, which is a refinement of *LCC_*0 and CONNECTS to the *LCC_PG_IF* interface, making the latter's variables accessible. As well as the earlier *mode* variable, we have a pliant variable, θ, which holds the current steering angle (this being a representation, on the lines noted above, of the predicted path in Fig. 1). It is real-valued and bounded within static contraints.

Proceeding to the events, θ is initialised to an arbitrary value at system switch on time. After that, *PliDefault* is the refinement of the *PliTrue* event of *LCC_*0 that demands no more than invariant preservation during pliant transitions when it is enabled. We observe that *PliDefault* is enabled during all modes other than *ACTIVE*, so in effect, all the additional design that is embodied in the refined system model is targetted at just the *ACTIVE* state.

Given the last remark, some of the earlier mode events remain unchanged. This applies for instance to the *SwOn* and *SwOff* events, that cater for the driver deliberately turning the LCC on or off.

The next two events deal with the driver trying to actively engage the LCC system. To facilitate the discussion, we introduce the *ACTIVE state tolerance condition*: $ASTC \equiv (|\,d\,| < \Delta_d \wedge |\,\theta - \theta_T\,| < \Delta_\theta)$. We make the assumption that the LCC is not to be used when the current conditions are such that the target path (as determined by the path generator) is too different from the current path. We model this in a simplified way by demanding that the target and current steering angle do not differ by too much $|\,\theta - \theta_T\,| < \Delta_\theta$, and that the lane centre deviation d is not excessive $|\,d\,| < \Delta_d$. Hence the *ASTC*.

The *ASTC* enters our model in a number of places. Firstly, when the driver attempts to engage the LCC, he cannot know in advance whether the *ASTC* will be satisfied or not — aside from anything else, it depends on internal constants, Δ_d and Δ_θ, that he does not know and would not be able to make use of during driving even if he did. So in the *LCC_*0 machine earlier, this ignorance was catered for in a genuinely nondeterministic choice between the *UnAl*igned and *Aligned* events, indicated by having them both demand the same input value *tryAct*. In the *LCC_*1 machine, the choice between *UnAl* and *Aligned* is further refined by the truth of *ASTC*. If the *ASTC* is true, then *Aligned* is selected, whereas if the *ASTC* is false, *UnAl* is selected. On a technical note, we remark that the disjunction of the *LCC_*1 machine's *UnAl* and *Aligned* guards is equivalent to the disjunction of the *LCC_*0 machine's *UnAl* and *Aligned* guards, addressing both guard strengthening and relative deadlock freedom refinement requirements. Besides this, we note that *UnAl*, in

INTERFACE *LCC_PG_IF*
PLIANT trq, θ_T, d
INVARIANTS
$trq \in \mathbb{R} \wedge |trq| \leq MAX_{trq}$
$\theta_T \in \mathbb{R} \wedge |\theta_T| \leq MAX_\theta$
$d \in \mathbb{R} \wedge |d| \leq MAX_d$
INITIALISATION
$trq \in [-MAX_{trq} \ldots MAX_{trq}]$
$\theta_T := 0$
$d := 0$
END

MACHINE *LCC_1*
REFINES *LCC_0*
CONNECTS *LCC_PG_IF*
VARIABLES *mode*
PLIANT θ
INVARIANTS
$mode \in \{OFF, STANDBY, ACTIVE, OVERRIDE, ERROR\}$
$\theta \in \mathbb{R} \wedge |\theta| \leq MAX_\theta$
EVENTS
INITIALISATION
STATUS ordinary
REFINES *INITIALISATION*
BEGIN
$mode := OFF$
$\theta \in [-MAX_\theta \ldots MAX_\theta]$
END
PliDefault
STATUS pliant
REFINES *PliTrue*
WHEN $mode \neq ACTIVE$
COMPLY *INVARIANTS*
END
SwOn
STATUS ordinary
ANY *in*?
WHERE $in? = swOn \wedge mode = OFF$
THEN $mode := STANDBY$
END
SwOff
STATUS ordinary
REFINES *SwOff*
ANY *in*?
WHERE $in? = swOff \wedge mode \in \{STANDBY, ACTIVE, OVERRIDE\}$
THEN $mode := OFF$
END
UnAl
STATUS ordinary
REFINES *UnAl*
ANY *in*?, *out*!
WHERE $in? = tryAct \wedge mode = STANDBY \wedge \neg(|d| < \Delta_d \wedge |\theta - \theta_T| < \Delta_\theta)$
THEN $out! := BEEP$
END
Aligned
STATUS ordinary
ANY *in*?
WHERE $in? = tryAct \wedge mode = STANDBY \wedge (|d| < \Delta_d \wedge |\theta - \theta_T| < \Delta_\theta)$
THEN $mode := ACTIVE$
END
LCC_Active
STATUS pliant
REFINES *PliTrue*
WHEN $mode = ACTIVE$
SOLVE $\mathcal{D}\theta = -C(\theta - \theta_T) - Kd$
END
SwOff_Emrg
STATUS ordinary
REFINES *SwOff*
ANY *out*!
WHEN $mode = ACTIVE \wedge \neg(|d| < \Delta_d \wedge |\theta - \theta_T| < \Delta_\theta)$
THEN $mode := OFF$
$out! := BEEP$
END
IndOn
STATUS ordinary
REFINES *IndOn*
ANY *in*?
WHERE $in? = indOn \wedge mode = ACTIVE$
THEN $mode := OVERRIDE$
END
IndOff
STATUS ordinary
REFINES *IndOff*
ANY *in*?
WHERE $in? = indOff \wedge mode = OVERRIDE \wedge (|d| < \Delta_d \wedge |\theta - \theta_T| < \Delta_\theta)$
THEN $mode := ACTIVE$
END

Fig. 4 Enhancing the mode level lane centering control to fully continuous control. First part.

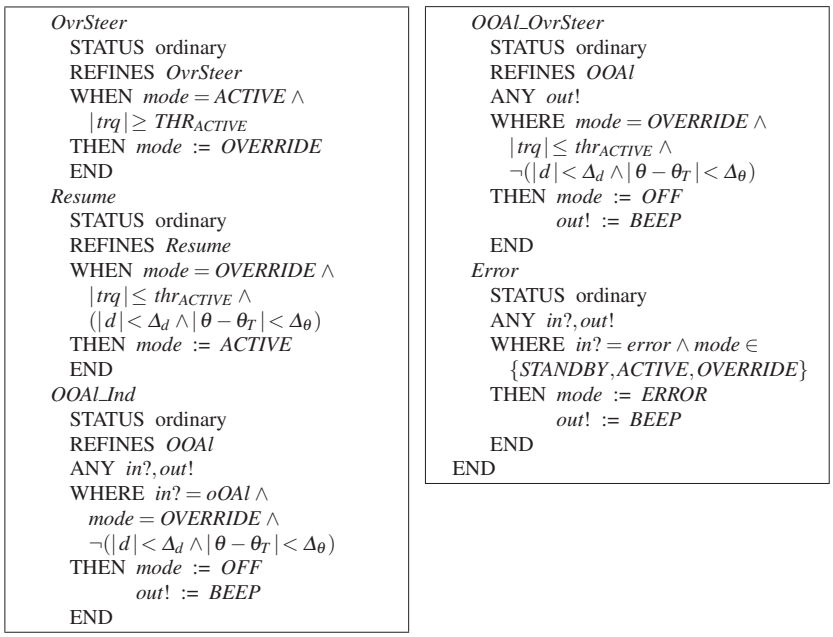

Fig. 5 Enhancing the mode level lane centering control to fully continuous control. Second part.

refusing to perform a function that the driver is expecting (i.e. to engage the LCC), alerts him to this fact via an audible alarm, represented in our model by sending a *BEEP* on the output variable *out*!.

Assuming successful engagement of the LCC, the *LCC_Active* pliant event runs. It has no nontrivial guards (aside from the *mode*), since the relevant initial conditions are already confirmed by the *Aligned* event that enables it. The job of *LCC_Active* is to align the car's path to the target path. For simplicity, we model this using a straightforward negative feedback control law applied to the current steering angle, indicated in the SOLVE clause of the *LCC_Active* event: $\mathcal{D}\theta = -C(\theta - \theta_T) - Kd$. Assuming that steering to the right, and deviation to the right from the lane centre, are both measured positively, the time derivative ($\mathcal{D}$) of the current steering angle θ is set to a negative linear combination of steering angle excess (of current over target) and deviation, tending both to bring the car closer to the lane centre, and to align the steering angle with the target steering angle.

We regard the θ_T and d parameters in the above ordinary differential equation (ODE) as external signals. Doing this reduces it to a linear ODE with inhomogeneous term; see [8, 31]. Ordinary differential equations of this form have a standard solution. In this case it is: $\theta(t) = \theta(\mathbb{t}_{\mathrm{L}})e^{-C(t-\mathbb{t}_{\mathrm{L}})} + \int_{\mathbb{t}_{\mathrm{L}}}^{t} e^{-C(t-s)}[C\theta_T(s) - Kd(s)]ds$, where $\mathbb{t}_{\mathrm{L}}$ is the symbol used in Hybrid Event-B to refer generically to the start time of any time interval during which a pliant event runs.

If, in the feedback system described, θ_T and d were both constant, then steady convergence of the car to the lane centre and of the current steering angle to the target steering angle would be guaranteed. A consequence of this would be that the *ASTC*, true at the start of an *LCC_Active* pliant transition, would be an invariant during any such transition. However, the fact that θ_T and d are both time dependent (preventing these terms from being extracted from under the integral in the $\theta(t)$ solution above) means that we have no such guarantee. If θ_T or d were to vary wildly enough during an *LCC_Active* transition, then the bounds in *ASTC* might be breached. Since we cannot prove that the bounds won't be breached, we have to make separate provision for the case where they are.

This is the purpose of the *SwOff_Emrg* mode event. It is enabled in the *ACTIVE* state when the *ASTC* fails. Since it has no input, like all mode events without input, it becomes eligible for scheduling as soon as its guard becomes true, and (if selected from among all the mode events whose guards become true at that moment, if there is more than one such event), it *preempts* the currently running pliant transition, *LCC_Active* in our case.

The effect of *SwOff_Emrg* is like that of *SwOff*, except that, being an event that is scheduled spontaneously rather than at the behest of the driver, there is an additional audible *BEEP* to alert the driver.

The next few events handle the driver's temporarily countermanding the *ACTIVE* state. Use of the indicator is modelled by *IndOn* and *IndOff*, mode events caused by the discrete actions of flicking the indicator on or off to enter or exit the *OVERRIDE* state. Of course, when the *ACTIVE* state is re-entered via *IndOff*, the *ASTC* must be checked.

The driver can also countermand the *ACTIVE* state by wilful use of the steering wheel. The *trq* variable tracks the torque applied to the steering wheel during a system run. If, in the *ACTIVE* state, this exceeds a threshold value THR_{ACTIVE}, the *OvrSteer* event changes the state to *OVERRIDE* while the driver takes control of the steering. Once the torque drops below the threshold value thr_{ACTIVE} again, the *Resume* event re-enters the *ACTIVE* state, having confirmed that *ASTC* holds. Normally, we would have that $thr_{ACTIVE} < THR_{ACTIVE}$ to prevent Zeno-like thrashing as the applied torque hovered around the threshold value.

Of course, since the *ASTC* has to hold if the *ACTIVE* state is to be re-entered, we have to contend with the possibility that it might not. Two new events cater for this. When using the indicator, event *OOAl_Ind*, scheduled when the indicator is flicked off by the driver (denoted using the $in? = oOAl$ input) but *ASTC* does not hold, switches the LCC off, alerting the driver with a *BEEP*. When using the steering wheel, event *OOAl_OvrSteer*, scheduled when the torque drops below thr_{ACTIVE} but *ASTC* does not hold, switches the LCC off, alerting the driver with a *BEEP*. We need two *OOAl* events at the *LCC_1* level, since one has an input and the other does not, even if their actions are the same. Both events refine the *LCC_0* level *OOAl* event, the different I/O signatures in the *OOAl_OvrSteer* case being handled by a suitable witness relation. Finally, we have the *LCC_1* level *Error* event, *BEEP*-enhanced compared with its *LCC_0* counterpart. It completes our survey of the *LCC_1* model.

5 Discussion and Conclusions

In the preceding sections, we overviewed the lane centering controller case study, previously examined using discrete Event-B in [34], with a view to creating an enhanced development using the richer facilities of Hybrid Event-B. We then presented such a development, based first on a mode level description in Section 3, subsequently refined to a description incorporating a definition of the required continuous behaviour in Section 4. The relatively simple modelling in these sections, partly a consequence of lack of space in this paper, raises two particular issues that deserve further discussion.

The first issue is that the simple modelling approach reduced path descriptions to a single real quantity, θ. This simplicity meant that controlling the path could be reduced to a simple linear feedback control law, with external input depending on the steering angle difference and lane centre deviation. Obviously, a path is actually a function from some parameter to position, i.e. a higher order concept, so more sophisticated representations can certainly be contemplated.

For example, the image processing unit may generate a moving representation of the road in front of the car —provided it is discernable (c.f. earlier remarks about muddy fields)— as a time dependent strip of varying width, length and direction. This could be communicated to the path generation unit via a few time dependent geometric parameters. The path generation unit could combine this with car velocity and direction information to derive the desired and predicted paths, as functions from time to position in the moving strip. According to the architectural diagram in Fig. 1, it would then be the responsibility of the LCC to synthesise the required steering angle from this information. In principle this is a problem in adaptive optimal control, and how it would be approached would depend crucially on the notion of optimality adopted (see, e.g. [15, 19, 20, 9, 28]).

We evaded the repercussions of all this potential complexity by assuming that the path generation unit already emitted a desired and safe steering angle and deviation from the middle of the road, and that it was sufficient for us to approach the required path via a relatively simple feedback law. However, if we were to take on board the more sophisticated modelling indicated in a more serious way, it would be appropriate to consider how the concepts involved would be handled in a system like Hybrid Event-B.

On the theoretical side there would be no problem, since higher order entities can be handled just as conveniently as basic variables can, within conventional continuous mathematics. On the practical side though, the relative dearth of analytic results for higher order entities, manifests itself in a smaller portfolio of cases that could be mechanised in an automated proving system. A further observation on the same point is that mechanical provers tend to perform much more poorly on higher order objects than they do on first order ones. So a completely abstract formulation of paths in the way we sketched it might need to be approached with caution in the context of mechanical proof.

The second issue concerns the nature of the *ASTC* that figured heavily around the *ACTIVE* state. We already argued that we could not guarantee that the *ASTC* would

be maintained for arbitrarily long periods while the LCC wished to remain in the *ACTIVE* state, i.e. that the *ASTC* would be an invariant for such periods. However, our remedy, of preempting the *LCC_Active* pliant transition whenever the *ASTC* failed at runtime, does in fact guarantee that '$mode = ACTIVE \Rightarrow ASTC$' is indeed an invariant. Why then did we not include such an invariant in our model?

To answer this, we note that *ASTC* contains variables from both the interface *LCC_PG_IF* (i.e. θ_T and d) and the machine *LCC_1* (i.e. θ). In which of these then, should we put this candidate invariant? Note that neither of them declares *all* of the variables mentioned. Assuming that an invariant should only use variables that are declared in the syntactic unit it resides in, we would have to amalgamate *LCC_PG_IF* into *LCC_1* to declare the suggested invariant.[2] In our case study, we managed to sidestep this problem, since we were able to demand *ASTC* on entry to *LCC_Active*, and preempted *LCC_Active* whenever *ASTC* failed, which together are tantamount to the stated invariant.

Nevertheless, the wider problem, of desirable invariants straddling the boundaries of otherwise sensible partitionings of large systems, remains. Perhaps the most promising suggestion for improvement regarding this issue comes from the 'shared event' approach of Butler [13] in which bound variables carrying communicated values can enjoy nontrivial properties without breaking the syntactic structuring of separate components. A generalisation of this to shared variables would be widely applicable across the B-Method.

Thus, we can safely say that the case study undertaken here has amply demonstrated the suitability of Hybrid Event-B for formally describing the requirements, specification and behaviour, of the kind of hybrid system that Event-B is increasingly being applied to these days, and moreover, it has raised a number of issues for future consideration. This is valuable experience which acts as a further spur to the development of full mechanical support for Hybrid Event-B within the framework of the Rodin Tool [4, 26, 27].

From the present vantage point, we can envisage without too much danger of error, on how such mechanical support could be organised. The KeYmaera tool [1, 24] provides sound inspiration. The KeYmaera tool started life by integrating an adapted version of the KeY proof tool [18] with Mathematica [21], thus allowing all the continuous reasoning to be delegated to the Mathematica tool. Since then, a number of other provers have been integrated into KeYmaera. A similar strategy could be followed for Rodin. The fact that Hybrid Event-B distinguishes cleanly between mode events and pliant events, makes it particularly evident that the POs of Hybrid Event-B [11] separate cleanly those in which discrete reasoning can be used from those in which continuous mathematics is needed. Just as for KeYmaera, the latter can be delegated to Mathematica in the first instance.

However, the fact that Mathematica is proprietary, and its reasoning is thus not open to scrutiny by users, means that complete reliance on its conclusions might

[2] It is sometimes suggested that existentially quantifying the 'other' variable(s) in an invariant of this kind can solve the problem. Unfortunately it does not, since asserting that some values *merely exist* (that satisfy some property) is quite different from asserting that the *actual current values* satisfy it. This is particularly dangerous when the invariant is a nontrivial safety property.

not be warranted in a context where the highest levels of dependability were demanded of a Hybrid Event-B development. In such cases, the requisite fragments of continuous mathematics could be developed in Rodin-specific rulesets, and the prover could be organised to use these in preference to Mathematica in situations where they were available. By this means, more and more of the problems tackled via Hybrid Event-B and Rodin could be covered by proofs that were fully open to inspection. The present authors intend to pursue the strategy just described for the mechanisation of Hybrid Event-B.

Acknowledgement: The authors would like to thank Sanaz Yeganefard for discussions, and for help with some of the figures. Michael Butler is partly funded by the FP7 ADVANCE Project (http://www.advance-ict.eu).

References

1. KeYmaera. http://symbolaris.com
2. Report: Cyber-Physical Systems (2008). http://iccps2012.cse.wustl.edu/_doc/CPS_Summit_Report.pdf
3. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010)
4. Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: An Open Toolset for Modelling and Reasoning in Event-B. STTT **12**, 447 (2010)
5. Alur, R., Dang, T., Esposito, J., Fierro, R., Hur, Y., Ivančić, F., Kumar, V., Lee, I., Mishra, P., Pappas, G., Sokolsky, O.: Hierarchical Hybrid Modeling of Embedded Systems. In: Henzinger, Kirsch (eds.) Proc. EMSOFT, *LNCS*, vol. 2211, pp. 14–31. Springer (2001)
6. Alur, R., Grosu, R., Hur, Y., Kumar, V., Lee, I.: Modular Specification of Hybrid Systems in CHARON. In: Lynch, Krogh (eds.) Proc. HSCC-00, *LNCS*, vol. 1790, pp. 6–19. Springer (2000)
7. Alur, R., Henzinger, T., Ho, P.: Automatic Symbolic Verification of Embedded Systems. IEEE TSE **22**, 181Ž013,201 (1996)
8. Antsaklis, P., Michel, A.: Linear Systems. Birkhauser (2006)
9. Astrom, K., Wittenmark, B.: Adaptive Control. Dover (2008)
10. Banach, R., Butler, M.: Cruise Control in Hybrid Event-B. In: Liu, Woodcock, Zhu (eds.) Proc. ICTAC-13, *LNCS*, vol. 8049, pp. 76–93. Springer (2013)
11. Banach, R., Butler, M., Qin, S., Verma, N., Zhu, H.: Core Hybrid Event-B: Adding Continuous Behaviour to Event-B (2012). Submitted.
12. Barolli, L., Takizawa, M., Hussain, F.: Special Issue on Emerging Trends in Cyber-Physical Systems. J. Amb. Intel. Hum. Comp. **2**, 249–250 (2011)
13. Butler, M.: Decomposition Strategies for Event-B. In: Leuschel, Wehrheim (ed.) Proc. IFM-09, vol. 5423, pp. 20–38. Springer, LNCS (2009)
14. Carloni, L., Passerone, R., Pinto, A., Sangiovanni-Vincentelli, A.: Languages and Tools for Hybrid Systems Design. Foundations and Trends in Electronic Design Automation **1**, 1–193 (2006)
15. Clarke, F., Ledyaev, Y., Stern, R., Wolenski, P.: Nonsmooth Analysis and Control Theory. Springer (1997)
16. DEPLOY: European Project DEPLOY IST-511599 http://www.deploy-project.eu/
17. Henzinger, T., Ho, P., Wong-Toi, H.: HyTech: A Model Checker for Hybrid Systems. IJSTTT **1**, 110–122 (1997)
18. KeY: http://www.key-project.org
19. Kirk, D.: Optimal Control Theory: An Introduction. Dover (2004)

20. Lewis, F., Vrabie, D., Syrmos, V.: Optimal Control. Wiley (2012)
21. Mathematica: http://www.wolfram.com
22. MATLAB and SIMULINK: http://www.mathworks.com
23. MODELICA: https://www.modelica.org/
24. Platzer, A.: Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer (2010)
25. Ratschan, S., She, Z.: Safety Verification of Hybrid Systems by Constraint Propagation Based Abstraction Refinement. In: Morari, Thiele (eds.) Proc. HSCC-05, *LNCS*, vol. 3414, pp. 573–589. Springer (2005)
26. RODIN: European Project RODIN (Rigorous Open Development for Complex Systems) IST-511599 http://rodin.cs.ncl.ac.uk/
27. RODIN Tool: http://www.event-b.org/ http://www.rodintools.org/ http://sourceforge.net/projects/rodin-b-sharp/
28. Sastry, S., Wittenmark, B.: Adaptive Control: Stability, Convergence and Robustness. Dover (2011)
29. Silva, B., Richeson, K., Krogh, B., Chutinan, A.: Modeling and Verifying Hybrid Dynamic Systems using CheckMate. In: Proc. 4th International Conference on Automation of Mixed Processes: Hybrid Dynamic Systems (ADPM 2000), pp. 323–328 (2000)
30. Sztipanovits, J.: Model Integration and Cyber Physical Systems: A Semantics Perspective. In: Butler, Schulte (eds.) Proc. FM-11. Springer, LNCS 6664, p.1, http://sites.lero.ie/download.aspx?f=Sztipanovits-Keynote.pdf (2011). Invited talk, FM 2011, Limerick, Ireland
31. Walter, W.: Ordinary Differential Equations. Springer (1998)
32. White, J., Clarke, S., Groba, C., Dougherty, B., Thompson, C., Schmidt, D.: R&D Challenges and Solutions for Mobile Cyber-Physical Applications and Supporting Internet Services. J. Internet Serv. Appl. **1**, 45–56 (2010)
33. Willems, J.: Open Dynamical Systems: Their Aims and their Origins. Ruberti Lecture, Rome (2007). http://homes.esat.kuleuven.be/~jwillems/Lectures/2007/Rubertilecture.pdf
34. Yeganefard, S., Butler, M.: Control Systems: Phenomena and Structuring Functional Requirement Documents. In: Proc. ICECCS-12, pp. 39–48. IEEE (2012)