DOI: 10.15439/2015F132
ACSIS, Vol. 5

Simulation and Formal Modelling of Yaw Control in a Drive-by-Wire Application

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

Pieter Van Schaik, Eric Verhulst
Altreonic,
Gemeentestraat 61/A,
Linden B3210, Belgium
Email: {pieter.vanschaik, eric.verhulst}@altreonic.com

***Abstract*—Cyberphysical systems, with their interdependence between physical behaviour and digital control, need insights from frequency domain control engineering, state space control engineering and discrete formal systems theory for their proper description. Neglecting any of these, results in descriptions that omit essential details. Hybrid Event-B is a formalism that enables all the relevant detail to be assimilated. A case study based on yaw control for the KURT e-vehicle is used as a testbed to explore the effective interaction between the various needed disciplines in exploring a specific design issue, the formalisation of yaw control discretization, using Hybrid Event-B.**

I. Introduction

TODAY, the low cost, small size, low energy consumption and wide availability of digital processors, together with the ready availability of a wide variety of stadardised control components, makes the embedding of computing components and digital control into what was previously purely analogue equipment, ubiquitous. This has now given rise to the burgeoning field of cyberphysical systems, in which computing systems are intimately connected to equipment that acts in the physical space. Increasingly, the impact of such systems is safety critical, and in such cases, the techniques by which the systems are developed demand scrutiny — at least in those spheres where it is recognised that safety and dependability (more precisely, their potential lack) merit certification processes that have to be successfully passed before systems can be deployed in the field. In reality, what is required to be certified often lags well behind what is attempted and shown to be feasible technically.

An essential element of many such certification processes is a verifiable audit trail of mathematical models of the system and of their relevant properties. Ideally, all models and properties should be verifiably consistent with one another, and demonstrably possess the properties needed for safe operation. In the case of cyberphysical systems this ideal is challenging, for the following reasons.

Traditionally, control design is done in the frequency domain [1], [2], [3]. This readily yields the quantities needed by the engineer, using a mixture of rigorous results and design heuristics. Although the rigorous results often do not hold with mathematical precision in reality (e.g. needed bandwidth assumptions), the degree of inexactitude is not harmful in practice. A major element of this approach is the use of simulation to judge the suitability of a design, using tools like Modelica [4].

Traditionally, computing systems, which proceed by discrete steps, are modelled and analysed within a discrete state space — there is no notion of frequency domain for an arbitrarily constructed discrete space. Since there is an enormous variety in the aspects of behaviour that can be modelled by the discrete steps of a computing system, correspondingly, depending on what the elements of the state space represent, we find an enormous variety of approaches to the formalisation of computing systems [5].

Following on from this observation, traditionally, formalisms for computing systems (e.g. the many surveyed in [5]) do not engage with continuous mathematics at all. To address this shortcoming in the context of cyberphysical systems, two different approaches are seen. In the first, the formalism does not engage with continuous mathematics, stays essentially discrete, and incorporates facts concerning continuous aspects of the modelled system as inputs or axioms. In the second, special purpose formalisms are designed to include continuous phenomena in suitable ways alongside the discrete ones.

More recently, a more rigorous approach to control has emerged within ‘mathematical control theory’ [6], [7], [8], that emphasises the state based approach to control. This perspective is able to relate more directly to the state based perspective of computing formalisms in a way that the frequency domain perspective (although equivalent to it via transform theory) would struggle to do. Moreover, the rigour of the proofs in mathematical control theory typically matches much better with the style of argument in computing formalisms, both being ultimately grounded in set theory. Then again, much of the useful engineering information that is evident and easily manipulated in the frequency domain, and is inferred smoothly from simulation, can become greatly obscured in the state based approach. Thus there is a conflict between the usual approaches to the various disciplines that contribute to the cyberphysical systems agenda.

In this paper we examine a case study that hosts an encounter (we hesitate to say collision) between the various approaches and issues mentioned. Although it is a simplified case study, it is not a toy, in that it is drawn directly from a genuine system, the KURT e-vehicle from Altreonic. We specifically look at yaw control and its stability in KURT. We embed the abridged development in the Hybrid Event-B (HEB) formalism [9], [10], and focus on the formal description and properties of the discretization step from a high level continuous design to a lower level time triggered discrete one.

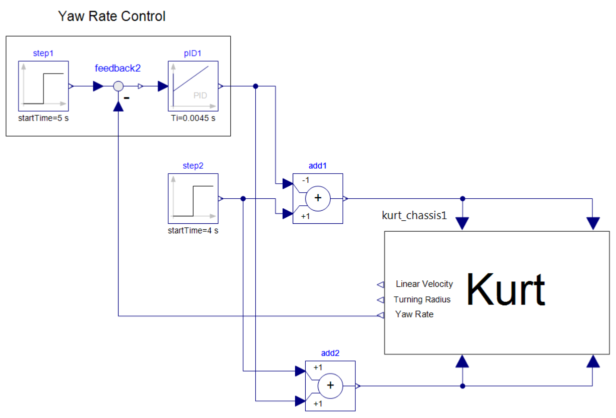

Fig. 1. KURT simulation: Modelica yaw rate control configuration.

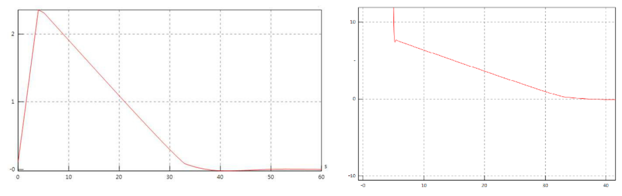

Fig. 2. KURT simulation: left, vehicle linear velocity (m/s) vs. simulation Time (s); right, vehicle turning radius (m) vs. simulation time (s).

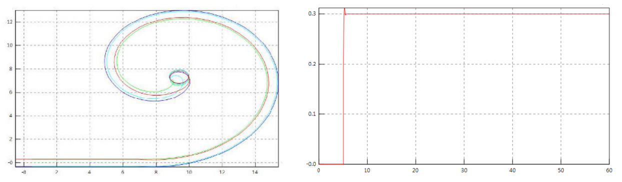

Fig. 3. KURT simulation: left, vehicle trajectory in Y vs. X axis (m); right, vehicle yaw rate response (rad/s) vs. simulation time (s).

From a rigorous point of view, discretization steps introduce copious amounts of low level detailed technical complexity. In order to keep the account within reasonable bounds, we do take some shortcuts in the development, commenting on the pros and cons as we go.

The rest of this paper is as follows. In Section II we overview KURT, and describe a Modelica simulation that was used to validate a number of design parameters for yaw control. In Section III we overview the HEB formalism, stressing the aspects that are most important for us. Section IV reformulates yaw control in HEB, and Section V examines the stability of the model in a state space based way. Section VI discusses the issues raised by going from a continuous to a discretized formulation in a formal manner. Section VII concludes.

II. Yaw Control in the KURT E-Vehicle

The KURT e-Vehicle is an innovative vehicle concept based on a modular, scalable and fault tolerant architecture. The propulsion system of KURT utilises four independently controlled in-wheel motors and employs a differential steering technique combined with a drive-by-wire architecture. In its simplest form such a steering technique entails steering the vehicle by creating a difference in linear velocity between the left and right side of the vehicle. Such approaches are often utilised in unmanned robotic platforms as well as heavy earth moving machinery. In the absence of a mechanical steering mechanism, such as articulated steering, these vehicles are often referred to as skid-steer vehicles. The advantages of utilising a 4-wheel drive differential steered concept includes minimising the mechanical complexity of the steering mechanism as well as increased manoeuvrability. However, reducing mechanical complexity demands a more intelligent propulsion control system which in turn will be deployed on an embedded target platform. The employed steering control strategy will therefore do well by minimising implementation complexity specifically with regards to aspects such as required processing power and number of sensors. To this end a control strategy has been devised for the KURT e-Vehicle whereby the effective yaw rate of the vehicle is utilised as the control parameter.

According to the kinematic relations [11] of a differential steered vehicle, yaw rate is related to linear velocity and instantaneous turning radius, and can be relatively easily measured with inexpensive MEMS based sensors. The added benefit of yaw rate control is related to safety, more specifically, with regards to maintaining stability of the combined vehicle and driver centre of gravity (COG) when executing turning manoeuvres. In a drive-by-wire, and specifically a steer-by-wire system, there is not necessarily provision for a feedback mechanism which serves to cause the driver to limit the turning radius when executing a turn in accordance with the linear velocity of the vehicle. Simply put, if a turn is taken too sharply at too high speed the vehicle can topple over. Therefore, by controlling the yaw rate of the vehicle when executing a turn, the effective turning radius can be controlled. This in turn permits preventing the centrifugal force component from getting large enough to cause the vehicle to overturn.

A basic propulsion and steering control strategy is as follows: the throttle command issued by the user is interpreted as a thrust request which is translated into a torque command and is applied equally to all four wheels; a steering command issued by the user is translated into a yaw rate request which serves as the set point to a closed loop PID controller. The output of the PID controller is added to the torque command of the wheels on one side and subtracted on the opposite side. The side to which it is added or subtracted depends on whether the steering request is to the left or to the right. When steering to the left the output of the PID controller will be subtracted from the torque command of the wheels on the left and added to the torque commands of the wheels on the right. When steering to the right the situation will be reversed.

In order to simulate the proposed control strategy, a dynamic model of a skid-steer vehicle was created in Modelica [4]. For the purposes of this investigation, the model does not include complex tyre and surface interactions, but rather models the wheels as point masses, on which the propulsion, rolling resistance and friction forces act.[1] The vehicle is represented with a simple H-shaped geometry with a single point mass located equidistant from the rear and front wheel

[1]Tyre and surface interactions result in friction forces (and are even necessary to control the vehicle properly).

representing the combined vehicle and payload mass. It is also the yaw rate of this mass that is measured and utilised in the steering control loop. The PID steering controller was designed by applying the Cohen-Coon tuning method [12], [13], [14]. A typical yaw control simulation setup is depicted in Fig. 1.

The kurt_chassis1 component from Fig. 1 resembles the dynamic model of the skid-steer vehicle. The model receives four inputs namely the thrust applied to the left rear and front wheel masses as well the thrust applied to the right rear and front wheel masses. The model provides three outputs of which the yaw rate is of primary concern. The step input source element step2 simulates a thrust request issued by the user. The steering PID controller is implemented by pID1 of which the output is subtracted from the output of step2 by add1 and added to the output of step2 by add2. The output of add1 is applied equally to the left wheel masses whereas the output of add2 is applied equally to the right wheel masses. The purpose of step1 is to simulate a steering request issued by the user in the form of a yaw rate request. The output of step1 serves as the set point to the closed loop controller with the yaw rate output of the kurt_chassis1 being the measured variable. The simulation setup therefore represents a steering request to turn left. The simulation sequence commences by issuing a constant thrust request ($thr = 15\,\mathrm{N}$) for a duration of 4 seconds during which no steering request is present ($yrr = 0\,\mathrm{rad/s}$). At $t = 4$ seconds the thrust command is removed ($thr = 0\,\mathrm{N}$) and at $t = 5$ seconds a constant steering request is issued ($yrr = 0.3\,\mathrm{rad/s}$). The simulation continues to run until $t = 40$ seconds.

The vehicle is therefore expected to accelerate from $t = 0$ to $t = 4$ seconds after which it will decelerate. From $t = 5$ seconds onwards the vehicle is expected to turn to the left. According to the kinematic relations the turning radius of the vehicle is expected to decrease proportionally to the linear velocity provided that the yaw rate is held constant. Figs. 2 and 3 depict the results obtained from the simulation run. The RHS of Fig. 3 shows the step response of the measured yaw rate. From the results it is seen that the controller is sufficiently capable of maintaining a constant yaw rate with acceptable overshoot (4%) and settling time (0.1 seconds). From Fig. 2 it is seen that the turning radius decreases in proportion to the linear velocity in accordance with the kinematic relations. Fig. 3 also shows the trajectory of the four wheel masses. The trajectory indicates the vehicle follows a spiral path as the linear velocity and the turning radius decreases. Correlating the trajectory with the turning radius it is observed that the rear wheels progressively digress from the trajectory of the front wheels as the turning radius decreases. From $t = 32$ seconds onwards the vehicle starts to rotate around its own centre of mass resulting in near zero turning radius.

III. An Outline of Hybrid Event-B

In this section we outline HEB, relating it to the more familiar Event-B [15]. The bulk of the material refers to a single machine. However, our models involve three machines: for the user, for KURT's behaviour, and for the control system, so we include what we need for multiple machines below.

A. Single Hybrid Event-B Machines

In Fig. 4 we see a schematic HEB machine. It starts with declarations of time and of a clock. Time is a first class citizen in that all variables are functions of time (which is read-only), explicitly or implicitly. Clocks are assumed to increase like time, but may be set during mode events. Variables are of two kinds. There are mode variables (like u) which take their values in discrete sets and change their values via discontinuous assignment in mode events. There are also pliant variables (such as x, y), declared in the PLIANT clause, which typically take their values in topologically dense sets (normally $\mathbb{R}$) and which are allowed to change continuously, such change being specified via pliant events.

Next are the invariants. These resemble invariants in discrete Event-B, in that the types of the variables are asserted to be the sets from which the variables' values *at any given moment of time* are drawn. More complex invariants are similarly predicates that are required to hold *at all moments of time* during a run.

Then, the events. The *INITIALISATION* has a guard that synchronises time with the start of any run, while all other variables are assigned their initial values as usual.

Mode events are analogues of events in discrete Event-B. They can assign all machine variables (except time). The schematic *MoEv* of Fig. 4, has parameters $i?, l, o!$, (input, local, and an output), and a guard *grd*. It also has the after-value assignment specified by the before-after predicate *BApred*, which can specify the after-values of all variables (except time, inputs and locals).

Pliant events are new to HEB. They specify the continuous evolution of the pliant variables over an interval of time. Fig. 4 has a schematic pliant event *PliEv*. There are two guards: *iv*, for specifying enabling conditions on the pliant variables, clocks, and time; and *grd*, for specifying enabling conditions on the mode variables.

The body of a pliant event contains three parameters $i?, l, o!$, (again, input, local, and output) which are functions of time, defined over the duration of the pliant event. The behaviour of the event is defined by the COMPLY and SOLVE clauses. The SOLVE clause contains direct assignments, e.g. of y and output $o!$ (to time dependent functions); and differential equations, e.g. specifying x via an ODE (with $\mathcal{D}$ as the time derivative).

The COMPLY clause can be used to express any additional constraints that are required to hold during the pliant event via the before-during-and-after predicate *BDApred*. Typically, constraints on the permitted ranges of the pliant variables, can be placed here. The COMPLY clause can also be used to specify properties at an abstract level, e.g. stating safety properties for the event without going into detail.

Briefly, the semantics of a HEB machine consists of a set of *system traces*, each of which is a collection of functions of time, expressing the value of each machine variable over the duration of a system run.

Time is modeled as an interval $\mathcal{T}$ of the reals. A run starts at some initial moment of time, t_0 say, and lasts either for a finite time, or indefinitely. The duration of the run, $\mathcal{T}$, breaks up into a succession of left-closed right-open subintervals: $\mathcal{T} = [t_0 \ldots t_1), [t_1 \ldots t_2), [t_2 \ldots t_3), \ldots$. Mode events (with their discontinuous updates) take place at the isolated times corresponding to the common endpoints of these subintervals t_i,

```
MACHINE HyEvBMch
TIME t
CLOCK clk
PLIANT x,y
VARIABLES u
INVARIANTS
  $x, y, u \in \mathbb{R}, \mathbb{R}, \mathbb{N}$
EVENTS
  INITIALISATION
    STATUS ordinary
    WHEN
      $t = 0$
    THEN
      $clk, x, y, u := 1, x_0, y_0, u_0$
    END
...  ...
```

```
...  ...
  MoEv
    STATUS ordinary
    ANY $i?, l, o!$
    WHERE
      $grd(x, y, u, i?, l, t, clk)$
    THEN
      $x, y, u, clk, o! :|$
        $BApred(x, y, u, i?, l, o!,$
          $t, clk, x', y', u', clk')$
    END
...  ...
```

```
...  ...
  PliEv
    STATUS pliant
    INIT $iv(x, y, t, clk)$
    WHERE $grd(u)$
    ANY $i?, l, o!$
    COMPLY
      $BDApred(x, y, u,$
        $i?, l, o!, t, clk)$
    SOLVE
      $\mathcal{D}\, x =$
        $\phi(x, y, u, i?, l, o!, t, clk)$
      $y, o! :=$
        $E(x, u, i?, l, t, clk)$
    END
END
```

Fig. 4. A schematic Hybrid Event-B machine.

and in between, the mode variables are constant, and the pliant events stipulate continuous change in the pliant variables.

We insist that on every subinterval $[t_i \ldots t_{i+1})$ the behaviour is governed by a well posed initial value problem $\mathcal{D}\, xs = \phi(xs\ldots)$ (where xs is a relevant tuple of pliant variables). Within this interval, we seek the earliest time t_{i+1} at which a mode event becomes enabled, and this time becomes the preemption point beyond which the solution to the ODE system is abandoned, and the next solution is sought after the completion of the mode event.

In this manner, assuming that the *INITIALISATION* event has achieved a suitable initial assignment to variables, a system run is *well formed*, and thus belongs to the semantics of the machine, provided that at runtime:

(1) Every enabled mode event is feasible, i.e. has an after-state, and on its completion enables a pliant event (but does not enable any mode event).[2]

(2) Every enabled pliant event is feasible, i.e. has a time-indexed family of after-states, and EITHER:
   (i) During the run of the pliant event a mode event becomes enabled. It preempts the pliant event, defining its end. ORELSE
   (ii) During the run of the pliant event it becomes infeasible: finite termination. ORELSE
   (iii) The pliant event continues indefinitely: nontermination.

Thus, in a well formed run mode events alternate with pliant events. The last event (if there is one) is a pliant event (whose duration may be finite or infinite). In reality, there are several semantic issues that we have glossed over in the framework just sketched. We refer to [9] for a more detailed presentation (and to [10] for the extension to multiple machines). The presentation just given is quite close to the modern formulation of hybrid systems. See e.g. [16], [17] — or [18] to get a perspective stretching further back.

If, from Fig. 4, we erase time, clocks, pliant variables and pliant events, we arrive at a skeleton (conventional) Event-B machine. This simple erasure process illustrates (in reverse) the way that HEB has been designed as a clean extension of the original Event-B framework. The only difference of note is that, now —at least according to the (conventional) way that Event-B is interpreted in the physical world— (the mode) events (left behind by the erasure) execute *lazily*, i.e. *not* at the instant they become enabled (which is, of course, the moment of execution of the previous event).

B. Multiple Hybrid Event-B Machines

The principal objective in modelling complex systems in the B-Method is to start with small simple descriptions and to refine to richer, more detailed ones. This means that, at the highest levels of abstraction, the modelling must **abstract away from concurrency**. By contrast, at lower levels of abstraction, the events describing detailed individual behaviours of components become visible. In a purely discrete event framework, like conventional Event-B, there can be some leeway in deciding whether to hold all these low level events in a single machine or in multiple machines — because all events execute instantaneously, isolated from one another in time (in the usual interpretation).

In HEB the issue is more pressing. Because of the inclusion of continuous behaviour, *all* components are always executing *some* event. Thus an integrated representation risks hitting the combinatorial explosion of needing to represent each possible combination of concurrent activities within a separate event, and so there is a much stronger incentive to put each (relatively) independent component into its own machine, synchronised appropriately. Put another way, there is a very strong incentive to **not abstract away from concurrency**.

The same impulse is reinforced when we wish to construct systems out of components, e.g. a plant and a controller. There, it is also convenient to conceive the pieces separately and combine them appropriatelty. The key concept in achieving this is the INTERFACE. This is a syntactic contruct (adapted from the idea in [19]) that includes the declarations of a set of variables, the invariants that involve them, and also their initialisations. A community of machines may have access to the variables declared in an interface if each machine CONNECTS to the interface. All events in the machines must

[2]If a mode event has an input, the semantics assumes that its value only arrives at a time strictly later than the previous mode event, ensuring part of (2) automatically.

```
PROJECT Kurt_Prj
  INTERACES
    YawCtrl_IF
  MACHINES
    KurtUser_Mch
    Kurt_Mch
    YawCtrl_Mch
  END
```

```
INTERFACE YawCtrl_IF
  SEES Kurt_Ctx
  TIME t
  PLIANT
    yrr, yrm, stc,
    yreP, yreI, yreD,
    thr, tal, tar
  INVARIANTS
    $yrr, yrm, stc \in \mathbb{R}, \mathbb{R}, \mathbb{R}$
    $yreD, yreP, yreI \in \mathbb{R}, \mathbb{R}, \mathbb{R}$
    $thr, tal, tar \in \mathbb{R}, \mathbb{R}, \mathbb{R}$
  INITIALISATION
    WHEN
      $t = 0$
    THEN
      $yrr, yrm, stc := 0, 0, 0$
      $yreP, yreI, yreD := 0, 0, 0$
      $thr, tal, tar := 0, 0, 0$
    END
  END
```

```
CONTEXT Kurt_Ctx
  ... ...
  AXIOMS
  ... ...
  END
```

```
MACHINE KurtUser_Mch
  CONNECTS YawCtrl_IF
  EVENTS
    SteerKurt
      STATUS pliant
      BEGIN
        $thr(t) := \Theta(4-t)$
        $yrr(t) := \Theta(t-5)$
      END
  END
```

```
MACHINE Kurt_Mch
  CONNECTS YawCtrl_IF
  EVENTS
    KurtBehaves
      STATUS pliant
      SOLVE
        $\mathcal{D}\, yrm(t) := C_K stc(t)$
      END
  END
```

```
MACHINE YawCtrl_Mch
  CONNECTS YawCtrl_IF
  EVENTS
    YawControl
      STATUS pliant
      SOLVE
        $yreP(t) := yrr(t) - yrm(t)$
        $yreD(t) := \mathcal{D}\, yreP(t)$
        $\mathcal{D}\, yreI(t) := yreP(t)$
        $stc(t) :=$
          $K_P[yreP(t) + yreI(t)/T_I + T_D\, yreD(t)]$
        $tal(t) := thr(t) - stc(t)$
        $tar(t) := thr(t) + stc(t)$
      END
  END
```

Fig. 5. A Hybrid Event-B system for yaw control.

preserve all of the invariants in the interface, of course. An important point is that *all* invariants involving the interface's variables must be in the interface.

Multi-machine HEB systems need more than what we have just described, namely (at least) synchronisation and instantiation mechanisms. These, and other issues, are discussed in [10]. What we have mentioned will suffice for this paper.

IV. A Hybrid Event-B Model of Yaw Control

In this section we take the model discussed in Section II and re-express it as a Hybrid Event-B PROJECT. The project itself appears in Fig. 5, where its overall structure is defined in the PROJECT *Kurt_Prj* file. This indicates the pieces that the system is constructed from. These consist of the INTERFACE *YawCtrl_IF* and the MACHINEs *KurtUser_Mch*, *Kurt_Mch* and *YawCtrl_Mch*.

The interface SEES the CONTEXT *Kurt_Ctx* which contains the definitions of all the constants and static mathematics that the project will need, and more importantly, it is also the home of any AXIOMS (concerning these static elements) that we may rely on for verification. The interface then names the (pliant) variables shared by the machines that connect to it, lists their invariants, and defines their intialisations. Table 1 lists the variables, and describes how they relate to the elements of the KURT simulation model in Fig. 1.

The three machines *KurtUser_Mch*, *Kurt_Mch* and *YawCtrl_Mch* are formal definitions of the three actors in the dynamics.

The *KurtUser_Mch* machine describes the behaviour of the user who drives KURT. The machine CONNECTS to the *YawCtrl_IF* interface, to access needed variables, and it has a single pliant event *SteerKurt*. This applies a constant thrust from time 0 to time 4 $thr(t) := \Theta(4-t)$, and a constant yaw request from time 5 onwards $yrr(t) := \Theta(t-5)$, where Θ is the Heaviside step function. This is consistent with the description in Section II.

Machine *Kurt_Mch* describes the intrinsic behaviour of the KURT e-vehicle. It also CONNECTS to *YawCtrl_IF*. In this simple model it is assumed that KURT will emit a measured yaw rate *yrm* whose derivative is proportional to the difference of the thrusts applied to left and right wheel sets $tar(t) - tal(t)$, and which is thus (via a positive constant C_K) proportional to the differential thrust $stc(t)$ (see Fig. 1):

$$\frac{d}{dt}\, yrm(t) \;=\; C_K\, stc(t) \tag{3}$$

Machine *Kurt_Mch* expresses this in HEB notation.

Machine *YawCtrl_Mch* describes the controller that turns the user's steering commands into thrust commands to KURT's wheels. Of course it CONNECTS to *YawCtrl_IF*. At its heart is the PID controller in Fig. 1 which calculates the differential steering thrust command $stc(t)$ from the value, integral and derivative of the yaw rate error $yer(t)$:

$$stc(t) \;=\; K_P\left[yre(t) + \frac{1}{T_I}\int_0^t yre(s)\, ds + T_D \frac{d}{dt}\, yre(t)\right] \tag{4}$$

The formalism of HEB does not permit us to write this directly since (aside from implicit constraints in the COMPLY clause), it allows direct assignment and differential equations only, in the SOLVE clause. The formulation in the *YawCtrl_Mch* machine unwinds (4) into an acceptable form. Thus, separate variables are introduced for the proportional, integral and derivative of $yre(t)$: $yreP(t), yreI(t), yreD(t)$ (variable $yre(t)$ itself is not an element of the *Kurt* project). On this basis, equation (4) turns into the following lines of the SOLVE clause of the *YawControl* pliant event:

$$yreP(t) \ := \ yrr(t) - yrm(t) \tag{5}$$
$$yreD(t) \ := \ \mathcal{D}\,yreP(t) \tag{6}$$
$$\mathcal{D}\,yreI(t) \ := \ yreP(t) \tag{7}$$
$$stc(t) \ := \ K_P[yreP(t) + yreI(t)/T_I + T_D\,yreD(t)] \tag{8}$$

The remaining assignments in the SOLVE clause of the *YawControl* event, quite faithfully mirror the relevant functions and connections of the yaw control model in Fig. 1, when the interpretation is mediated via the information in Table 1.

V. Formal Properties of Yaw Control

Some properties can be easily checked from the text of Fig. 5. For instance, each of the variables of the *YawCtrl_IF* interface appears exactly once in the left hand side of any of the assignments or ODEs in any of the pliant events in the project. Since all these pliant events run concurrently, this property is a prerequisite for consistency.

The next obvious thing is the observation that all of the assignments and equations of the *Kurt* project are linear. This means that an analytic solution to the system's behaviour is within reach, which we examine now.

A. Stability Analysis of the Simulation

Given that there are step functions in the system inputs thr and yrr at $t = 4$ and $t = 5$, the behaviour splits naturally into three intervals: $[0\ldots4), [4\ldots5), [5\ldots\infty)$.

During $[0\ldots4)$ the vehicle accelerates from 0: thus $thr = 15 = tal = tar$, and all other variables remain at 0. During $[4\ldots5)$, the thrust is switched off $thr = 0 = tal = tar$. So all variable values are 0. (In the simulation of Section II the vehicle begins to slow, although this depends on frictional forces not included in our formal model.)

Table 1: Variables Used in the Yaw Control Models

Variable	Meaning
yrr	Yaw Rate Request (output of step1)
yrm	Yaw Rate Measured (output of Kurt)
yre	Yaw Rate Error, i.e. $yre = yrr - yrm$ (output of feedback2)
yreD	Time Derivative of Yaw Rate Error (derivative of output of feedback2)
yreP	Proportional Steering Yaw Rate Error, i.e. $yreP = yre$ (output of feedback2)
yreI	Time Integral of Yaw Rate Error (integral of output of feedback2)
stc	(Differential) Steering Thrust Command (output of pID1)
thr	Thrust Request (output of step2)
tal	Thrust Applied Left (output of add1)
tar	Thrust Applied Right (output of add2)

Turning starts at $t = 5$, and we must solve the system of equations in the *Kurt* project. The *KurtBehaves* event in the *KurtMch* machine implies that $yrm(t)$ is the time integral of $C_K stc(t)$ up to a constant of integration L_P. This can be substituted into the right hand side of (5) which then yields $yreP(t)$. Differentiating this, in turn yields $yreD(t)$ via (6). Integrating it instead, yields $yreI(t)$ via (7), up to another constant of integration L_I. Substituting these relationships into (8) yields the integral equation:

$$\begin{aligned} stc(t) \ &= \ K_P(0.3 - L_P) - C_K K_P \int_5^t stc(s)\,ds \\ &+ \frac{K_P}{T_I}\left[(0.3 - L_P)(t-5) - L_I - C_K \int_5^t\!\!\int_5^s stc(u)\,du\,ds\right] \\ &- C_K K_P T_D\, stc(t) \end{aligned} \tag{9}$$

Differentiating this twice yields the homogeneous ODE:

$$\left(T_D + \frac{1}{C_K K_P}\right)\frac{d^2}{dt^2}stc(t) + \frac{d}{dt}stc(t) + \frac{1}{T_I}stc(t) \ = \ 0 \tag{10}$$

The only solutions of (10) are exponential. Putting in the ansatz $stc(t) = Re^{\lambda t}$ and integrating twice yields candidates for the integral terms in the RHS of (9). Since (9) must be an identity, equating coefficients of $e^{\lambda t}$ and of the linear terms allows L_P, L_I and R to be determined from initial conditions. And with $stc(t)$ determined, we can easily calculate the behaviour of all the other system variables if we wish.

For mechanical stability, we need the real part of either value of λ to be negative. This yields two constraints on the family of constants in the *Kurt* system, each being of the form $expr > 0$. However, up to positive constant factors and an additional factor of T_I, one $expr$ is the reciprocal of the other. Therefore, the two cannot be consistent unless $T_I > 0$, whence we get:

$$T_I > 0 \quad \text{and} \quad T_D + \frac{1}{C_K K_P} > 0 \tag{11}$$

We can add these as AXIOMS to the context *Kurt_Cxt*:

AXIOMS
$T_I > 0$
$T_D + 1/(C_K K_P) > 0$

With axioms like these included in the project, new invariants become provable. Specifically, because $yreP(t)$ in $[5\ldots\infty)$ is bounded by a negative exponential displaced by a constant, its maximum is finite, so that we can add:

INVARIANTS
$yreP(t) \leq yreP_{MAX}$

to the interface *YawCtrl_IF*, where $yreP_{MAX}$ can be calculated explicitly.

B. More General Stability Analysis

The above analysis accurately reflected —though from a formal vantage point— the kind of evaluation that can be achieved by a simulation based approach, such as we had in Section II. In this section, we extend the formal analysis to the case of a more arbitrary yaw rate request input $yrr(t)$, provided it stays within specified bounds. We illustrate thereby the greater reach of a more symbolically based approach, in cases where the calculational challenges remain tractable.

With a relatively arbitrary $yrr(t)$, we can redo the derivation of the previous section. We arrive at an analogue of (10) in which the LHS is as before and the RHS is modified:

$$\begin{aligned} \ldots &= \frac{1}{C_K}\left(T_D\frac{d^3}{dt^3}yrr(t)+\frac{d^2}{dt^2}yrr(t)+\frac{1}{T_I}\frac{d}{dt}yrr(t)\right)\\ &\equiv inh(t) \end{aligned} \tag{12}$$

We see that the inhomogeneous term $inh(t)$ depends solely on the derivatives of $yrr(t)$.

Introducing the vector $\mathbf{stc}(t) = [stcP(t) \quad stcD(t)]^{\mathrm{T}}$ where $stcP(t) \equiv stc(t)$ and $stcD(t)$ is the time derivative of $stcP(t)$, we can write the second order ODE (12) as a first order system:

$$\frac{d}{dt}\mathbf{stc}(t) = \mathbf{A}\,\mathbf{stc}(t) + \mathbf{b}(t) \tag{13}$$

where:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -H/T_I & -H \end{bmatrix} \quad \text{and} \quad \mathbf{b}(t) = \begin{bmatrix} 0 \\ H\,inh(t) \end{bmatrix} \tag{14}$$

and $H = C_K K_P/(1 + C_K K_P T_D)$, (the latter being the reciprocal of the constant appearing in (11)).

The form of (13) is standard (see, e.g. [20], [21], [22] and many other places), so the system can be integrated by applying a routine procedure. For $t \geq 5$ we have:

$$\mathbf{stc}(t) = e^{\mathbf{A}(t-5)}\mathbf{stc}(5) + \int_5^t e^{\mathbf{A}(t-s)}\mathbf{b}(s)\,ds \tag{15}$$

Since $\mathbf{b}(t)$ consists solely of derivatives of $yrr(t)$, we can integrate by parts repeatedly. To do so we introduce the notation $\mathbf{yrr}(t) = [0 \quad yrr(t)]^{\mathrm{T}}$, and we observe that:

$$\begin{aligned} &\int_5^t e^{\mathbf{A}(t-s)}\frac{d^k}{ds^k}\mathbf{yrr}(s)\,ds = \\ &\qquad \left[e^{\mathbf{A}(t-s)}\textstyle\sum_{j=0}^{k-1}\mathbf{A}^{k-j-1}\dfrac{d^j}{ds^j}\mathbf{yrr}(s)\right]_5^t \\ &\qquad + \mathbf{A}^k\int_5^t e^{\mathbf{A}(t-s)}\mathbf{yrr}(s)\,ds \end{aligned} \tag{16}$$

So as not to have to deal with a large collection of boundary terms coming from (16), we now hypothesise a turning episode in which $yrr(t)$ starts at zero (for $t = 5$), smoothly increases and then smoothly decreases back to zero (for $t > 9$ say). Dropping the boundary terms, we get, in the $t > 9$ region:

$$\begin{aligned} \mathbf{stc}(t) &= e^{\mathbf{A}(t-5)}\mathbf{stc}(5) + \\ &\frac{1}{C_K}\left(\frac{1}{T_I}\mathbf{A} + \mathbf{A}^2 + T_D\mathbf{A}^3\right)\int_5^t e^{\mathbf{A}(t-s)}\mathbf{yrr}(s)\,ds \end{aligned} \tag{17}$$

A result like (17) allows us to estimate in a symbolic manner the steering thrust command required for turning episodes corresponding to $yrr(t)$'s that behave in ways characterised by some generic pattern. For example we may be able to confirm that for the class of turning episodes considered, the magnitude of the steering command will not breach the physical boundaries engineered into the system.[3]

If this strategy is pursued, then the properties assumed for $yrr(t)$ can be introduced axiomatically in the interface *YawCtrl_IF*. Technically, constants would be introduced in *YawCtrl_IF*, e.g. a constant *YRR* naming a function of time, which would be endowed with the properties required to be assumed for $yrr(t)$, expressed via axioms. Then the behaviour of $yrr(t)$ would be set equal to *YRR* in machine *KurtUser_Mch*.

The properties concerning $yrr(t)$ derivable from this basis could be dealt with in various ways. For persistent properties, the most natural approach would be to recast them as invariants of the system. Properties not of this kind cold be expressed as THEOREMS in the syntax. Both kinds would then need to be proved.

C. On Mechanical Verification

The previous two sections gave examples of what could be addressed within a formal development framework capable of treating continuous behaviour as first class citizen. But writing a desirable property is one thing, and mechanically discharging a proof of it is another. While proper mechanical support for HEB is, as yet, an aspiration, achieving the power to do the kind of mathematics indicated in a reasonable time would require the import of the capabilities of existing tools like *Mathematica* [23]. Such an approach is entirely practical, and would provide a good level of additional assurance, beyond what can be achieved by explorations of system behaviour via simulation.

For applications requiring an even higher level of assurance, the user would have to program the rules and tactics for the relevant portion of mathematics directly, so that the details of the derivation could be exposed to scrutiny, in contrast to tools like *Mathematica*, where the internal reasoning algorithms are commercial secrets. The capability to approach the verification task in both ways is part of the planned tool support for HEB.

VI. Discretizing Hybrid Event-B Yaw Control

A major issue in turning a conceptual design into a reality in today's engineering environment, is going from the original continuous control model to a discretized control model. This is because, with today's components, analogue control is prohibitively expensive when compared to its discrete counterpart. (There are, of course, many other reasons for preferring discrete control which are well known, such as the flexibility of software, and the lack of drift in digital components.)

In some approaches, the design is initiated directly in the discrete sphere, bypassing the continuous world altogether. However, that forces the problem of deciding the sampling frequency, to be confronted immediately. The advantage of starting in the continuous world is that this issue is postponed in favour of engagement with the primary design challenges, which are most clearly viewed in the continuous world. This is what we do here, starting with the continuous model, and then contemplating the discretized version.

[3]A more realistic simulation of KURT than shown in Section II includes limiters to do just that.

```
PROJECT $KurtD\_Prj$
**REFINES –??–** $Kurt\_Prj$
INTERACES
  $YawCtrlD\_IF$
MACHINES
  $KurtUserD\_Mch$
  $KurtD\_Mch$
  $YawCtrlD\_Mch$
END
```

```
INTERFACE $YawCtrlD\_IF$
REFINES –??– $YawCtrl\_IF$
SEES $KurtD\_Ctx$
TIME $t$
PLIANT
  $yrr_D, yrm_D,$
  $stc_D, stc_D^{pr},$
  $yreP_D, yreP_D^{pr},$
  $yreI_D, yreD_D,$
  $thr_D, tal_D, tar_D$
INVARIANTS
  $yrr_D, yrm_D \in \mathbb{R}, \mathbb{R}$
  $stc_D, stc_D^{pr} \in \mathbb{R}, \mathbb{R}$
  $yreP_D, yreP_D^{pr} \in \mathbb{R}, \mathbb{R}$
  $yreI_D, yreD_D \in \mathbb{R}, \mathbb{R}$
  $thr_D, tal_D, tar_D \in \mathbb{R}, \mathbb{R}, \mathbb{R}$
  $thr_D = thr$
  $yrr_D = yrr$
  $|yrm_D - yrm| < B_{yrm}$
  $|stc_D - stc| < B_{stc}$
  $|stc_D^{pr} - stc| < B_{stc}$
  $|yreP_D - yreP| < B_{yreP}$
  $|yreP_D^{pr} - yreP| < B_{yreP}$
  $|yreI_D - yreI| < B_{yreI}$
  $|yreD_D - yreD| < B_{yreD}$
  $|tal_D - tal| < B_{tal}$
  $|tar_D - tar| < B_{tar}$
...   ...
```

```
...   ...
  INITIALISATION
    WHEN
      $t = 0$
    THEN
      $yrr_D, yrm_D$ := $0, 0$
      $stc_D, stc_D^{pr}$ := $0, 0$
      $yreP_D, yreP_D^{pr}$ := $0, 0$
      $yreI_D, yreD_D$ := $0, 0$
      $thr_D, tal_D, tar_D$ := $0, 0, 0$
    END
  END
```

```
CONTEXT $KurtD\_Ctx$
EXTENDS $Kurt\_Ctx$
  ...   ...
AXIOMS
  $NT = 1$
  ...   ...
END
```

```
MACHINE $KurtUserD\_Mch$
REFINES $KurtUser\_Mch$
CONNECTS $YawCtrlD\_IF$
EVENTS
  $SteerKurt$
    REFINES $SteerKurt$
    STATUS pliant
    BEGIN
      $thr_D(t)$ := $\Theta(4 - t)$
      $yrr_D(t)$ := $\Theta(t - 5)$
    END
END
```

```
MACHINE $KurtD\_Mch$
REFINES –??– $Kurt\_Mch$
CONNECTS $YawCtrlD\_IF$
EVENTS
  $KurtBehavesPli$
    REFINES $KurtBehaves$
    STATUS pliant
    COMPLY skip
    END
  $KurtBehavesMo$
    STATUS ordinary
    WHEN $(\exists n \in \mathbb{N} \bullet t = nT)$
    THEN
      $yrm_D$ := $yrm_D + C_K T stc_D$
    END
END
```

```
MACHINE $YawCtrlD\_Mch$
REFINES –??– $YawCtrl\_Mch$
CONNECTS $YawCtrlD\_IF$
EVENTS
  $YawControlPli$
    REFINES $YawControl$
    STATUS pliant
    COMPLY skip
    END
  $YawControlMo$
    STATUS ordinary
    WHEN $(\exists n \in \mathbb{N} \bullet t = nT)$
    THEN
      $yreP_D$ := $yrr_D - yrm_D$
      $yreP_D^{pr}$ := $yreP_D$
      $yreI_D$ := $yreI_D + T yreP_D$
      $yreD_D$ := $(yreP_D - yreP_D^{pr})/T$
      $stc_D$ := “$K_P[yreP_D + yreI_D/T_I + T_D yreD_D]$”
      $stc_D^{pr}$ := $stc_D$
      $tal_D$ := $thr_D - stc_D$
      $tar_D$ := $thr_D + stc_D$
    END
END
```

Fig. 6. A discretized Hybrid Event-B system for yaw control.

A. *Continuous and Discretized Systems*

From a formal development standpoint, the most desirable relationship between a system model and its more idealised predecessor, is a refinement. Typically, a refinement enriches a more idealised model with detail taking it 'closer to implementation'. The enriched model is proved consistent with its predecessor (normally, via a formal simulation relation). Done properly, a refinement has the potential to preserve valuable properties established earlier, in the new model. Unfortunately, in the context of the discretization issue, this strategy, applied naively, fails. The reasons are as follows.

A continuous description of a system contains an 'infinite' amount of information: i.e. the values of all system variables over a continuum of times. Any implementable sampling method will unavoidably 'forget' all but a tiny fraction of this information, i.e. all but the sampled values themselves. If the system response to the environment depends on the information it has about the system's behaviour, it is more or less inevitable that, in principle, the quality of a sampled system's response will be inferior compared with the continuous case. Thus the discretization process is not an enrichment of the continuous model but an impoverishment, and refinement, as a technique, struggles to cope with it, since the impoverishment degrades the information available for the consistency proof rather than enhancing it.

Still, the news is not all bad. Typically, the interaction between the system and the environment/plant is two way (closed loop). If, overall, both the continuous and discrete versions of the combined system are stable (with suitable choices of parameters etc.), then a reaction in the discretized system that is in some way undesirably increased compared to what it would be in the continuous system under similar circumstances, can be compensated for by the environment of the discretized system, which can increase suitably its input to the system to steer overall behaviour towards the desired regime. Doing this successfully depends on a number of things: good understanding of both system and environment; the deviations spoken of being moderate in magnitude; the overall system (in both the continuous and discrete versions) being stable; suitable choices of parameters being made.[4] See [16]

[4]Suitable parameter choice is heavily dependent on insight from the frequency domain. We return to this point below.

for a relevant technical discussion. However, if the interaction between the system and the environment is one way (open loop), it is much easier to see less acceptable devations.

B. The Discretized Model

In Fig. 6 there is a discretized version of the previous continuous yaw control model. Each syntactic construct is replaced by its discretized counterpart; e.g. *Kurt_Prj* is replaced by *KurtD_Prj* which **REFINES –??–** it. The question marks qualifying the REFINES claim refer to a certain level of ignorance concerning the precision of the relationship between the continuous and discretized versions that we must endure, and that affects many components of the two models. We discuss this point in detail in Section VI-D.

In this exercise, for simplicity, we keep all the model constants (such as C_K, K_P etc.) the same.[5] In addition, there is a further constant T, which represents the sampling period. For simplicity, T is axiomatized to be $1/N$'th of a unit of time, so that the external stimuli to the system can remain the same as in the continuous model (and both, moreover, are open loop).

Variables var_D are the discretized counterparts of their earlier predecessors var, sampled and updated every T time units. Now, the semantics of Hybrid Event-B imposes a specific interpretation on the assignments that occur in mode events, e.g. $var_D := expr(var_D)$. When such an assignment is executed at a time kT say, the LHS of the assignment denotes the new value $var_D(kT)$, which we write as $var_{D,k}$. However, the RHS is evaluated using the limiting value of var_D just before kT. If we assume that var_D does not change during any sampling interval, then the RHS is in fact $expr(var_{D,k-1})$, so the assignment implements the difference equation $var_{D,k} = expr(var_{D,k-1})$. We return to this point below.

In order to implement simple approximations to derivatives and integrals (done via backward differences and accumulated sums, respectively), preceding values of some variables need to be recorded: var_D^{pr}. (As with the model constants, the literature contains many approaches that tackle these issues in more sophisticated ways; see e.g. [24], [25], [26].)

We discuss the machines, one by one. The simplest is *KurtUserD_Mch*. This genuinely REFINES the earlier *KurtUser_Mch* machine in a manner which is easy to see. Namely, the original and discretized variables thr and yrr vs. thr_D and yrr_D have identical behaviours in the sole (pliant) event of the two machines *SteerKurt*. This is formalised via the equalities $thr_D = thr$ and $yrr_D = yrr$ in the invariants of the interface.

Next we have *KurtD_Mch*. This has both a pliant event *KurtBehavesPli* and a mode event *KurtBehavesMo*. The pliant event (continuously) skips. This models the zero order hold that characterises a simple sampling scheme. The pliant event REFINES –??– the *KurtBehaves* event of *Kurt_Mch*. The mode event models the periodic updates to yrm_D at the sampling times, obtained by replacing the differential equation of the *KurtBehaves* event with a discretized approximation of the corresponding integral equation via $yrm_D := yrm_D + C_K T stc_D$. This is to be interpreted as discussed above, which means that the assignment represents the difference equation $yrm_{D,k} = yrm_{D,k-1} + C_K T stc_{D,k-1}$.

Then we have *YawCtrlD_Mch*. The conventions already described hold here too. Thus there is a pliant event *YawControlPli* that skips while it REFINES –??– the *YawControl* event of *YawCtrl_Mch*, and a mode event *YawControlMo* that models the periodic updates to the discretized counterparts of all the variables modified by *YawControl*. At this point a subtlety needs to be pointed out.

In a pliant event, there is no difference between the (parallel) direct assignments $x,y := y,z$ and $x,y := z,z$ because of the equality semantics of direct (instantaneous) assignment. However, when the two assignments are naively discretized, they turn into $x_D, y_D := y_D, z_D$ and $x_D, y_D := z_D, z_D$ respectively, which are to be interpreted as we discussed above. The first of these corresponds to the difference equation $x_{D,k}, y_{D,k} = z_{D,k-2}, z_{D,k-1}$, because the LHS and RHS of such assignments refer to values one sampling period apart, as noted above. Thus a chain of n dependent equalities in a pliant event —which in the pliant event relate values at the same time point— can generate an n'th order difference equation upon discretization. This can have detrimental effects on the quality of the approximation and on its stability due to the use of older and older values — as is discussed extensively within numerical analysis, e.g. [27], [28].

In order to minimise the impact of this, we can back substitute to make use of values that are as fresh as possible.[6] Note that every different choice of scheme for doing the back substitutions results in a discretization scheme that is correspondingly different, amounting to a different design decision regarding what discretization means.

We can go further, designing the difference equations that we wish to use for the discretization *a priori*, and independently of the 'obvious' discretizations of the continuous model, and then work back to derive the discrete assignments that would implement them.

In the context of these remarks, the main impact on the *YawCtrlD_Mch* machine is to alter the detailed expressions that occur on the RHS of the assignments of the mode event. The assignment that is of most interest is the assignment for stc_D, where the feedback from the control strategy is most felt. Its RHS is enclosed in heavy quotes to allude to this.

C. Discretized Stability Analysis

Let $k \in \mathbb{N}$ index the number of $1/N$'ths of a time unit elapsed since $t = 5$. We address the discretization of stc_D in more detail. In the light of all the possibilities just discussed, we proceed as follows.

Looking at the assignments for yrm_D and $yreP_D$ that appear in Fig. 6 and dropping the inhomogeneous terms, it is not difficult to derive the difference equation $yreP_{k+1} = -C_K T \sum_{r=0}^{k} stc_{D,r}$. Deriving the analogous expression for $yreI_D$ involves a double summation. Dealing with these directly in the assignment for stc_D is certainly inconvenient. However, if we take second differences of the stc_D assignment, the summations

[5]In many discretization approaches, model constants are adjusted, in order to better approximate the continuous model.

[6]This amounts to using the $x_D, y_D := z_D, z_D$ form in the earlier example.

cancel, and we obtain:

$$\begin{aligned} stc_{D,k+3} - 2stc_{D,k+2} + stc_{D,k+1} &= \\ -C_K K_P[T_D(stc_{D,k+2} - 2stc_{D,k+1} + stc_{D,k}) \\ + T(stc_{D,k+2} - stc_{D,k+1}) + T^2 stc_{D,k+2}/T_I] \quad (18) \end{aligned}$$

which is much more amenable to analysis. Inserting the ansatz $stc_{D,k} = RW^k$ into (18) yields:

$$\begin{aligned} D(W) \equiv W^3 + C_K K_P[T^2/T_I + T + T_D - 2/C_K K_P]W^2 \\ + C_K K_P[1/C_K K_P - 2T_D - T]W + C_K K_P T_D = 0 \quad (19) \end{aligned}$$

This is a cubic equation for W. For stability in the system as a whole, we need $|W| < 1$ for all the solutions of $D(W) = 0$. Standard resources for cubics such as e.g. [29], [30], show the technical burden of trying to analyse this directly.

We observe that because of the sign of the W^3 term in (19), if all the roots of $D(W) = 0$ are real and of modulus < 1, then (1) $D(+1) > 0$, (2) $D(-1) < 0$, and if D' is the derivative of D, then (3) the roots $r_\pm$ of $D'(W) = 0$ satisfy $-1 < r_- < r_+ < +1$, (4) $D(r_-) > 0$, (5) $D(r_+) < 0$. Since $D'(W) = 0$ is a quadratic it is a lot easier to handle. (We note that the constraints cited can be related to the Sturm technique for finding regions containing real roots of an arbitrary polynomial [31].)

Although the general form of the coefficients of (19) makes it cumbersome to test for the conditions (1)-(5) in full generality, we note that we are predominantly interested in the region $T \to 0$. If any necessary conditions do not hold in the limit of vanishing sampling interval, then they cannot be of interest for any engineering purpose. The $T \to 0$ limit simplifies the coefficients considerably.

Beyond this, we can rely on the physical properties of the system to simplify the case analysis further. From the Cohen-Coon tuning analysis of Section II, it emerges that $T_D \approx 10T$ and $T_I \approx 4T_D$. As well, the kinematics of the problem mean that C_K is positive, and it also follows that K_P is positive. So all the constants in our problem space are positive.

All of these observations make the corresponding tests relatively straightforward to carry out. Tests (1) and (2) are straightforward evaluations. The former yields a triviality while the latter yields the constraint:

$$1 > C_K K_P T_D \quad (20)$$

which turns out to be necessary for the small T limit to be feasible. Constraint (3) gives rise to a number of further conditions. However, in the small T limit, they are all subsumed by the stronger condition (20).

The expressions for the roots $r_\pm$ of $D'(W) = 0$ are the standard formulae for a quadratic, and turn out to be expressions in the combination $C_K K_P T_D$. Accordingly, it is easiest to use (20) to substitute numerical values for $C_K K_P T_D$ and thence to check conditions (4) and (5). It turns out that values of $C_K K_P T_D$ around approximately 0.5 permit (4) and (5) to be satisfied in the small T limit. We thus conclude that there is a stable regime in the small T region, and hence furthermore, that there is a still larger region of stability when a pair of roots of $D(W) = 0$ fuses and bifurcates into a pair of complex roots (which will be close to the real axis for some range of values of the parameters).

We concede that the above analysis was somewhat *ad hoc*. More significantly, it was purposely confined entirely within the state space formulation of the problem. This is important to the extent that the HEB approach is lodged in the state space domain for reasons which were explained in the Introduction.

By contrast, most discretizations of continuous designs in engineering practice take place within the frequency domain. As mentioned previously, there are various approaches described in the literature cited earlier. Happily, one of them coincides with what we derived: the discrete equivalence approach. In that approach, a zero order hold is introduced into the model at the right point, standard z-transform elements are introduced for the PID components, and a transfer function is calculated by combining all of these. The poles of the transfer function give the characteristic frequencies of the system, which are checked for stability. It turns out that the denominator of the transfer function (which is a rational function in the z plane), coincides exactly with (19) aside from the change of variable.

In the conventional approach to discretization, stability is analysed using the Jury test [32], [26], [8], which is applied in the z domain. This generates a sequence of tests, all of which have to be passed to deduce stability (which is the property $|z| < 1$ for the characteristic frequencies). Happily once more, the first few of these coincide with the first few of the *ad hoc* tests we did above. All of this shows not only the desirability, but the feasibility of greater cooperation between the two formulations of control within the formal HEB approach.

D. Relating the Continuous and Discretized Models

The cornerstone of any formal development technique like (Hybrid) Event-B is the idea of relating successive models via a formal refinement relation, which relates a more abstract model to a more concrete one. In practice this is always a simulation relation, which amounts to the statement: *IF* the invariants hold at a given moment *THEN* they hold after any update (mode or pliant) of the concrete variables — for a suitable choice of update of the abstract variables. Thus, suitably initialised, the implicational structure can be cascaded inductively into a statement that holds true at all times. Evidently, for the described approach to have force, the invariants mentioned must express a desired relationship between the two families of variables that is also expected to hold at all times.

Establishing this, when viewing discretization as an instance of refinement, proves to be very demanding in all but the simplest cases. A trivial case of discretization treated this way occurs in [9] — no technical difficulties occur there. From our own development, an equally trivial case of refinement occurs between machines *KurtUser_Mch* and *KurtUserD_Mch*, since in the INVARIANTS section of the *YawCtrlD_IF* interface we find the joint invariants $thr_D = thr$ and $yrr_D = yrr$, which express the equality of the relevant pairs of variables. Since the original and discretized variables are defined to behave in exactly the same way in their respective machines, establishing the required properties is indeed trivial, and *KurtUserD_Mch* is a genuine refinement of *KurtUser_Mch*.

For the other machines in the model, the situation is a lot less clear cut. The detailed behaviour of the continuous and discretized variables in each corresponding pair is known a lot less precisely. In particular, there is a significant lack of detail compared with what is stated regarding *thr* and *yrr* and their counterparts. A number of points arise concerning this.

Firstly, given a linear and third order discretization scheme, with sufficient additional effort an analytic solution could in principle be obtained for the various variables involved, at least in the form of summations over powers of the roots. The effort involved would be considerable, yet the perspicacity of the solution obtained would not be a given. With a relatively opaque formulation of the solution, its relationship with the analytic solution to the continuous system (which we did not pursue to its conclusion either) would also not be clear. This would lead to further technical difficulties in formulating relevant joint invariants for corresponding pairs of variables. It is not evident that they would enjoy the same level of precision that we were able to indicate for *thr* and *yrr*.

Secondly, the effort expended in achieving the goals indicated (if actually expended) would only apply for the single pair of abstract and concrete system trajectories given by the specific driving inputs specified by *thr* and *yrr* in *KurtUser_Mch* and its counterpart. They would not necessarily apply for any other inputs. But, for dependability, we would want a generic result that applied to a whole family of trajectories that was large enough to include all that could be expected to arise in practice. For that, much more generic and powerful results would be needed. And this mismatch between what typical formal techniques routinely demand, and what the analysis of control systems can routinely supply, rapidly grows greater the more complex the application system becomes.

Based on these observations, the relationships that we have quoted in the *YawCtrlD_IF* interface between continuous and discretized versions of our variables (excepting *thr* and *yrr*) is approximate equality. Some justification for this lies in the fact that we were able to show that both systems were stable (for suitably chosen static parameters).

For us, approximate equality meant $|var_D - var| < B_{var}$, for corresponding variable pairs and constants B_{var}. When the stability properties are sufficiently strong, such results can become provable in principle; see e.g. [16]. Still, it is not unquestionably the case that these results can be related directly to problem domain quantities, because of their reliance on existential properties.

The claim that $|var - var_D| < B_{var}$ can serve as a suitable joint invariant depends crucially on knowing that the dynamics is such that the difference between continuous and discretized versions of variables always strictly decreases, i.e. that we always have behaviour consistent with asymptotic stability, *regardless of the behaviour of any external input*. However, in Section VI-A, we acknowledged the possibility of impoverished information from sampling allowing the discrepancy between variables to grow, even if temporarily. In such cases, the arguments in [16] will not hold.

In cases like that, it is often not too hard to prove *local properties*, i.e. properties concerning a single sampling interval, that assert not-too-bad divergence. The downside of such properties though, is that they do not cascade inductively into statements that hold at all times.

A variation on the refinement technique that is focused on such local properties is retrenchment [33], [34], [35], [36]. It provides a formal framework in which such local properties can be expressed, and additionally, related to properties controlled by refinement (see particularly [35]). A judicious combination of refinement and retrenchment could prove to be the best approach to the technical difficulties mentioned.

Of course, we are not the first authors to discuss the challenges posed by discretization. As well as standard discrete control references such as [32], [26], [25], [37], [8], there are more specialised treatments aimed at specific aspects, e.g. [38], [39], [40], [41]. Frequently they focus on a frequency domain approach or on statistical properties. It is probably fair to say though, that there is no detailed treatment that is an ideal match to the needs of a formal approach to control systems design and development.

VII. Conclusions

In the preceding sections, we took a simplified though non-trivial version of the yaw control problem for the KURT e-vehicle, and used it as a testbed for complementing and strengthening the assurance obtainable via conventional engineering approaches, by using a formal modelling and refinement approach to the development. The vehicle for the latter was the Hybrid Event-B formalism [9], [10], a formalism designed to capture hybrid system behaviour in way that is compatible with established formal development approaches from the computing sphere, and with state based approaches from the control sphere.

One immediate consequence of this is the evident tension between the state based perspective of formal approaches (which need to deal with arbitrarily structured state spaces) and the frequency domain based perspective of conventional engineering design. The frequency domain approaches, typically based on Laplace transforms and z-transforms, turn functional properties into algebraic ones — the latter are usually much easier to manipulate in practice, and thus are strongly favoured in practical engineering. The simplification of the design process coming from the use of algebraic techniques is amplified by the use of specific stimuli (such as response to step function inputs), and of simulation, as preferred techniques to gauge the appropriateness of a design. Of course, to quote a familiar truism, simulation can only show the presence of faults, not their absence, so enhancing the development methodology with formal techniques which potentially have broader coverage is a worthwhile aim.

Doing this seriously though, quickly invites back the technical obstacles that the algebraic and simulation based approaches strove to avoid. We rapidly saw this in our examination of the discretization of PID control in the KURT e-vehicle application — the technical difficulties inherent in moving from a relatively perspicuous continuous design to a suitable discretized version quickly proliferated.

These observations readily propose a series of topics that it would be very worthwhile to develop more fully in order to increase the utility of a combined approach.

[I] A closer cooperation is needed between, on the one hand, the state space methods typical in state based control and formal approaches, and on the other hand, the frequency domain based approaches so widely used in conventional engineering. Normally, discussions take place exclusively within one domain or the other, but a greater interaction could improve the feasibility of a combined approach. For this, and depending on the specific goals of a given analysis, more contact points would be identified between state space and frequency methods, such that the solution to a state space question could be derived in the frequency domain. However, it has to be appreciated that only certain questions translate well between the two domains.

[II] With issues such as discretization, it is clearly impractical to tackle each development from first principles in a practical methodology. A family of suitable generic results is needed that can be applied in a wide variety of contexts to inform the formal strand of the development. Similar remarks apply to other aspects of the development methodology — we could mention topics such as sensitivity and robustness, besides stability, which we concentrated on.

[III] To facilitate the efficient incorporation of the relevant kinds of formal derivation (done by hand in an *ad hoc* manner in this paper) additional special purpose syntactic support could be provided within the Hybrid Event-B formalism, especially in the context of mechanised tool support.

Besides these issues concerned with the technical interaction between different approaches, lies the application level scope of the formal modelling and refinement. We confined our attention to stability in an ideal environment, thereby neglecting many things. In reality, friction and sensitivity to external influences in the equipment affect the behaviour of the system. If these are not modelled properly, then the rigour of a formal refinement becomes spurious. Likewise statistical fluctuations in both the equipment and the environment of operation impacts the behaviour, and needs to be taken into account.

The traditional engineering approach to such questions is again via the frequency domain. It is assumed that influences such as these are each characterised by a suitable frequency domain response profile, often determined experimentally where needed. With a characterisation of that kind to hand, the controller can be designed to filter out stimuli that are undesired, and to respond appropriately to those that are important. With such a bandwidth limited frequency domain controller design, the Nyquist theorem and engineering heuristics give a good guide to the sampling frequency needed for a dependable discretization. However, this kind of frequency domain derivation is rather far from the state space approaches that could be directly placed in a formal development framework. All of these topics provide fertile territory for further work, to be pursued in other publications.

REFERENCES

[1] K. Ogata, *Modern Control Engineering*. Pearson, 2008.

[2] R. Dorf and R. Bishop, *Modern Control Systems*. Pearson, 2010.

[3] K. Dutton, S. Thompson, and B. Barraclough, *The Art of Control Engineering*. Addison Wesley, 1997.

[4] Modelica Homepage, https://www.modelica.org/.

[5] J. Van Leeuven, *Handbook of Theoretical Computer Science, Vol. A and Vol. B*. Elsevier, 1990.

[6] E. Sontag, *Mathematical Control Theory*. Springer, 1998.

[7] N. Ahmed, *Dynamic Systems and Control With Applications*. World Scientific, 2006.

[8] D. Hinrichsen and A. Pritchard, *Mathematical Systems Theory I*. Springer, 2005.

[9] R. Banach, M. Butler, S. Qin, N. Verma, and H. Zhu, "Core Hybrid Event-B I: Single Hybrid Event-B Machines," *Sci. Comp. Prog.*, 2015, to appear.

[10] R. Banach, M. Butler, S. Qin, and H. Zhu, "Core Hybrid Event-B II: Multiple Cooperating Hybrid Event-B Machines," 2015, submitted.

[11] K. Kozłowski and D. Pazderski, "Modeling and Control of a 4-Wheel Skid-Steering Mobile Robot," *Int. J. Appl. Match Comput. Sci.*, vol. 14, pp. 477–496, 2004.

[12] ControlsWiki, https://controls.engin.umich.edu/wiki/index.php/PIDTuningClassical.

[13] K. Astrom and T. Hagglund, *Advanced PID Control*. ISA, 2006.

[14] K. Ogata, *System Dynamics*. Pearson, 2013.

[15] J. R. Abrial, *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.

[16] P. Tabuada, *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer, 2009.

[17] A. Platzer, *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, 2010.

[18] L. Carloni, R. Passerone, A. Pinto, and A. Sangiovanni-Vincentelli, "Languages and Tools for Hybrid Systems Design," *Foundations and Trends in Electronic Design Automation*, vol. 1, pp. 1–193, 2006.

[19] S. Hallerstede and T. Hoang, "Refinement by Interface Instantiation," in *Proc. ABZ-12*, Derrick, Fitzgerald, Gnesi, Khurshid, Leuschel, Reeves, Riccobene, Ed., vol. 7316. Springer, LNCS, 2012, pp. 223–237.

[20] W. Walter, *Ordinary Differential Equations*. Springer, 1998.

[21] C. Chicone, *Ordinary Differential Equations with Applications*, 2nd ed. Springer, 2006.

[22] P. Antsaklis and A. Michel, *Linear Systems*. Birkhauser, 2006.

[23] Mathematica Homepage, http://www.wolfram.com.

[24] J. D'Azzo and C. Houpis, *Linear Control System Analysis and Design: Conventional and Modern*. McGraw Hill, 1995.

[25] G. Franklin, J. Powell, and M. Workman, *Digital Control Systems*. Prentice Hall, 1996.

[26] P. Paraskevopoulos, *Digital Control Systems*. Prentice Hall, 1996.

[27] E. Isaacson, *Analysis of Numerical Methods*. Dover, 2003.

[28] A. Iserles, *A First Course in the Numerical Analysis of Differential Equations*. Cambridge University Press, 1996.

[29] Wikipedia, "Cubic function."

[30] F. Olver, D. Lozier, R. Boisvert, and C. Clark, *NIST Handbook of Mathematical Functions*. Cambridge University Press, 2010.

[31] Wikipedia, "Sturm's theorem."

[32] B. Kuo, *Digital Control Systems*. Oxford University Press, 1992.

[33] R. Banach, M. Poppleton, C. Jeske, and S. Stepney, "Engineering and Theoretical Underpinnings of Retrenchment," *Sci. Comp. Prog.*, vol. 67, pp. 301–329, 2007.

[34] R. Banach, C. Jeske, and M. Poppleton, "Composition Mechanisms for Retrenchment," *J. Log. Alg. Prog.*, vol. 75, pp. 209–229, 2008.

[35] R. Banach and C. Jeske, "Retrenchment and Refinement Interworking: the Tower Theorems." *Math. Struc. Comp. Sci.*, vol. 25, pp. 135–202, 2015.

[36] Retrenchment Homepage, http://www.cs.man.ac.uk/retrenchment.

[37] M. Fadali and A. Visioli, *Digital Control Engineering: Analysis and Design*. Academic Press, 2009.

[38] B. Widrow and I. Kollar, *Quantization Noise*. Cambridge University Press, 2008.

[39] R. Marks, *Introduction to Shannon Sampling and Interpolation Theory*. Springer, 1991.

[40] R. Pytlak, *Numerical Methods for Optimal Control Problems with State Constraints*, ser. Lecture Notes in Mathematics. Springer, 1999, vol. 1707.

[41] L. Grune, *Asymptotic Behavior of Dynamical and Control Systems under Perturbation and Discretization*. Springer, 2002.