Journal of Software Engineering for Robotics 7(1), July 2016, 100-119
ISSN: 2035-3928

Modeling Electromechanical Aspects of Cyber-Physical Systems

Yingfu Zeng[1] Chad Rose[1] Walid Taha[1,2] Adam Duracz[2]
Kevin Atkinson[1] Roland Philippsen[2] Robert Cartwright[1] Marcia O'Malley[1]
[1] Department of Computer Science, Rice University, Houston 77051, USA
[2] School of Information Technology, Halmstad University, Halmstad 30118, Sweden

Abstract—Model-based tools have the potential to significantly improve the process of developing novel cyber-physical systems (CPS). In this paper, we consider the question of what language features are needed to model such systems. We use a small, experimental hybrid systems modeling language to show how a number of basic and pervasive aspects of cyber-physical systems can be modeled concisely using the small set of language constructs. We then consider four, more complex, case studies from the domain of robotics. The first, a quadcopter, illustrates that these constructs can support the modeling of interesting systems. The second, a serial robot, provides a concrete example of why it is important to support static partial derivatives, namely, that it significantly improves the way models of rigid body dynamics can be expressed. The third, a linear solenoid actuator, illustrates the language's ability to integrate multiphysics subsystems. The fourth and final, a compass gait biped, shows how a hybrid system with non-trivial dynamics is modeled. Through this analysis, the work establishes a strong connection between the engineering needs of the CPS domain and the language features that can address these needs. The study builds the case for why modeling languages can be improved by integrating several features, most notably, partial derivatives, differentiation without duplication, and support for equations. These features do not appear to be addressed in a satisfactory manner in mainstream modeling and simulation tools.

Index Terms—Domain-Specific Language, Cyber-Physical Systems.

1 Introduction

Model-based design tools can revolutionize the process of developing new products. This is especially the case for cyber-physical systems (CPSs), which consist of networks of computational and physical components. The presence of computational and networking components means a bigger state space and more non-determinism, both of which limit the utility of physical testing. This observation has also been made in the context of automotive systems with advanced driver assistance and autonomy functions [4],[5], and smart home design [6].

Model-based tools provide functions on models such as simulation, analysis, verification, and conformance testing [7], [8], [9], [10], [11]. Using such tools shifts the user's focus from building and maintaining disjointed software artifacts like the "simulation code" or the "analysis code" to "the model of the product" being studied or developed. In addition to what is often described as "raising the level of abstraction," the model-based approach reduces the pressure to maintain different (often implicit) models of the same system. Avoiding such duplication reduces the work needed to ensure the consistency of these different models.

Modeling and simulation tools can improve software development for robotics in two ways. First, they can virtualize the robot and allow fast and cheap testing of real robots. Second, they make it easier to explore and gradually develop the specification of software and hardware before we start implementing it. For example, we can start with an idealized controller, gradually adding more realistic constraints such as quantization, discretization, and energy consumption.

Regular paper – *Manuscript received September 02, 2015; revised June 1, 2016.*

- *This manuscript revises and consolidates two papers published under similar titles at the DSLRob 2012 [1] and DSLRob 2013 [2] workshops. A shortened revision of the second paper was also published at ICESS 2014 [3]. The work reported on is substantially extended, including the addition of new examples and case studies. A key technical improvement on what is presented in these papers is the introduction of support for Lagrangian modeling in Acumen. This makes it possible to express the dynamics of the RiceWrist-S model in a more direct manner. The syntax of the language has also been updated, as well as the 3D graphics library (and associated figures in the paper).*
- *This work was supported by US NSF CPS award (No.1136099), Swedish KK-Foundation CERES and CAISR Centres, and the Swedish SSF NG-Test.*

The overarching goal of our research is to develop the semantic foundations for rigorous simulation languages for the cyber-physical systems domain (see, for example [12], [13], [14], [15], [16], [17]). Because semantics research must focus primarily on minimal calculi, it is critical that the choice of constructs in a calculus has domain-based justification that is grounded in an accurate understanding of the needs of the CPS domain. There is a need for guidelines for the demands of the CPS domain in terms of modeling language features, as well as evaluations of the extent to which particular features are able to meet these demands. Domain-based justification is of significance not only for our own research but for any other research effort on the semantics of hybrid systems modeling languages. Beyond the research community, such guidelines can also illuminate the design space for providers of such tools. It affects decisions about syntax, semantics, building implementations, and understanding how a language could and should be used. This paper reports on the results of our efforts to address this question.

1.1 Contributions

The main contributions of this paper are:

1) Identifying a number of aspects that are representative of some of the most basic and prominent modeling needs of the CPS domain.
2) Showing that a small language for modeling hybrid (continuous/discrete) systems is sufficient for modeling these aspects.
3) Reporting on four larger case studies expressed within the same language.

After introducing the syntax and the informal semantics of Acumen, a minimalist language used for the investigation (Section 2), we compare it with other standard modeling and simulation tools that are widely used in the robotics domain (Section 3). Then we turn to the different aspects identified and treated. Each section explains an aspect and how it can modeled in a minimal formalism. Simpler and more self-evident aspects are presented in earlier sections. Naturally, a larger part of these earlier sections is spent explaining the formalism than is the case in later sections.

Visual and geometric presentation is a critical aspect of analytical modeling that can hide in plain sight. We begin by showing how to support visual and geometric presentation in both static and dynamic scenes, as well as in simple and composite components (Section 4). Traditionally, visualization is not part of analytical modeling, but it is indispensable for efficiently understanding both the specification and the results of model-based simulations or analysis. Basic mechanics and dynamics come next (Section 5), along with a range of analytical principles used to model physical systems. After a brief introduction to the notion of sub-models (Section 6), we return to expressing the differential equations needed to model control (Section 7). We then consider how to express a more realistic model of control systems by capturing the way in which implementation on a digital computer introduces both discretization and quantization (Section 8).

The four larger case studies carried out as part of this work are drawn from the robotics domain. The first is a quadcopter, a non-trivial, rigid body system that is often used as a CPS example. This case study shows that the language can simply and directly express Newtonian models (Section 9). The second case study is a research robot called the RiceWrist-S. For this robot, developing a Newtonian model is difficult and inconvenient. The case study helps illustrate how the more advanced technique of Lagrangian modeling can be advantageous for some problems. As a prelude to modeling the RiceWrist-S robot [18], we consider two ostensibly simple dynamic systems, namely a single pendulum and a double pendulum. For the second system, we show that Lagrangian modeling leads to a much simpler mathematical formalization (Section 10). We then confirm this insight by showing how Lagrangian modeling enables us to construct a simple model of the dynamics for the RiceWrist-S (Section 11). The third case study is an example of an integrated electromechanical system, namely a linear solenoid actuator. This case study illustrates the core language's ability to model multiphysics, such as coupled mechanical and electrical subsystems. Whereas the first three case studies are continuous systems, the fourth study (Section 13), a compass gait biped, is a classic example drawn from the domain of robotic walking. It shows how to model hybrid systems with non-trivial dynamics using the core language. We summarize our observations about the needs of the CPS domain and language constructs (Section 14), and then conclude.

2 A Small, Experimental Language for Hybrid Modeling

In this section we introduce the syntax and the (informal) semantics for the small language that we will use in the rest of this paper.

The syntax for the Acumen language [25], [26] consists of the following constructs:

- Constant literal values (e.g., `true`, `5`, `1.3`, `"Hello"`)
- Vectors and matrices (e.g., `(1,2)`, `((1,2),(3,4))`)
- Expressions and operators on ground and composite types (+, −, ...)
- Model definitions (`model C (x,y,z) = ... )`
- Model instantiation and termination operations (`create`, `terminate`)
- Variable derivatives (`x'`, `x''`, ...) with respect to time
- Variable declarations (`initially ... always`). For convenience, we included in the set of variables a special variable called `_3D` for generating 3D animations.
- Time and partial derivatives (e.g., `(x^2)'`, `(x+y)'[x]`)
- Continuous equation and discrete equations (=, + =)
- Conditional constraints (`if`, and `match`)

TABLE 1: Comparison of Modeling and Simulation Tools

	MATLAB R2015b [19]	Simscape R2015b [11]	Octave 4.0 [20]	OpenModelica 1.9.3 [21]	Dymola 5.3 [22]	Mathematica 7.0 [23]	20-sim 4.6 [24]	This paper
Partial derivatives	Yes	No	No	Limited	No	Yes	Yes	Yes
Compact derivatives	No	No	-	-	-	No	Unknown	Yes
Support for equations	No	Yes	No	Yes	Yes	Yes	Yes	Yes

Derivatives can be applied to expressions or variables. The time derivative on a variable has special status, in that it can both be used in expressions to mean the value of the derivative at a given time and can also be equated to a value. When there is a constraint that equates a time derivative of a variable to a value, the effect is that integration can be used to compute the value of the variable itself. In principle, one can imagine that, in an equational language, if a symbolic expression for the variable is known, the derivative variable can be determined from that expression. In practice, it is rare that a closed form expression for the result of a simulation is known. Instead, it is more common to have the value of the derivative known, and then numerical integration can be used to compute the value of the variable itself. The partial derivative is an operator that takes two expressions and returns the result of the first expression differentiated with respect to the second expression. The syntax for this operator is `expr'[expr]`. For two scalars, the result is simply the first expression partially differentiated with the second one. If one expression is a scalar, and the other a vector, the operator is applied to all elements of the vector. For two vectors, a matrix is generated where every row is the result of the corresponding element in the first vector, partially differentiated with the second vector. For example, let $f = x + y$, $g = (x^2, y^2)$ and $q = (x, y)$, then the operators $f'[x]$ and $f'[y]$ would both result in 1. Moreover $g'[x]$, $f'[q]$, $g'[q]$ will return $(2 * x, 0)$, $(1, 1)$ and $((2 * x, 0), (0, 2 * y))$ respectively. Allowing arbitrary expressions instead of just variables in partial derivatives allows us to express families of equations like the Euler-Lagrange equation directly.

Continuous equations are used for equating continuous behaviors (as in differential equations). Discrete equations are use to define values at discontinuities, where an instantaneous change of a value (a reset) can occur in juxtaposition to continuous dynamics. Initial values for variables (at the time of the creation of a new instance) are treated as discrete equations. Currently, we take a conservative approach to initial conditions, which requires users to express them explicitly even for variables where there is a continuous equation that may immediately override this explicit initial value. For the sake of minimality, Acumen has no special notation for introducing constants (in the sense of variables that do not change value over time).

This language is used for a term-long project in a course on CPSs [27], which has been enthusiastically received in the first four offerings (see for example [28], [29]). The parsimonious language seems to help students connect different concepts and avoid the introduction of artificial distinctions between manifestations of the same concept in different contexts. This bodes well for the utility of such a language for reducing the need for different notations to model the same system during different phases of a product development process. However, to fully overcome this challenge, we must develop a clear understanding of how different features in such a language match up with the demands of different types of cyber-physical systems.

The Acumen distribution contains implementations of differential equation solvers for simulation, accessible from a "Semantics" menu. For this paper, we pick the "Traditional" semantics, which simply uses a Runge-Kutta method and a constant time step for integration.

3 Related Work

Today, a wide range of tools exists to support the modeling and simulation of cyber-physical systems [7]. In this section we identify the main differences between these tools and the language that used in the present research. Table 1 provides an overview of how several related tools compare on key properties of relevance to the present work. The table summarizes the presence of the features discussed in this paper in important modeling languages. We emphasize that Acumen is an experimental research prototype, and that the other languages have numerous features not present in Acumen. Furthermore, we have not evaluated how complex it would be to add such constructs to the existing languages. The properties are:

- Partial derivatives: The ability of a tool to support the expression of a partial derivative of two expressions directly.
- Compact derivatives: The ability to compute symbolic derivatives without producing exponentially large expressions.
- Support for equations: The ability to write equations instead of just assignments.

The main observations summarized in the table are as follows. Scripting or programming languages such as Octave [20], MATLAB, and its graphical extension Simulink, focus on providing a convenient programming language for a broad class of scientific problems. However, these formalisms are not intended to capture equational mathematical models directly.

For example, they limit the user to only expressing so-called causal models, which are not expressed as equations, but rather, as directed (continuous or discrete) assignments. The Simscape [11] language from MathWorks is based on MATLAB and is a dedicated textual language for modeling physical systems. It supports equations but not partial derivatives. To express equations that require partial derivatives, one must use another toolbox to compute the derivative symbolically and then copy and paste the result into the model manually [11]. Copying and pasting are problematic because they make it difficult to avoid code duplication in differentiation, and harm the traceability/maintainability of the code.

Symbolic algebra tools such as Mathematica [23] and Maple, in principle, can assist in manipulating analytical models and in making them executable. In practice, however, as the size of the physical system being modeled increases, symbolic computing systems can both take an exponentially long time to compute a symbolic result and produce a result that is exponentially larger than necessary. To avoid this problem, Acumen uses a specialized symbolic differentiation procedure that operates on partial derivatives as they appear in the context of the equations. The procedure computes compact derivatives by avoiding any inlining and by reusing any expressions either appearing in the original system of equations or that have been generated during symbolic differentiation anywhere in the system of equations. For many common situations, this procedure outperforms Mathematica by being polynomial both in time and size, with respect to size of the term being differentiated [12].

With regard to the discussion of Lagrangian modeling, the paper includes a comparison between code in Acumen, MATLAB, and Modelica. The code for the latter two languages is included in the Supplemental Material (Section 14).

OpenModelica [21] and Dymola support equational modeling and provide many solvers for differential algebraic equation (DAE) systems. However, partial derivatives of arbitrary expressions are not directly supported in the current implementation of Modelica. While partial derivatives can be included by advanced Modelica users, for example, with inline semantics and the user-defined `pder()` operator [30], this indirect method does not lend itself to many modeling methods, such as the Euler-Lagrange.

The language that comes closest to supporting the language features emerging from our study is 20-sim. It is a commercial modeling and simulation language for multidomain dynamic systems. It supports both equational modeling and partial derivatives [24]. However, it is unclear whether the symbolic differentiation produces a compact result.

Finally, it is worth addressing the relation to the Robot Operating System (ROS), which is not addressed in Table 1. ROS is a middleware designed to be a communication interface for sending and receiving sensor data in many robotic devices [31]. It is concerned more with interactions between each individual component rather than the overall design of the mechatronic system. Thus, it does not help with modeling the physical context or the intended continuous dynamics of the system. Rather, it is intended to help with implementing real-time control of the hardware.

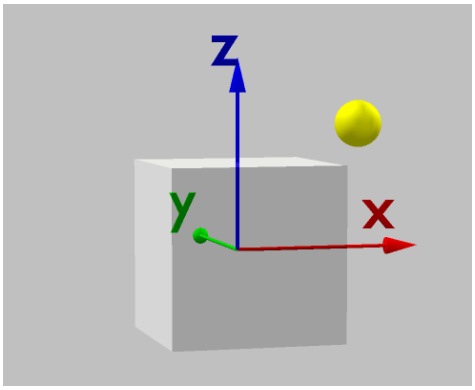


Fig. 1: The 3D output generated for an instance of the model sphere.

4 Geometry and Visual Form

Most physical systems either take up space or have an effect on space. As a result, visual presentation has a role in CPS design. For many people, it is hard to imagine a design without conjuring an image of a general visual form. If we want to replace physical prototyping with virtual prototyping, visualization becomes a necessity. Animating the evolving state of a hybrid system using visual geometric presentation often reveals behaviors of the system that might otherwise be undetected.

A hybrid modeling and simulation language can be naturally extended with a lightweight mechanism for three-dimensional (3D) visualization [32]. In the core language of Acumen, the user can specify 3D visualizations through a special variable called `_3D`. This variable is read by the implementation and used to generate a dynamic 3D scene. In principle, any graphical rendering technology can be used by an implementation to realize these visualizations. The current implementation uses the jPCT library [33] and supports four primary objects: Sphere, Cylinder, Box, and Cone. The following Acumen model illustrates how simple a visualization primitive can be:

```
model sphere (m,D) =
 initially
  q = (0,0,1), _3D = ()
 always
  _3D = (Sphere center = D+q
         size = 0.03*sqrt(m)
         color = (m/3,2+sin(m),2-m/2))
```

This is a custom-made model that a user may have created to represent a particle with a given mass and position. The

parameter m represents the mass of the sphere. In this example the mass is reflected in the size and color of the sphere object displayed in the 3D view. The parameter D is a display reference point. Passing different D values to an individual instance facilitates creating visualizations where instances of the same model can appear in different places. The initially section declares variables present in each instance and their initial values at simulation time, when an instance of the model is created. The variable q represents the position of the sphere. The special variable _3D must be bound to a vector with a format determined by the 3D visualization system used in our implementation of the core language. The continuous equation is computed for as long as the instance exists in simulation. The _3D vector has the following format. The first field, in this case Sphere, indicates the shape we want in the visualization. The second field is the coordinates for the center of the shape. The third field is the radius. In this example the user has chosen to make the radius a simple function of the mass. This function is not intended to have any physical meaning other than to produce an illustrative visualization. The next field is a vector that represents the red/green/blue (RGB) color components for this sphere, using an *ad hoc* formula to generate a color based on the mass. Fig. 1 depicts a visualization generated using this model.

We can create an instance of the model described above by writing s = create sphere (5,(0,0,0)) in the initialization (initially) section and then s.q = (0.1,0.2,0.3) in the body. To generate 3D animations, all we have to do is to let the value of q vary over time, as in the following code:

```
model moving_sphere (m, D) =
 initially s = create sphere(m,D),
           t = 0, t' = 0
 always
  t' = 5,
  s.q = (sin(t)*sqrt(1 - (sin(t/10)^2)),
        cos(t)*sqrt(1 - (sin(t/10)^2)),
        sin(t/10))
```

Here the variable t and its derivative t′ are introduced to model a local variable that progresses at exactly five times the rate of time. All that is needed to accomplish this is to include the equation t′=5. The time-varying variable t is then used to generate some interesting values for the X, Y, and Z components of the position field q that represents the center of the sphere model s.

As noted earlier, we can place different instances of a model (such as moving_sphere) at different locations by varying the D parameter. By changing the value of the position parameter q, we can create an animation with two spheres moving in a synchronized fashion.

It is useful to note that a 3D visualization facility can be used to visualize not only spatial parameters and dimensions but also abstract values such as energy. For example, it can be useful to define models that assist in visualizing such values during a simulation. The following code defines a model to visualize a scalar value as a cylinder, whose length is proportional to that value:

```
model display_bar (v,c,D) =
initially
  _3D  = ()
always
  _3D = (Cylinder center=D+(0,0.2,v/2)
         radius=0.02 length=v  color=c
         rotation=(-1*pi/2,0,0))
```

Following the 3D primitive name Cylinder, the next argument represents the center of the cylinder. We take this to be v/2 because this will allow us to keep one end of the cylinder fixed as the value of v changes. The next two arguments specify the radius and length of the cylinder. The next argument is color, and the last specifies orientation angles for the cylinder.

In addition to having a mechanism for specifying visual form, working with geometry places some intuitive but nevertheless specific requirements on the modeling languages. In particular, it is generally necessary to perform some vector and trigonometric calculation to create the desired shape. This need arises even in simple situations. An example of such a context is drawing a cylinder between two points. Often, visualizations cannot be specified directly because the underlying library uses a different approach to describe the orientation of a figure. In the case of cylinders, it is common to use polar coordinates (two angles) to specify the orientation of the axis of a cylinder. Once we have figured out all necessary calculations, they can be encapsulated in one model as follows:

```
model cylinder (D) =
initially
  q1  = (0,0,0), q2 = (0,0,0),
  dis = (0,0,0), r  =  0.01,
  l  =  0.01, alpha = 0, theta =  pi/2,
  x = 0, y = 0, z = 0, _3D  =  ()
always
  dis = q1 - q2,
  x = dis(0), y = dis(1), z = dis(2),
  l = norm(q1-q2), alpha = asin(z/l),
  if y>0 then
    theta = asin(x/(l*cos(alpha)))
  else
    theta = -asin(x/(l*cos(alpha)))+pi,
  _3D = (Cylinder center=(q1+q2)/2+D
         radius=r length=l color=(1,1,1)
         rotation=(alpha,0,-theta))
```

The dot and norm operators compute the dot product and the vector norm (or length). Creating such a model is a good exercise in making customized building blocks for visualiza-

tion.

If we are used to programming in a typical programming language where such functions are present, the convenience of such operations is no surprise. From the semantic point of view, what is significant is their necessity. The necessity of supporting transcendental functions means that approximating the set of real numbers with the set of rationals is not possible, as the results of transcendental functions are not rational. In addition to having significant implications for representability and computability of approximations to these operations, their presence implies that nonlinearities are hard to avoid when working with general physical systems. All of these complications arise even before there is any consideration of dynamics or a time dimension in the description of physical systems.

5 PARTICLE DYNAMICS AND IMPACTS

The most basic approach to model the mechanical dynamics of a system is to view it as a point mass, or a *particle*. In contrast to the syntax needed to describe geometric and visual objects, describing particle dynamics can be done more concisely.

A point mass that can only move in one dimension can be represented as follows:

```
model mass_1d (m,q0,D) =
initially
 q = q0, q' = 0, q'' = 0,
 f = 0, e_k = 0,
 s = create sphere (m,D)
always
 q'' = f/m,
 e_k = 0.5 * m * (q')^2,
 s.q = (0,0,q)
```

The model constructor takes three parameters: a mass `m`, an initial position `q0`, and a reference point for visualization. Internally, the mass keeps track of a position `q`, its first and second derivatives `q'` and `q''`, a force `f`, and the kinetic energy `e_k`. For visualization, a `sphere` instance is created during initialization. The body of the model definition specifies that the acceleration of the object, `q''`, is determined by Newton's law $F = ma$, where we are solving for acceleration (which is just `q''` here). Finally, we set the position `q` of the visual sphere instance to be the same as the position `q` of the current instance.

Supporting vector operations makes it possible to define a similar model that has a three-dimensional position almost as simply:

```
model mass (m,q0,D) =
 initially
  q=q0, q'=(0,0,0), q''=(0,0,0),
  f=(0,0,0), e_k=0,
  s = create sphere (m,D)
 always
  q'' = f/m,
  e_k = 0.5 * m * (dot(q',q'))^2,
  s.q = q
```

Note that it is convenient in technical discourse in science and engineering to refer to the derivatives of vectors (and not just scalars) in specifications of dynamics. We can induce continuous behaviors in such models by means of an external continuous equation. For example, the effect of a gravitational force on a mass `m` by a continuous equation `m.f =m.m*(0,0,-9.81)`. The expression for energy uses the built-in dot product operation on vectors. An idealized 3D spring can be modeled as follows:

```
model spring (k,L0,D) =
initially
 q1=(0,0,0), q2=(0,0,0),
 f1=(0,0,0), f2=(0,0,0),
 dl = (0,0,0), e_p=0
always
 dl = q2-q1 * (1-L0/norm(q2-q1)),
 f1 = k*dl, f2 = -k*dl,
 e_p = 0.5 * k * dot(dl,dl)
```

This model associates a different force with each end of the spring. It computes a potential energy `e_p` rather than a kinetic energy. No visualization is included in this model, but this can be easily achieved using the techniques presented above. An important physical effect in dynamics is impact, often modeled as a sudden change. Discrete equations can be used for this purpose. The following model provides an example of the use of discrete equations to model a classic hybrid system, the bouncing ball:

```
model bouncing_ball (D) =
initially
 m  =  create mass_1d (10, 2,D) ,
 bk  =  create display_bar
        (0,(3,0.2,0.2),D+(0.1,0.2,0)),
 bp  =  create display_bar
        (0,(0.2,3,0.2),D+(-0.1,0.2,0)),
 bt  =  create display_bar
        (0,(0.2,0.2,3),D+(0,0.2,0))
always
 m.f = m.m * -9.81,
 if (m.q < 0 && m.q' < 0) then
   m.q' + = -0.9 * m.q'
 noelse,
 bk.v = m.e_k / (m.m * 9.81),
 bp.v =   (m.m * 9.81 * m.q)
         / (m.m * 9.81),
 bt.v = bk.v + bp.v
```

The model uses the mass model along with a continuous gravity model and a ground-impact model, where the ball loses 10% of its velocity. The model `display_bar` is used to

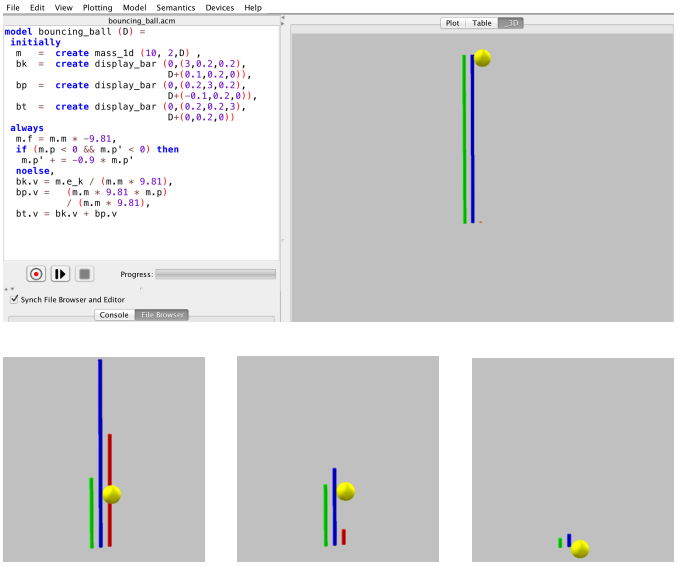


Fig. 2: The IDE of the implementation with the bouncing ball model and simulation results. The green bar indicates the potential energy, the red one is the kinetic energy, and the blue bar is their sum. The total energy decreases with each ground impact, and during the free flight phase the two energies behave as expected.

display colored bars to present some additional information in the 3D output. The mass model used here has only one degree of freedom along the Z axis. We use three display bars to visually represent the kinetic and potential energy, as well as their sum. The discrete equation occurs inside the `if` statement that detects impact with the ground plane. Fig. 2 shows a sequence of screenshots, one including the Integrated Development Environment (IDE), which results from running this example. It can be seen that, as expected, the total energy decreases at each impact, while the kinetic and potential energies reach their respective maxima and minima at the height of the bounce and the impact at ground level.

Thus, to describe even the basic Newtonian dynamics for simple particles, there is a need for not only ordinary differential equations (ODEs) but also hybrid ODEs. It is worth emphasizing that modeling these seemingly elementary aspects of physical systems necessitates the support of continuous equations, derivatives, discrete equations, and conditionals.

6 Composite Models

Most systems are composite, in that they consist of smaller, interacting components. We now consider the specific requirements that the need for modeling such compositions introduces. The most elementary requirement is that of a mechanism to connect components by relating fields in different components though continuous equations. For example, the following model specifies a system consisting of three masses connected by two springs. Note that the model uses an instance

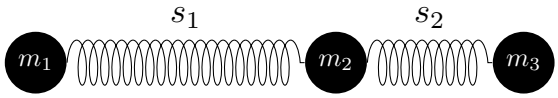


Fig. 3: The diagram of the three-mass/two-spring system.

of the model `display_bar` to draw a cylinder to display the kinetic energy in the system.

```
model example_3 (D) =
initially
 m1  =  create mass_1d (30,1,D),
 m2  =  create mass_1d (10,-1,D),
 m3  =  create mass_1d (5,-1.5,D),
 s1  =  create spring (5,1.75,D),
 s2  =  create spring (5,0.5,D),
 b   =  create display_bar(0,(0.1,3,0.1),D),
always
 s1.q1 = m1.p, s1.q2 = m2.p,
 s2.q1 = m2.p, s2.q2 = m3.p,
 m1.f  = s1.f1,
 m2.f  = s1.f2 + s2.f1,
 m3.f  = s2.f2,
 b.v   = (m1.e_k + m2.e_k +
          m3.e_k + s1.e_p + s2.e_p)*12
```

This example uses the mass and spring model. The system consists of three masses and two springs as shown in Fig. 3. The interaction between them can be expressed though external continuous equations. For example, the effect of the two spring forces acting upon mass `m2` is captured by a continuous equation `m2.f = s1.f2 + s2.f1`.

Some modeling languages such as Modelica support more sophisticated mechanisms for expressing connections. In particular, they support notions of connectors that can serve as sum-equalizing or value-preserving junctions. These notions can make it more intuitive to model systems built by connecting more basic components. However, they do not appear to provide the ability to express connections that cannot be expressed using the more elementary notion of equality presented here. As such, they do not have a natural place in a minimal formalism for modeling cyber-physical systems. Their utility, however, suggests that additional syntactic extensions can make such a minimal formalism more convenient for practical modeling.

7 Control

A pervasive aspect of physical systems is the presence of one or more mechanisms to drive their operation to a certain goal or function. The basic role of control is to bring a certain quantity close to a desired target by manipulating the value of some parameters that affect this quantity. For the model

presented above, and given a controller instance `c`, control can be introduced as follows:

```
// Goal is spring length at rest
 c.r = s1.l+s2.l,
// Value is actual spring length
 c.y = m1.p-m3.p,
 // Add c.u
 m1.f = s1.f1 + c.u,  m2.f = s1.f2 + s2.f1,
// Subtract c.u
 m3.f = s2.f2 - c.u;
```

In this model the goal value for the controller is to have the length of the system be the same as the natural lengths of the two springs. The quantity that we wish to control is the position of the first mass minus the position of the third one. The way we will achieve this is to take a force value `u` that is generated by the controller, and apply it to both sides of the system that we have constructed, but in opposing directions.

Now the question that remains is how the controller `c` should compute its output force `u`, given the goal `r` and measured value `y`. This is a prototypical question in the design of control systems, and one that can be approached in a variety of different ways. Three of the most basic types of controllers are 1) proportional feedback, 2) proportional/differential feedback, and 3) proportional/integral/differential feedback. The first type can be used successfully in some systems, such as those governed by first-order differential equations, or higher-ordered systems that can dissipate energy. It can be modeled as follows:

```
model controller_p (k_p) =
initially
 r = (0,0,0), y = (0,0,0), u = (0,0,0),
 e = (0,0,0)
always
 e = r - y, u = k_p * e
```

The force `u` computed is directly proportional (hence the name) to the error term `e`, which is the difference between the goal `r` and current value `y` of the quantity that we want to control. The higher the constant `k_p`, the higher the force that will be applied for the same amount of difference between the goal value and the current value.

If the system has inertia or does not dissipate the extra energy introduced by the control force, it might oscillate indefinitely as a result of the proportional control. To deal with this problem, a slightly more sophisticated controller can add a force opposing the direction of the motion (or rate of change) of the value being measured. Such a proportional/differential (PD) controller can be defined as follows:

```
model controller_pd (k_p,k_d) =
initially
 r = (0,0,0), y = (0,0,0), u = (0,0,0),
 e = (0,0,0), r_dot = (0,0,0),
 e_dot = (0,0,0), y_dot = (0,0,0)
always
 e = r - y,
 e_dot = r_dot - y_dot,
 u = k_p * e + k_d * e_dot
```

Note that this controller has two extra fields, `r_dot` and `y_dot`, that should be provided from outside the model to serve as the speed reading that should affect the final force `u`.

An interesting feature of the first two types of controllers described above is that they do not keep track of history. We may wish to build a controller that exerts a higher force only after a weaker force has been tested for some time. This can be helpful, for example, if there are external constant forces (such as gravity) acting on our system, and we do not know their precise quantities ahead of time. This type of behavior can be achieved by adopting a proportional/integral/differential (PID) controller such as the following:

```
model controller_pid (k_p,k_i,k_d) =
initially
 r = (0,0,0), y = (0,0,0), u = (0,0,0),
 e = (0,0,0), r_dot = (0,0,0),
 e_dot = (0,0,0), y_dot = (0,0,0),
 e_i = (0,0,0), e_i' = (0,0,0)
always
 e = r - y,
 e_dot = r_dot - y_dot,
 u = k_p * e + k_d * e_dot + k_i * e_i,
 e_i' = e
```

The variable `e_i` is being used to integrate the difference between the goal `r` and the value `y` over time, so no extra inputs are needed.

Using the formalism presented above, it is easy to model several instances of the three-mass/two-spring example, and to visualize the result of simulating it by showing both the behavior of the mass and the energy of the system with different controllers. The controllers presented here illustrate the design of basic, idealized control. The language is also expressive enough to capture more realistic models, such as when a system does not have velocity as input but rather uses discrete sampling of position to compute an estimate of velocity that is then used in the PD controller. The experiment shows that a P controller will not dissipate any energy and therefore will not stabilize the system. In fact, at times it will add energy to the system and at others absorb energy from it. This example motivates formally analyzing this system to show that this controller will function essentially as simply another spring between the two extreme masses. The PD controller will suffice in stabilizing the system quickly, and this will be clear from the height of the bar representing the energy in the system.

From the point of view of domain needs, it appears that high-level modeling of controllers as continuous systems does not introduce additional demands on the modeling language beyond that presented above for particle dynamics and composite models. In particular, support for ODEs suffices. More sophisticated controllers can employ different control dynamics in different modes, in which case the full features needed to model hybrid ODEs would be useful.

8 Discretization and Quantization

The one aspect of controllers that we have not captured in the models presented above is that they are generally implemented by digital computers. The introduction of digital components in a physical setting introduces a need for both discretization and quantization. Both notions involve mapping the set of reals into a discrete set, such as the naturals or an isomorphic set. The term 'discretization' refers to performing this type of operation on quantities representing time, whereas 'quantization' refers to performing it on other values. Both effects can be concisely expressed with few additional requirements on the modeling formalism. To model discretization, the key mechanism needed is to define a local clock and to allow actions to be performed or observed only at clock transitions. The following model describes a proportional/integral/differential feedback (PID) controller (like the one presented above) with discretization and quantization effects.

```
model force_controller_pid_d
      (k_p,k_i,k_d,period) =
 initially
  r=(0,0,0), y=(0,0,0), s=(0,0,0),
  u=(0,0,0), t = 0,     t' = 0,
  e=(0,0,0), sensor = (0,0,0)
  r_dot=(0,0,0), y_dot=(0,0,0),
  e_i=(0,0,0), e_i'=(0,0,0)
 always
  t' = 1,
  if (t>period) then
   t = 0,
   sensor = floor(y*10)/10,
   u = k_p*e + k_i*e_i - k_d*e_dot,
  noelse,
  e = r - sensor, e_dot = r_dot - y_dot,
  e_i' = e
```

The variables `t` and its derivative `t'` are used in a manner similar to that performed earlier in this paper to generate an interesting signal for a moving sphere. Here we do two new things with the variable `t`. Firstly, we have a conditional statement based on this variable that waits until (`t > period`). The parameter period models the time it takes the particular microprocessor that implements our controller to produce the new value of the result of the controller. Once the condition is true the first thing we do is reset the counter. The second action is to reset its value to 0 using the statement `t=0` as soon as that condition is true. In addition to this reset, the conditional also allows the equation for the variable `u` in the original model to take effect only for that instant when `t` has surpassed the value of `period`. Because no other definition is given for this value until this event occurs again (at the start of the next period), the value `u` remains constant until that change occurs. With this model, it is easy to illustrate that, as the sampling period goes up, the system that we are trying to control can become unstable. Quantization can be modeled using the `floor` function that quantizes the value of input signal `y` by letting `sensor = floor(y*10)/10` as shown above. Quantization can also be modeled using a conditional statement that updates the value of `sensor` up or down by a fixed amount, based the parameter `quanta`, as shown below. We have one conditional statement that waits until (`sensor + quanta < y`), then increases the value of `sensor` by `quanta`. Similarly, there is another conditional statement that decreases the value of `sensor` by the same amount.

```
...
  if (sensor + quanta < y) then
    sensor+ = sensor + quanta
  else if  (sensor - quanta > y) then
    sensor+ = sensor - quanta
  noelse,
...
```

Simple equations using `if` statements and increment/decrement operations suffice to express the operations of rounding functions. Thus, there is again no additional expressivity here beyond the operators we have already introduced for modeling hybrid systems.

We now consider larger case studies to gain better understanding of the practical expressivity of this core set of language constructs.

9 Case Study I : Quadcopter

A rigid body system consists of a set of solid bodies with well-defined masses and inertias, connected by constraints on distances and/or angles between the solid bodies. The dynamics of many robotic systems can be modeled with reasonable accuracy as rigid body systems. It is widely used for describing road vehicles, gear systems, walking bipeds, etc. In this section, we consider an example of a complex system that can be successfully modeled as a simple rigid body, namely, the quadcopter.

9.1 Background

The quadcopter is a popular mechatronic system with four rotor blades to provide thrust. This robust design has seen use in many UAV applications, such as surveillance, inspection,

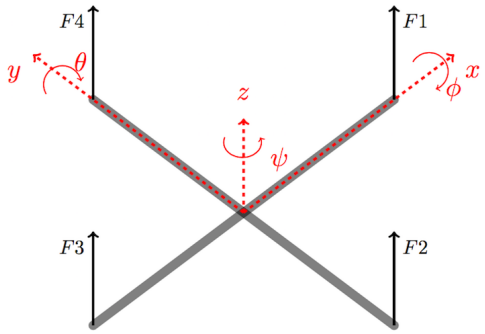


Fig. 4: Free body diagram of the quadcopter.

and search and rescue. Modeling a quadcopter is technically challenging, because it consists of a close combination of different types of physics, including aerodynamics and mechanics. A mathematical model of a quadcopter may need to address a wide range of effects, including gravity, ground effects, aerodynamics, inertial counter torques, friction, and gyroscopic effects.

9.2 Reducing Model Complexity Through Control

Even if we limit ourselves to considering just six degrees of freedom (three for position and three for orientation), the system is underactuated (one actuation from each rotor vs. six degrees of freedom) and is therefore not trivial to control. Fortunately, controllers exist that can ensure that actuation is realized by getting the four rotors to work in pairs, to balance the forces and torques of the system. With this approach, the quadcopter can be usefully modeled as a single rigid body with mass and inertia, by taking account of abstract force, gravity, and actuation control torques. This model is depicted in Fig. 4.

9.3 Mathematical Model

To generate the equations for the dynamics of our common quadcopter model [34], we first construct the rotational matrix to translate from an inertial (globally-fixed) reference frame to the body-fixed reference frame shown in Fig. 4. This matrix represents rotation about the y axis (θ), followed by rotation about the x axis (ϕ), and then rotation about the z axis (ψ).

$$R = \begin{bmatrix} c_\psi c_\theta & c_\psi s_\theta c_\phi - s_\psi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi \\ s_\psi c_\theta & s_\psi s_\theta s_\phi + c_\psi c_\theta & s_\psi s_\theta c_\phi - c_\psi s_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix} \quad (1)$$

Here, c , s and t refer to cos, sin, and tan, respectively. Next, summing forces on the quadcopter results in:

$$\sum F = m\bar{a} = G + RT \quad (2)$$

where G is the force due to gravity, R is the rotational matrix, and T is the thrust from the motors. This expands to:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = -g \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \frac{T}{m} \begin{bmatrix} c_\psi s_\theta c_\phi + s_\psi s_\phi \\ s_\psi s_\theta c_\phi - c_\psi s_\phi \\ c_\theta c_\phi \end{bmatrix} \quad (3)$$

Finally, by summing moments about the center of mass, the equations for the dynamics for each of the rotational degrees of freedom can be determined as follows:

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & \dot{\phi} c_\phi t_\theta + \dot{\theta} \frac{s_\phi}{c_\theta^2} & -\dot{\phi} s_\phi c_\theta + \dot{\theta} \frac{c_\phi}{c_\theta^2} \\ 0 & -\dot{\phi} s_\phi & -\dot{\phi} c_\phi \\ 0 & \dot{\phi} \frac{c_\phi}{c_\theta} + \dot{\phi} s_\phi \frac{t_\theta}{c_\theta} & -\dot{\phi} \frac{s_\phi}{c_\theta} + \dot{\theta} c_\phi \frac{t_\theta}{c_\theta} \end{bmatrix} \nu$$

$$+ W_\eta^{-1} \dot{\nu} \quad (4)$$

Where

$$\nu = \begin{bmatrix} p \\ q \\ r \end{bmatrix} = W_\eta \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -s_\theta \\ 0 & c_\phi & c_\theta s_p hi \\ 0 & -s_\phi & c_\theta c_\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \quad (5)$$

$$\dot{\nu} = \begin{bmatrix} (I_y y - I_z z) \frac{qr}{I_x x} \\ (I_z z - I_x x) \frac{qr}{I_y y} \\ (I_x x - I_y y) \frac{qr}{I_z z} \\ -I_r \end{bmatrix} \begin{bmatrix} \frac{q}{I_x x} \\ \frac{-p}{I_y y} \\ 0 \end{bmatrix} \omega_\Gamma + \begin{bmatrix} \frac{\tau_\phi}{I_x x} \\ \frac{\tau_\theta}{I_y y} \\ \frac{\tau_\psi}{I_z z} \end{bmatrix} \quad (6)$$

$$\begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} lk\left(-\omega_2^2 + \omega_4^2\right) \\ lk\left(-\omega_1^2 + \omega_3^2\right) \\ \sum_{i=1}^{4} \tau_{M_i} \end{bmatrix} \quad (7)$$

9.4 Acumen Model for Quadcopter

The equations derived earlier for the dynamics can be expressed simply in the core language for hybrid systems, and are included with the supplementary materials. Fig. 5 presents snapshots of a 3D visualization of the quadcopter responding to a signal from a basic stabilizing controller [34]. Here the controller is bringing the quadcopter from an initial setting, indicated by the green sphere, to the desired height and to the roll, pitch, and yaw angles of zero. Yellow arrows attached to each rotor indicate the relative thrust. This example shows that the Acumen core language can model the dynamics of non-trivial robots that are widely used in both research and education today.

10 Lagrangian Modeling, and Why we Need it

While the quadcopter case study does not point to the need for additional constructs, working with other case studies does reveal this need. Mathematical modeling of rigid body systems draws heavily on the field of classical mechanics. The goal is to derive the mathematical expression of the system's

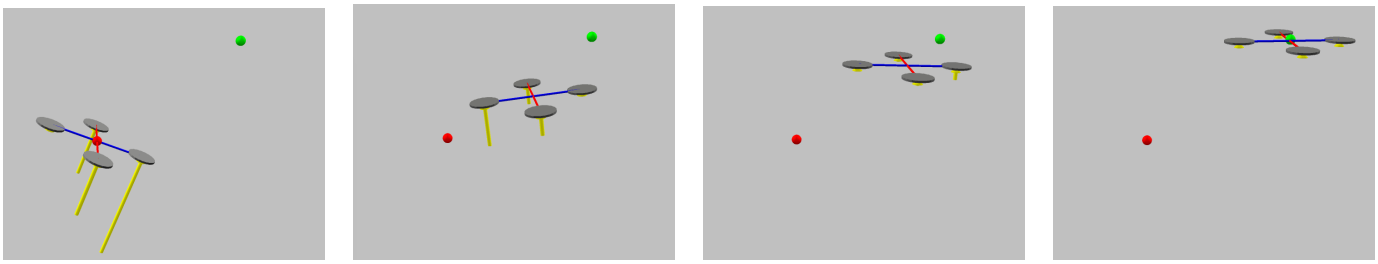


Fig. 5: The simulation results of the quadcopter model with PID control.

dynamics. It so happens that with the quadcopter the Newton method for analysis is convenient. However, as we will see in this section, there are many systems for which other methods are noticeably more convenient.

The Newton method is focused on taking into consideration the forces and torques operating on a rigid body, and then computing the linear and angular accelerations of the center of mass of that body. In general, this method consists of isolating the rigid body of interest in a free body diagram, selecting coordinate frame and summing forces and torques on the body with respect to that frame, then using kinematics to express the linear and angular acceleration terms, before finally deriving the equations for the dynamics. Since the Newton method requires each rigid body to be isolated, forces and torques are modeled explicitly for the interfaces between them in multi-body systems, thus yielding models where forces are readily available for inspection and/or analysis.

It is standard practice for mechanical engineers to use the Lagrangian method to analyze rigid body systems when modeling internal forces between rigid bodies are not the focus of the investigation. However, supporting this method requires constructs beyond the ones discussed so far. The Lagrangian method is based on the notion of a function called *Lagrangian* $L = T - V$, which is the difference between the kinetic energy T and potential energy V. In Lagrangian modeling of physical systems, this condition should be seen as the analogy of the combined conditions $\Sigma F = ma$ and $\Sigma \tau = I\omega''$ in Newtonian mechanics. The Euler-Lagrange equation is as follows:

$$\forall i \in \{1...|q|)\}, \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q_i}}\right) - \frac{\partial L}{\partial q_i} = Q \tag{8}$$

The $\forall$ quantifier is used to introduce the index variable for a family of equations. In the Acumen syntax, the keyword `foreach` represents this quantifier. The name contained in the *i*th element of the tuple is looked up. Q represents non-conservative forces (e.g., friction) acting on the system. Using just this equation, the modeling process is reduced to specifying the kinetic and potential energy in the system. Part of the power of the method comes from the fact that this can be done using Cartesian, polar, spherical, or any other generalized coordinates. The Euler Lagrange equation makes it possible to describe large and complex systems in a modular fashion, since all that is needed for each new object is an expression for both its potential and kinetic energy. For example, this method makes modeling the interactions between electrical and mechanical subsystems much more straightforward than with other methods. Compared to the classic Newtonian, force-vector based methods, the Euler-Lagrange equation can provide a more direct specification of the dynamics, particularly in systems with coupled dynamics. The Lagrangian modeling process consists of four steps:

1) Start with a description of the components of the system, consisting of rigid bodies and joints. This description generally comes with a set of variable names which are collectively called the generalized coordinates vector q. While more commonly associated with Newtonian modeling, a basic *free body diagram* can be an intuitive way to capture the potential and kinetic energy information, as well as assisting in judiciously choosing the generalized coordinates.
2) Determine the expression for the total kinetic and potential energy T and V, respectively, of the system, in terms of q.
3) Identify and include any external forces Q, such as friction.
4) Substitute the values into the Euler-Lagrange equation (8).

This process and its benefits can be illustrated with two small examples. The benefits apply whether or not the language supports directed or undirected equations. Fig. 6 presents a free body diagram marked up with generalized coordinates (θ in one, and θ_1, θ_2 in the other) for a single and a double pendulum system. First, we consider the single pendulum. A direct application of the angular part of Newton's law gives us the following equation:

$$\ddot{\theta} = \frac{g}{l}\cos\theta \tag{9}$$

which can be easily expressed in Acumen.

Lagrangian modeling can be applied to the single pendulum problem, but Newton's method works well enough here. However, Lagrangian modeling does pay off for a double pendulum. It is instructive for language designers to recognize that such a seemingly small change in the complexity of the rigid body makes the model that most of us learn about in high-school much more cumbersome than necessary. Whether or not

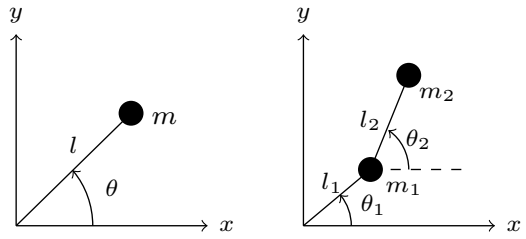


Fig. 6: Diagram for single and double pendulums.

this difficulty in modeling is due to a weakness in Newtonian modeling or intrinsic complexity in this seemingly simple example is not obvious: The double pendulum is sophisticated enough to be widely used to model a human standing or walking [35], or a basic two-link robot such as the MIT-Manus [36].

To derive a model for the double pendulum using Lagrangian modeling, we proceed as follows:

1) A minimal set of generalized coordinates is chosen, $q = (\theta_1, \theta_2)$, for this case it is the angle related to the pose of each link of the pendulum.
2) The kinetic and potential energies are defined as follows:

$$T = \frac{1}{2}m_1 v_1^2 + \frac{1}{2}m_2 v_2^2 \tag{10}$$

$$V = m_1 g y_1 + m_2 g y_2 \tag{11}$$

where we have introduced shorthands for velocities $v_1^2 = l_1^2 \dot{\theta_1}^2$ and $v_2^2 = v_1^2 + \frac{1}{2}m_2(l_1^2\dot{\theta_1}^2 + l_2^2\dot{\theta_2}^2 + 2l_1 l_2 \dot{\theta_1}\dot{\theta_2}\cos(\theta_2 - \theta_1))$ and positions of center of mass $y_1 = l_1 sin\theta_1$, $y_2 = y_1 + l_2 sin\theta_2$. Substituting these terms we get:

$$T = \frac{1}{2}m_1(l_1\dot{\theta_1})^2 + \frac{1}{2}m_2(l_1^2\dot{\theta_1}^2 + l_2^2\dot{\theta_2}^2 + 2l_1 l_2\dot{\theta_1}\dot{\theta_2}\cos(\theta_2 - \theta_1)) \tag{12}$$

$$V = m_1 g l_1 \sin\theta_1 + m_2 g l_2 \sin\theta_2 + m_2 g l_1 \sin\theta_1; \tag{13}$$

3) We assume frictionless joints, and so there are no external forces ($Q = 0$).
4) A Euler-Lagrange equation (8) is written in Acumen syntax.

The equations derived earlier for the dynamics can be expressed in our core language as follows:

```
model double_pendulum(m_1,m_2,L_1,L_2,g)=
initially
  t_1   = 0, t_2   = 0,
  t_1'  = 0, t_2'  = 0,
  t_1'' = 0, t_2'' = 0,
  q = (0,0)
always
  q = (t_1, t_2),
  T = 0.5*m_1*(l_1*t_1')^2 +
      0.5*m_2*(l_1^2*t_1'^2 +
      l_2^2*t_2'^2 +
      2*l_1*l_2*t_1'*t_2'*cos(t_2-t_1)),
  V = m_1*g*l_1*sin(t_1) +
        m_2*g*l_2*sin(t_2) +
        m_2*g*l_1*sin(t_1),
  L = T - V,
  foreach i in length(q)
      L'[(q(i))']' - L'[q(i)] = 0
```

Here `L'[q(i)]` is the Acumen syntax for partial derivatives of $\frac{\partial L}{\partial q_i}$ and `L'[(q(i))']` is the syntax for $\frac{d}{dt}\left(\frac{\partial L}{\partial q_i}\right)$.

To illustrate why supporting partial derivatives and equations in the language can make the modeling process easier, let us consider the same double pendulum model in MATLAB and OpenModelica. As already shown in Section 3, MATLAB does not support writing equations directly. In order to use the ODE solver it provides, one has to manually differentiate the Euler-Lagrange equations and transform them into the explicit ODE form. In OpenModelica, writing equations directly is supported, however, no partial derivatives can appear in the equations. Thus, it is the responsibility of the user to eliminate the partial derivatives and transform the Euler-Lagrangian equations into differentiation algebraic equation from. The complete model in MATLAB and OpenModelica can be seen in Supplemental Material (Section 14).

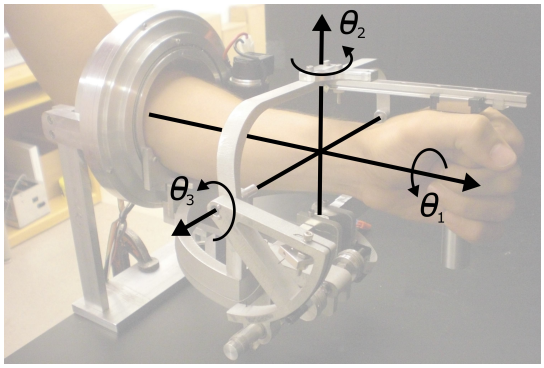


Fig. 7: The RiceWrist-S, with superimposed axes of rotation.

11 Case Study II : The RiceWrist-S robot

Engineers utilize Lagrangian modeling in the manner presented above to model multi-link robots much more directly than with the Newtonian method. In this section we present one such case study, using the RiceWrist-S research robot.

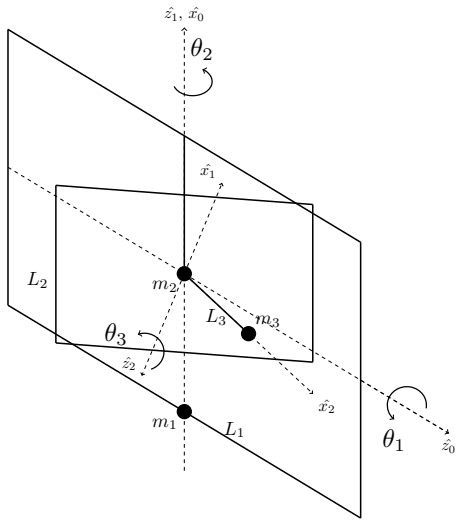


Fig. 8: Diagram of the RiceWrist-S as a gimbal

11.1 Background

Each year, approximately 795,000 people suffer a stroke in the United States, where stroke injuries are the leading cause of long-term disability. The RiceWrist-S is a research robot designed to assist in the rehabilitation of the wrist and forearm after neurological injuries such as stroke (Fig. 7). It consists of a revolute joint for each of the three degrees of freedom at the wrist. A good starting point for its dynamic modeling is the gimbal, a commonly studied mechanical device that, like the RiceWrist-S, features several rotational axes intersecting at one point.

11.2 Analytical Model

We can apply the Lagrangian modeling process to determine the dynamics of a gimbal as follows:

1) We take $q = (\theta_1, \theta_2, \theta_3)$, where each of the angles corresponds to one of the three rotations possible in the RiceWrist-S (Fig. 8). We choose to represent the mass of the system as centralized to three locations, one at the origin, one at the bottom of the outermost ring, and one at the end of the third link. The masses in this figure correspond to the motors and handle depicted in Fig. 7.
2) To describe the energies concisely, it is convenient to use the following angular velocities of the gimbal frames in the kinetic energy terms, and the resulting heights for the potential energy terms:

$$\omega_1 = \dot{\theta_1} \cdot \hat{z_0} \tag{14}$$
$$\omega_2 = \dot{\theta_1} \cdot \hat{z_0} + \dot{\theta_2} \cdot \hat{z_1} \tag{15}$$
$$\omega_3 = \dot{\theta_1} \cdot \hat{x_1} + \dot{\theta_2} \cdot \hat{z_1} + \dot{\theta_3} \cdot \hat{z_2} \tag{16}$$

where $\hat{x_i}\hat{y_i}\hat{z_i}$ refers to the unit vector and coordinate frame about which these rotations occur, as shown in Fig. 8.

TABLE 2: Denavit Hartenberg parameters.

Joint	**rot**(x)	**tr**(x)	**rot**(z)	**tr**(z)
Forearm	0	0	θ_1	0
Wrist F/E	$-\frac{\pi}{2}$	0	θ_2	0
Wrist R/U	$-\frac{\pi}{2}$	0	θ_2	0

Here, the ω_i terms correspond to the m_i masses, and describe the angular velocities of each mass. Since this is a rotational system, many of the rotations do not occur in the coordinate frames of the respective gimbal. Therefore, in order to express each ω_i in terms of the same coordinate frame, we apply the following coordinate transforms, which follow the Denavit-Hartenberg convention:

$$\omega_1 = \dot{\theta_1} \cdot \hat{z_0} \tag{17}$$
$$\omega_2 = \dot{\theta_1} \cdot \hat{z_0} + T_1^0 \cdot \dot{\theta_2} \cdot \hat{z_1} \tag{18}$$
$$\omega_3 = \dot{\theta_1} \cdot \hat{z_0} + T_1^0 \cdot \dot{\theta_2} \cdot \hat{z_1} + T_1^0 T_2^1 \cdot \dot{\theta_3} \cdot \hat{z_2} \tag{19}$$

where the elements of T_i^{i-1} are given in Table 2 and each coordinate rotation is defined as:

$$T_i^{i-1} = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} c_{\alpha_i} & s_{\theta_i} s_{\alpha_i} \\ s_{\theta_i} & c_{\theta_i} c_{\alpha_i} & -c_{\theta_i} s_{\alpha_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} \end{bmatrix} \tag{20}$$

Note that these coordinate transforms are essential for the standard development of multi-link robot kinematics. Since the gimbal's frames are all coincident, only the rotation about the individual x and z frames, α_i and θ_i, respectively, can be non-zero. Next, we express the heights above the predefined plane of zero potential energy (in Fig. 8, the plane perpendicular to x_0) of each of the masses m_1, m_2, m_3, respectively, as the following:

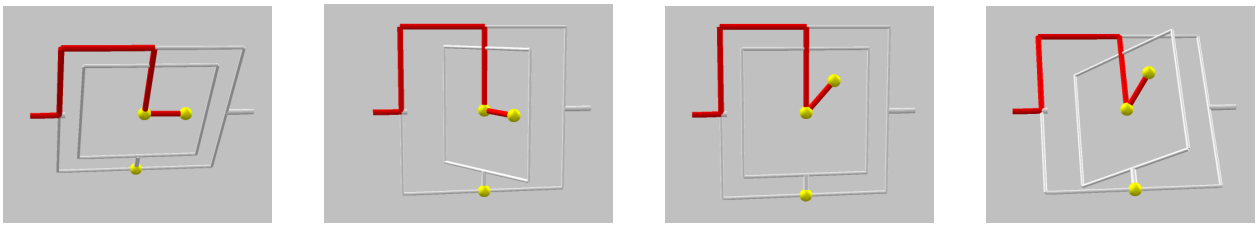
$$h_1 = l_2(1 - \cos(\theta_1)) \tag{21}$$
$$h_2 = 0 \tag{22}$$
$$h_3 = -l_3 \cos(\theta_1) \sin(\theta_3) \tag{23}$$

With this completed, the T and V terms can be quickly and easily defined. Since this is a purely rotational-only system, T is defined as the sum of the rotational energy terms, shown below:

$$T = \frac{1}{2}(I_1 \omega_1 \cdot \ \omega_1 + I_2 \omega_2 \cdot \omega_2 + I_3 \omega_3 \cdot \omega_3) \tag{24}$$

where I_i is the rotational inertia corresponding to θ_i, and ω_i is defined as above. Since there are no potential energy storage elements other than those caused by gravity, V can be expressed with these heights:

$$V = m_1 g h_1 + m_2 g h_2 + m_3 g h_3$$
$$= m_1 g l_2 (1 - \cos(\theta_1)) - m_3 g l_3 \cos(\theta_1) \sin(\theta_3) \tag{25}$$



(a) Rotation about θ_1 (b) Rotation about θ_2 (c) Rotation about θ_3 (d) Compound motion

Fig. 9: RiceWrist-S modeled as gimbal in Acumen.

3) Again, we assume frictionless joints, so $Q = 0$. In this model, we neglect electrical motor dynamics, due to the prevalence of high-performance motor drivers with fast or negligible dynamics.
4) Substitute the values into the Euler-Lagrange equation.

The Euler-Lagrangian model above can be expressed in Acumen as follows:

```
model RWS_3DOF_gimbal
  (m1,m2,m3,I1,I2,I3,l1,l2,l3) =
initially
 t1  =  0, t2  =  0, t3  =  0,
 t1' =  0, t2' =  0, t3' =  0,
 t1''=  0, t2''=  0, t3''=  0,
 g = 9.81, q = zeros(3),
 T  =  0,  V  =  0, L = 0,
 T01 = zeros(3,3), T12 = zeros(3,3),
 w1 = zeros(3), w2 = zeros(3),
 w3 = zeros(3)
always
 m1 = 2, m2 = 1, m3 = 1, I1 = 1,
 I2 = 1, I3 = 1, l1 = 3, l2 = 1.5,
 q = (t1,t2,t3), g = 9.81, l3 = 0.75,
 T01 = ((-cos(t2),-sin(t2), 0),
        (0,       0,       1),
        (-sin(t2), -cos(t2), 0)),
 T12 =  ((-cos(t3),-sin(t3), 0),
        (0,        0,      -1),
        (sin(t3),  cos(t3), 0)),
 w1 = (0,0,t1'), w2 = w1 + (0,0,t2') * T01,
 w3 = w2 + (0,0,t3') * (T01 * T12),
 T = 0.5 * (I1 * dot(w1,w1) +
            I2 * dot(w2,w2) +
            I3 * dot(w3,w3)),
 V=  m1*g*l2*(1- cos(t1))-
     m3*g*l3*sin(t1)*sin(t3),
 L = T - V,
 foreach i in length(q)
  L'[(q(i))']' - L'[q(i)]= 0
```

Fig. 9 shows the compound motion of the RW-S Gimbal model. This model led us to understand the need for Lagrangian modeling and, as a result, provides us with concrete justification for introducing support for static partial derivatives and undirected equations. Supporting static partial derivatives and Euler-Lagrangian equations allows the modeler to specify the dynamics directly instead of performing symbolic differentiation and equation solving (acasual to casual transformation) manually.

12 Case Study III: Electromechanical Systems.

In this section we present a case study of an integrated electromechanical system, the basics of which are integral for fields such as robotics.

12.1 Background

Simple electromechanical systems like the solenoid attached to a mechanical spring and damper, shown in Fig. 10, exhibit the coupled dynamics of much more complicated systems. They can demonstrate the core language's ability to perform multiphysics simulations, such as coupled mechanical and electrical subsystems.

This system in Fig. 10 is powered by an input voltage $e(t)$ that drives a solenoid with a core of mass m and inductance $L(x)$, which is the inductance of the solenoid given as a function of the core's position. The system also has an electrical resistance R, coupled to a linear stiffness and damping of k and β, respectively, constraining its movement to be in only one direction: perpendicular to gravity.

12.2 Analytical Model

To derive a model for the system, we proceed as follows:

1) Choose generalized coordinates to be the electrical charge of the circuit and the position of the solenoid's core, that is, $q_1 = q$, $q_2 = x$.
2) The kinetic energy term is the sum of the field energy stored in the inductor and the velocity of the solenoid's mass, defined as follows:

$$T = \frac{1}{2}L(x)\dot{q_1}^2 + \frac{1}{2}m\dot{q_2}^2 \tag{26}$$

Since the solenoid travels in a direction perpendicular to gravity, and neglecting any capacitance of the electrical

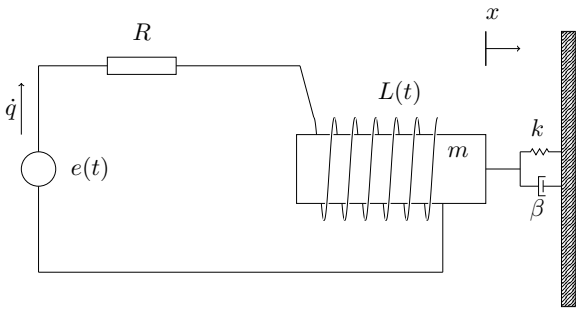


Fig. 10: A simple coupled electromechanical system

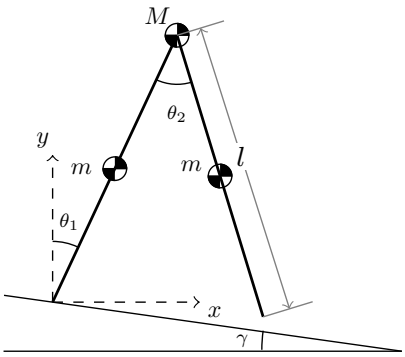


Fig. 11: A Compass Gait Biped [41].

circuit, the potential energy term is simply the mechanical energy stored in the spring, defined as:

$$V = \frac{1}{2}kq_2^2 \tag{27}$$

3) The dissipation energy term of the electrical resistance and mechanical damping can be included in the external forces:

$$Q = -\frac{\partial D}{\partial \dot{q}} + e(t) \tag{28}$$

where D is defined as

$$D = \frac{1}{2}R\dot{q_1}^2 + \frac{1}{2}\beta\dot{q_2}^2 \tag{29}$$

4) Write the Euler-Lagrange equation (8) in Acumen syntax.

While this example seems straightforward, it does show one important benefit of using the Euler-Lagrange method, namely the development of coupled dynamics expressions. In this case, the coupled terms are the force exerted on the solenoid's mass by the magnetic field of the inductor, and the induced voltage due to the translation x, which can be seen to be:

$$F = \frac{1}{2}\frac{\partial L}{\partial q_2}\dot{q_1}^2 \tag{30}$$

and

$$V = \frac{1}{2}\frac{\partial L}{\partial q_2}q_1 q_2 \tag{31}$$

If we use Newton's method, these terms would need to be known so that we can create the free body diagrams.

13 Case Study IV : A Compass Gait Biped

McGeer first introduced the concept of passive dynamic walking (PDW) in 1990 [37], shedding light on achieving stable human-like locomotion without any external forces. A number of biped robots have been developed following this model, including the Cornell biped [38] and Amber [39], [40], demonstrating great success in the domain of robotic walking. Hence it is worthwhile to investigate how well the core formalism that we have identified in this paper can model such systems.

The compass gait biped model is a two-dimensional unactuated rigid body system placed on a downward surface, inclined with a fixed angle γ from the horizontal plane. A diagram of the model is shown in Fig. 11 with its physical parameters. The configuration of this two-link mechanism can be described by the generalized coordinates $q = [\theta_1, \theta_2]$, where θ_1 is the angle from the vertical line to the *stance leg*, and θ_2 is the angle between the two legs. It is a hybrid model featuring two phases. At the start of each step, the system is governed by its continuous dynamics until the *swing leg* hits the ground. The discrete event can be modeled as an inelastic collision conserving angular momentum. The stance and swing legs switch instantaneously during the collision and go into the next step after.

13.1 Continuous Dynamics

The continuous phase of this system can be modeled using the same Lagrange method shown earlier. Let point (x_i, y_i) denote the position of centralized masses shown in Fig. 11, from which it is easy to define the kinetic and potential energy of the system as follows:

$$T = \frac{1}{2}m_1(\dot{x_1}^2 + \dot{y_1}^2 + \dot{x_2}^2 + \dot{y_2}^2 + \dot{x_3}^2 + \dot{y_3}^2)$$

$$V = m_1 g(y_1 + y_3) + m_2 g y_2$$

Applying the same Lagrange equation shown in Equation 1 with $q = (\theta_1, \theta_2)$, we have the dynamic equations of the system during the swing phase.

13.2 Discrete Events

When the swing foot impacts the surface of the slope, a discrete event is triggered to update the angular velocities, in order to prevent the biped from falling through the floor. The

perpendicular distance from the walking surface to the tip of the *swing leg* is given by

$$guard = lsin\gamma(sin\theta_1 + sin(\theta_2 - \theta_1))$$

where γ is the slope of the ground. Impact occurs when the tip of the swing leg hits the walking surface in a downward direction, which can be described as follows:

$$guard \leq 0 \wedge \dot{guard} < 0$$

Using conservation of angular momentum [41], the explicit solution of post-impact velocities is as follows:

$$H_1 = \begin{bmatrix} m_1 l^2(\frac{5}{4} - \frac{cos\theta_2^-}{2}) + m_2 l^2 & \frac{m_1}{4} l^2(1 - 2cos\theta_2^-) \\ \frac{m_1}{2} l^2 cos\theta_2^- & \frac{m_1}{4} l^2 \end{bmatrix}$$

$$H_2 = \begin{bmatrix} -\frac{m_1}{4} l^2 + (m_2 l^2 + m1l^2)cos\theta_2^- & -\frac{m_1}{4} l^2 \\ -\frac{m_1}{4} l^2 & 0 \end{bmatrix}$$

$$\theta_1^+ += \theta_2^- - \theta_1^- \quad \theta_2^+ += -\theta_2^-$$

$$[\dot{\theta_1}^+, \dot{\theta_2}^+]^T = H_1^{-1} \cdot H_2 \cdot [\dot{\theta_1}^-, \dot{\theta_2}^-]^T$$

Fig. 12 in the Supplementary Material section shows the compass gait biped mathematical model and the Acumen program. This example shows that the core language can support a direct mapping from a mathematical model to simulation code for a hybrid system model with complex dynamics.

Thus, this case study suggests that a a formalism supporting hybrid ODEs and partial derivatives suffices to model the dynamics of non-trivial robot rigid body dynamics.

14 Conclusion and Future work

In this paper we have identified a number of aspects of cyber-physical systems that are arguably pervasive, and have identified the requirements that these aspects place on a modeling language. These requirements were checked against a small collection of language constructs that were found to be a reasonable match, both when considering minimal examples and larger examples. Many of these constructs are supported in existing modeling languages, although usually not in exactly the same form, and not with the same degree of minimalism. For example, while the Acumen language is minimal, it can express many examples more concisely than mainstream tools such as MATLAB and Modelica. Furthermore, there are important constructs in such modeling languages, such as partial derivatives, that do not currently enjoy sufficient support in most languages considered in the paper. One example of particular importance from the point of view of classical mechanics is partial derivatives, which are essential for Lagrangian modeling. The only language that appears to have the necessary support is 20-sim.

The study illustrates clearly how a small language suffices to address the needs of several aspects of cyber-physical systems, including visualization, geometry, particle dynamics and impact, control, quantization and discretization, and composition. It also illustrates that it suffices for expressing rather concise models of several interesting subsystems that arise in real robot design projects. The analysis of these aspects is thorough in that it used an implementation that supports the language constructs described, and all the examples discussed were expressed fully in this language. The models were also tested to check their soundness.

The collection of aspects identified and studied in this work is not intended to be exhaustive by any means. Most notably, it is not based on an exhaustive analysis of the needs of modeling all types of physical phenomena, nor is it based on case studies in modeling CPSs. Rather, it is intended to illustrate a minimal set of requirements from this domain on a modeling and simulation language, and is a starting point for further studies on the linguistic needs of multi-physics and large-scale modeling.

In future work, we would like to further develop the set of basic aspects in several directions. For example, we would like to explore the possibility of supporting different coordinate systems (especially relative coordinate frames) to facilitate modeling composite mechanical systems. A key question in this respect is whether special language support may be needed, or whether developing appropriate modeling patterns will suffice. Similarly, we would like to study the needs of more sophisticated types of control, including parameter tuning, optimal control, and model-based control. We expect that providing support for minimization of values and functions will be key questions in this respect. At this time it is not obvious whether such support requires special language extensions or whether finding suitable modeling patterns and libraries would suffice. Similar questions apply for path and trajectory planning. The current prototype of the modeling language focuses on creating a 'minimum viable prototype' to enable research such as that reported on here. It is an interpreter-based prototype and not tuned for performance. It is sufficient for reasonably accurate near real-time simulation of the models presented in this paper. However, for larger models, it is important to improve the efficiency of the implementation Finally, we would like to explore the space of physically-inspired modularity, for example, to support physical connectivity network structure and/or abstraction constructs that can facilitate the development of larger (composite) models in a way that mimic physical connectivity.

Supplemental Material

This section contains additional codes referred to in the paper. The complete Acumen model of the quadcopter is as follows:

```
model QuadCopter(P,phi,theta,psi) =
initially
  g = 9.81,      m = 0.468,
  l = 0.225,     k = 2.98*10^(-6),
  b = 1.140*10^(-7),
  IM = 3.357*10^(-5),
```

```
  Ixx = 4.856*10^(-3),
  Iyy = 4.856*10^(-3),
  Izz = 8.801*10^(-3),
  Ax  = 0.25,  Ay  = 0.25,
  Az  = 0.25,
  w1 = 0,  w2= 0,   w3 = 0,
  w4 = 0,  wT = 0,  f1 = 0,
  f2 = 0,  f3 = 0,  f4 = 0,
  TM1 = 0, TM2 = 0, T = 0,
  TM3 = 0, TM4 = 0,
  P' = (0,0,0),    P'' = (0,0,0),
  phi' = 0, theta' = 0, psi' = 0,
  phi'' = 0, theta'' = 0, psi'' = 0,
  p = 0, q = 0, r = 0, p' = 0,
  q' = 0, r' = 0, Ch=0, Sh=0,
  Sp=0, Cp=0, St=0, Ct=0, Tt=0
always
  T = k* (w1^2 + w2^2 + w3^2 + w4^2),
  f1 = k * w1^2, TM1 = b * w1^2,
  f2 = k * w2^2, TM2 = b * w2^2,
  f3 = k * w3^2, TM3 = b * w3^2,
  f4 = k * w4^2, TM4 = b * w4^2,
  wT = w1 - w2 + w3 - w4,

  Ch = cos(phi), Sh = sin(phi),
  Sp = sin(psi), Cp = cos(psi),
  St = sin(theta), Ct = cos(theta),
  Tt = tan(theta),

  P'' = -g * (0,0,1) + T/m
   *(Cp*St*Ch+Sp*Sh,Sp*St*Ch-Cp*Sh,Ct*Ch)
   -1/m*(Ax*P'(0),
         Ay*P'(1),
         Az*P'(2)),
  p' = (Iyy-Izz)*q*r/Ixx - IM*q/Ixx*wT
   + l*k*(w4^2 - w2^2)/Ixx,
  q' = (Izz-Ixx)*p*r/Iyy-IM*(-p)/Iyy*wT
   + l*k*(w3^2 -w1^2)/Iyy,
  r' = (Ixx - Iyy)*p*q/Izz  + b*(w1^2
   + w2^2 -w3^2 -w4^2)/Izz,
  phi'' = (phi'*Ch*Tt+ theta'*Sh/Ct^2)*q
   + (-phi'*Sh*Ct+theta'*Ch/Ct^2)*r
   + (p'+q'*Sh*Tt+r'*Ch*Tt),
  theta'' = (-phi'*Sh)*q + (-phi'*Ch)*r
   + (q'*Ch+r'*(-Sh)),
  psi'' = (phi'*Ch/Ct+phi'*Sh*Tt/Ct)*q
   + (-phi'*Sh/Ct+theta'*Ch*Tt/Ct)*r
   + (q'*Sh/Ct+r'*Ch/Ct),
```

```
% MATLAB model
theta_1 = pi; der_theta_1 = 0;theta_2 = pi;
der_theta_2 = 5; g = 9.81; m1 = 1; m2 = 1;
l1 = 1; l2  = 1; L = 0;Tend = 20; fps = 10;
ivp=[theta_1; der_theta_1; theta_2;
der_theta_2; g; m1; m2; l1; l2];
nsteps=Tend*fps;
sol=ode23(@ODEs,[0 Tend], ivp);
t = linspace(0,Tend,nsteps);

function xdot = ODEs(t,x)
g=x(5); m1=x(6); m2=x(7); l1=x(8); l2=x(9);
xdot=zeros(9,1);
xdot(1)=x(2);
xdot(2)=-((g*(2*m1+m2)*sin(x(1))+
 m2*(g*sin(x(1)-2*x(3))+2*(l2*x(4)^2+
 l1*x(2)^2*cos(x(1)-x(3)))*sin(x(1)-
 x(3))))/(2*l1*(m1+m2-m2*cos(x(1)
 -x(3))^2)));
xdot(3)=x(4);
xdot(4)=(((m1+m2)*(l1*x(2)^2+g*cos(x(1)))+
 l2*m2*x(4)^2*cos(x(1)-x(3)))*
 sin(x(1)-x(3)))/(l2*(m1+m2-m2*
 cos(x(1)-x(3))^2));
```

`xdot(2)` and `xdot(2)` are the second order derivatives of the two angles.

```
// OpenModelica model
model doublependulum
  type angle = Real(unit = "rad");
  parameter Real m1 = 1; parameter Real m2 = 1;

  parameter Real l1 = 1;
  parameter Real l2 = 1;
  parameter Real g = 9.81;
  parameter Real pi = 3.14;
  parameter angle t10 = pi;
  parameter angle t20 = pi;
  parameter angle t1prime0 = 0.0;
  parameter angle t2prime0 = 5.0;
  angle t1; angle t1prime;
  angle t2; angle t2prime;
initial equation
  t1 = t10;
  t2 = t20;
  t1prime = t1prime0;
  t2prime = t2prime0;
equation
  der(t1) = t1prime;
  2*l1*(m1+m2-m2*cos(t1-t2)^2)*der(t1prime) =
  -(g*(2*m1+m2)*sin(t1)+
  m2*(g*sin(t1-2*t2)+
  2*(l2*t2prime^2+
  l1*t1prime^2*cos(t1-t2))*sin(t1 - t2)));
  der(t2) = t2prime;
  l2*(m1+m2-m2*cos(t1-t2)^2)*der(t2prime=
  ((m1+m2)*(l1*t1prime^2+g*cos(t1)) +
  l2*m2*t2prime^2*cos(t1-t2))*sin(t1 - t2);
end doublependulum;
```

$q = [\theta_1, \theta_2] \quad m_1 = 1 \quad m_2 = 2 \quad l = 1$	`q = (t1,t2), m1 = 1, l = 1, m2 =2,`
$\gamma = 0.044 \quad g = 9.8$	`r = 0.044, g = 9.8,`
$x_1 = \frac{1}{2} l sin\theta_1 \quad y_1 = \frac{1}{2} l cos\theta_1$	`x1 = l/2*sin(t1),y1 =l/2*cos(t1),`
$x_2 = l sin\theta_2 \quad y_2 = l cos\theta_2$	`x2 = l*sin(t2), y2 = l*cos(t2),`
$x_3 = x_2 + \frac{l}{2} sin(\theta_2 - \theta_1) \quad y_3 = y_2 - \frac{l}{2} cos(\theta_2 - \theta_1)$	`x3 = x2+l/2*sin(t2-t1), y3 = y2-l/2*cos(t2-t1),`
$L = T - V \quad guard = l sin\gamma(sin\theta_1 + sin(\theta_2 - \theta_1))$	`L=T-V, guard=l*sin(r)*(sin(t1)+sin(t2-t1)),`
$T = \frac{1}{2} m_1 (\dot{x_1}^2 + \dot{y_1}^2 + \dot{x_3}^2 + \dot{y_3}^2)$ $+\frac{1}{2} m_2 (\dot{x_2}^2 + \dot{y_2}^2)$	`T = 1/2*m1*((x1)'^2+(y1)'^2+(x3)'^2+(y3)'^2)` `+ 1/2*m2*((x2)'^2 + (y2)'^2),`
$V = m_1 g(y_1 + y_3) + m_2 g y_2$	`V = m1*g*(y1+y3)+m2*g*y2,`
$H = H_1^{-1} \cdot H_2 \cdot [\dot{\theta_1}^-, \dot{\theta_2}^-]^T$	`H = inv(H1)*H2*trans((t1',t2')),`
$\forall i \in \{1...\|q\|)\}. \frac{d}{dt} \left(\frac{\partial L}{\partial \dot{q_i}} \right) - \frac{\partial L}{\partial q_i} = 0$	`foreach i in length(q)` `L'[(q(i))']' - L'[q(i)] = 0 ,`
$H_1 =$ $[m_1 l^2 (\frac{5}{4} - \frac{cos\theta_2^-}{2}) + m_2 l^2 \quad \frac{m_1}{4} l^2 (1 - 2cos\theta_2^-)$ $\frac{m_1}{2} l^2 cos\theta_2^- \quad \frac{m_1}{4} l^2]$	`H1 =` `((m1*l^2*((5/4-cos(t2)/2)+m2*l^2),` `m1/4*l^2*(1-2*cos(t2))),` `( m1/2*l^2*cos(t2), m1/4*l^2)),`
$H_2 = [-\frac{m_1}{4} l^2 + (m_2 l^2 + m1 l^2) cos\theta_2^- \quad -\frac{m_1}{4} l^2$ $\frac{m_1}{4} l^2 \quad 0\ ]$	`H2 =` `((-m1/4*l^2+(m2*l^2+m1*l^2)*cos(t2), -m1/4*l^2)` `(m1/4*l^2, 0),`
if $guard < 0 \wedge \dot{guard} < 0$ then $\theta_1^+ = \theta_2^- - \theta_1^- \quad \theta_2^+ = -\theta_2^-$ $\dot{\theta_1}^+ = H(0) \quad \dot{\theta_2}^+ = H(1)$	`if guard <0 && (guard)' then` `t1 += t2 - t1, t2 += -t2,` `t1' += H(0), t2' += H(1) noelse`

Fig. 12: Compass Gait Biped in Mathematical Notation and in Acumen Syntax

The mathematical model and Acumen model of the compass gait biped is presented in Fig 12.

Acknowledgments

We would like to thank the reviewers of DSLRob 2012 DSLRob 2013 and ICESS 2014 for comments and feedback on earlier versions of this manuscript.

References

[1] W. Taha and R. Philippsen, "Modeling basic aspects of cyber-physical systems," *arXiv preprint arXiv:1303.2792*, 2012. (document)

[2] Y. Zeng, C. Rose, P. Brauner, W. Taha, J. Masood, R. Philippsen, M. O'Malley, and R. Cartwright, "Modeling basic aspects of cyber-physical systems, part ii," *arXiv preprint arXiv:1312.5952*, 2013. (document)

[3] Y. Zeng, C. Rose, P. Brauner, W. Taha, J. Masood, R. Philippsen, M. OMalley, and R. Cartwright, "Modeling basic aspects of cyber-physical systems, part ii," in *The 11th IEEE International Conference on Embedded Software and Systems*, 2014, pp. 550–557. (document)

[4] J. Masood, R. Philippsen, J. Duracz, W. Taha, and H. Eriksson, "A case study on design-time verification of automatic emergency breaking," in *International Federation of Automotive Engineering Societies 2014 World Automotive Congress*, 2014. 1

[5] A. Duracz, H. Eriksson, F. Á. Bartha, Y. Zeng, F. Xu, and W. Taha, "Using rigorous simulation to support iso 26262 hazard analysis and risk

assessment," in *The 12th IEEE International Conference on Embedded Software and Systems*, 2015. 1

[6] J. Bruneau, C. Consel, M. O'Malley, W. Taha, and W. M. Hannourah, "Virtual Testing for Smart Buildings," in *Proceedings of the 8th International Conference on Intelligent Environments (IE'12)*, Guanajuato's, Mexico, 2012. 1

[7] L. P. Carloni, R. Passerone, A. Pinto, and A. L. Sangiovanni-Vincentelli, *Languages and Tools for Hybrid Systems Design*. now Publishers Inc, 2006. 1, 3

[8] J. Jensen, D. Chang, and E. Lee, "A Model-Based Design Methodology for Cyber-Physical Systems," in *Proceedings of The First IEEE Workshop on Design, Modeling and Evaluation of Cyber Physical Systems (CyPhy'11)*, Istanbul, Turkey, Jul. 2011. 1

[9] E. Allen, D. Chase, J. Hallett, V. Luchangco, J. Maessen, S. Ryu, G. L. Steele Jr., and S. Tobin-Hochstadt., "The Fortress Language Specification," Technical report, Sun Microsystems, Inc., 2007. 1

[10] P. Fritzson and P. Bunus, "Modelica A General Object-Oriented Language for Continuous and Discrete-Event System Modeling and Simulation," in *SS '02: Proceedings of the 35th Annual Simulation Symposium*. Washington, D.C., USA: IEEE Computer Society, 2002, p. 365. 1

[11] Simscape Language Guide, http://se.mathworks.com/help/physmod/simscape, 2015. 1, 1, 3

[12] Y. Zhu, W. Edwin, J. Inoue, A. Chapoutot, C. Salama, M. Peralta, T. M. s, W. Taha, M. O'Malley, and R. Cartwright, "Mathematical equations as executable models of mechanical systems," in *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*. ACM, 2010, pp. 1–11. 1, 3

[13] M. Konecny, W. Taha, J. Duracz, A. Duracz, and A. Ames, "Enclosing the behavior of a hybrid system up to and beyond a zeno point," in *Cyber-Physical Systems, Networks, and Applications (CPSNA), 2013 IEEE 1st International Conference on*. IEEE, 2013, pp. 120–125. 1

[14] M. Konečnỳ, J. Duracz, A. Farjudian, and W. Taha, "Picard method for enclosing odes with uncertain initial values," in *11th International Conference on Computability and Complexity in Analysis, Darmstadt, Germany, July 21-24, 2014*, 2014, pp. 41–42. 1

[15] J. Duracz, A. Farjudian, M. Konečnỳ, and W. Taha, "Function interval arithmetic," in *Mathematical Software–ICMS 2014*. Springer, 2014, pp. 677–684. 1

[16] M. Mohaqeqi, M. R. Mousavi, and W. Taha, "Conformance testing of cyber-physical systems: A comparative study," in *The 14th International Workshop on Automated Verification of Critical Systems, University of Twente, Enschede, Netherlands, 24–26th September, 2014*. European Association of Software Science and Technology, 2014. 1

[17] W. Taha and R. Cartwright, "Some challenges for model-based simulation?" in *The 4th Analytic Virtual Integration of Cyber-Physical Systems Workshop, Vancouver, Canada, December 3, 2013*. Linköping University Electronic Press, 2013, pp. 1–4. 1

[18] A. U. Pehlivan, F. Sergi, A. Erwin, N. Yozbatiran, G. E. Francisco, and M. K. O'Malley, "Design and validation of the ricewrist-s exoskeleton for robotic rehabilitation after incomplete spinal cord injury," *Robotica*, vol. FirstView, pp. 1–17, 7 2014. [Online]. Available: http://journals.cambridge.org/article_S0263574714001490 1.1

[19] MATLAB Documentation, http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.html, 2015. 1

[20] Octave Documentation, https://www.gnu.org/software/octave/support.html, 2015. 1, 3

[21] OpenModelica User Guide, https://www.openmodelica.org/useresresources/userdocumentation, 12 2015. 1, 3

[22] H. Elmqvist, D. Brück, and M. Otter, "Dymola-user's manual," *Dynasim AB, Research Park Ideon, Lund, Sweden*, 2004. 1

[23] Mathematica Documentation, http://reference.wolfram.com/mathematica/guide/Mathematica.html, 2009. 1, 3

[24] Reference Manual 20-sim 4.6, http://www.20sim.com/downloads/files/20simReference46.pdf, 12 2015. 1, 3

[25] W. Taha, P. Brauner, Y. Zeng, R. Cartrwright, V. Gaspes, A. Ames, and A. Chapoutot, "A Core Language for Executable Models of Cyber Physical Systems (Preliminary Report)," in *Proceedings of The Second International Workshop on Cyber-Physical Networking Systems (CPNS'12)*, Macau, China, Jun. 2012. 2

[26] Acumen website, www.acumen-language.org, 2010. 2

[27] W. Taha, "Lecture notes on cyber-physical systems," Available online from www.effective-modeling.org/p/teaching.html, Sep. 2012. 2

[28] W. Taha, R. Cartwright, R. Philippsen, and Y. Zeng, "A First Course on Cyber Physical Systems," in *2013 Workshop on Embedded and Cyber-Physical Systems Education (WESE)*, Montreal, Canada, October 2013. 2

[29] W. Taha, R. Cartwright, R. Philippsen and Y. Zeng, "A first course on cyber physical systems," in *Proceedings of the First Workshop on Cyber-Physical Systems Education (CPS-Ed 2013) at Cyber Physical Systems Week (CPSWeek 2013)*, Philadelphia, Pennsylvania, USA, April 2013. 2

[30] P. Fritzson, *Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach*. John Wiley & Sons, 2014. 3

[31] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5. 3

[32] Y. Zeng, "Lightweight three-dimensional visualization for hybrid systems simulation," Master's thesis, Halmstad University, Halmstad, 2012. 4

[33] "jPCT 3D engine," http://www.jpct.net. 4

[34] T. Luukkonen, "Modelling and control of quadcopter," 2011, aalto University, Espoo. 9.3, 9.4

[35] D. Pekarek, A. D. Ames, and J. E. Marsden, "Discrete mechanics and optimal control applied to the compass gait biped," in *Decision and Control, 2007 46th IEEE Conference on*. IEEE, 2007, pp. 5376–5382. 10

[36] N. Hogan, H. I. Krebs, J. Charnnarong, P. Srikrishna, and A. Sharon, "MIT-MANUS: a workstation for manual therapy and training. i," in *Robot and Human Communication, 1992. Proceedings., IEEE International Workshop on*. IEEE, 1992, pp. 161–165. 10

[37] T. McGeer, "Passive Dynamic Walking," *International Journal of Robotic Research*, vol. 9, pp. 62–82, 1990. 13

[38] S. Collins, A. Ruina, R. Tedrake, and M. Wisse, "Efficient bipedal robots based on passive-dynamic walkers," *Science*, vol. 307, no. 5712, pp. 1082–1085, 2005. 13

[39] R. W. Sinnet and A. D. Ames, "2D bipedal walking with knees and feet: A hybrid control approach," in *Conference on Decision and Control*, 2009, pp. 3200–3207. 13

[40] H.-H. Zhao, W.-L. Ma, A. D. Ames, and M. B. Zeagler, "Human-inspired multi-contact locomotion with amber2," in *Cyber-Physical Systems (ICCPS), 2014 ACM/IEEE International Conference on*. IEEE, 2014, pp. 199–210. 13

[41] H. Chen, "Passive dynamic walking with knees: A point foot model," Ph.D. dissertation, Massachusetts Institute of Technology, 2007. 11, 13.2

Yingfu Zeng received his B. Sc. in electronic engineering from XiDian University, China in 2010, and his M. Sc. degree in embedded and intelligent systems in Halmstad University, Sweden in 2012. He is currently working toward his Ph.D in computer science at Rice University. His research interests include domain specific languages, cyber-physical systems and partial evaluation.

Chad Rose received his B.S. degree in mechanical engineering from Auburn University, Auburn, AL, USA in 2012, and his M.S. degree in mechanical engineering from Rice University, Houston, TX, USA in 2015. Supported by an NASA Space Technology Research Fellowship, he is currently working toward his Ph.D. at Rice University as a member of the Mechatronics and Haptic Interfaces Laboratory. His research interests include rehabilitation robotics, human-robot physical interaction, and mechatronics.

Walid Taha received his B. Sc.in computer engineering from the Kuwait University, in 1993, and his Ph. D. degree in computer science from Oregon Graduate Institute, in 1999. He is a Professor at Halmstad University. His current research focus is on modeling, simulation, and verification of cyber-physical systems, and in particular the Acumen language (www.effective-modeling.org). Taha is credited with developing the idea of multi-stage programming and is the designer of several multi-stage languages including MetaOCaml, ConCoqtion, Java Mint, and the Verilog Preprocessor. He has contributed to other programming language innovations such as statically-typed macros, tag elimination, tagless staged interpreters, event-driven functional reactive programming (E-FRP), the notion of exact software design, and gradual typing.

Adam Duracz received his B. Sc. and M. Sc. degrees in computer science from the Stockholm University, in 2005 and 2006, respectively. From 2007 to 2012, he worked at IBM as an IT specialist. Currently, he is a Ph. D. student in the school of information technology at Halmstad University. His research interest covers hybrid systems, rigorous simulation and parallel programming.

Kevin Atkinson received his B. Sc. and M. Sc. degrees in computer science from the University of Maryland in 2001 and 2005, respectively, and his Ph. D. degree from the University of Utah, Salt Lake City in 2011. His research interests focus on language design, extensible syntax and macro systems.

Roland Philippsen received his Ph.D. from EPFL, Switzerland, in 2005 for work on mobile robot path planning and obstacle avoidance. During his thesis and subsequent positions at LAAS-CNRS, ETH, and Stanford University, he has contributed to real-world robots such as the autonomous tourguides of the Swiss Expo.02, educational robots for a student contest, robotic theater actors, a Cognitive Robot Companion, the PR2 of Willow Garage, and the Meka robot of HCRL at UT Austin. His expertise lies in interweaving reasoning, planning, and control, which he applies to intelligent vehicles and his interest in robots that take an active part in modifying their environment.

Robert Cartwright received his B. Sc. in applied mathematics from the Harvard College , in 1971, and his Ph. D. degree in computer science from Stanford University, in 1977. He has served as Program Chair of the ACM Conference on LISP and Functional Programming and ACM Symposium on Principles of Programming Languages and as General Chair for the SIGPLAN Conference on Programming Language Design and Implementation. His current research interests include developing programming languages and environments to support cyber-physical computing (CPS).

Marcia O'Malley Marcia K. O'Malley received her B.S. degree in mechanical engineering from Purdue University, West Lafayette, IN, USA, in 1996, and her M.S. and Ph.D. degrees in mechanical engineering from Vanderbilt University, Nashville, TN, USA, in 1999 and 2001, respectively. She is currently a Professor of mechanical engineering and computer science, Rice University, Houston, TX, USA, and directs the Mechatronics and Haptic Interfaces Laboratory. Her research addresses issues that arise when humans physically interact with robotic systems, with a focus on training and rehabilitation in virtual environments.