Elements of Model-Based Design

Jeff C. Jensen

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2010-19
http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-19.html

February 19, 2010

Elements of Model-Based Design

Jeff C. Jensen
Department of Electrical Engineering & Computer Science
University of California, Berkeley
Berkeley, CA 94720
Email: jeffcjensen@cal.berkeley.edu

Abstract—Model-Based Design (MBD) is a powerful design technique for cyber-physical systems, but too often literature assumes knowledge of a methodology without reference to a specific design process. This report decomposes MBD into ten fundamental steps and introduces an iterative design process that is exercised with two case studies of cyber-physical systems.

Index Terms—model-based design, cyber-physical systems, embedded systems, models of computation, PTIDES.

1 Introduction

Cyber-physical systems [1] are dynamic systems that integrate physical processes with computation, often in feedback loops, where physical processes affect computations and visa-versa. These systems are *reactive* since they maintain a permanent interaction with their environment, and most commonly *real-time* since by design they must conform to externally defined timing constraints [20]. We describe and evaluate a design methodology that leverages mathematical modeling of physical dynamics, formal models of computation, simulation of heterogeneous systems, and code synthesis for cyber-physical systems.

2 Model-Based Design

2.1 Model-Based Design

Model-based design (MBD) [2] [3] [4] [9] emphasizes mathematical modeling to design, analyze, verify, and validate dynamic systems [4]. An embedded computer interacts with the real world through sensing and actuation, impacting the physical dynamics of its environment. A complete model of a cyber-physical system represents its coupling of physical processes and embedded computations. The notion of correctness for real-time embedded software must be extended beyond the traditional functional relationship between inputs and outputs to include the times at which inputs are processed and outputs are produced. In the case of hard real-time or safety-critical systems, timing constraints may be specified by the developer to facilitate the explicit modeling of time.

Mathematical models are used to design, simulate, synthesize, and test cyber-physical systems. Such models are based on system specifications and analysis of the physical context in which the system resides. Modeled systems may be tested and simulated offline [9], enabling developers to verify the logic of their application, assumptions about its environment, and end-to-end (*i.e.* closed-loop) behavior. We introduce an MBD workflow for the design of cyber-physical systems through extensive use of mathematical and computer-aided modeling.

2.2 Models of Computation

Embedded software is commonly written directly in languages like C/C++, Java, or assembly without regard to a formal programming model beyond the language specification itself. Cyber-physical systems are complex systems that demand strong analyzability, reliable performance, and confidence in the correctness of concurrent software; this is especially important in the case of sensor networks where success hinges on scalability, modularization, and the ability to model both individual nodes and the network as a whole.

Traditional programming models such as threading are difficult to analyze, burdening developers to informally reason about correctness, determinism, real-time performance, and reliability. The analysis of threaded software is intractable since any arbitrary execution of concurrent processes is valid, and faulty or nondeterministic interleavings of instructions may exist without detection. The interactions between threads are (in all but the simplest cases) incomprehensible to humans [5]. Without explicit modeling of the interactions between concurrent processes, it is difficult (if not impossible) to reason about deadlock avoidance. While threading is the

This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at the University of California at Berkeley, which receives support from the National Science Foundation (NSF awards #CCR-0225610 (ITR),#0720882 (CSR-EHS: PRET), #0647591 (CSR-SGER), and #0720841 (CSR-CPS)), the U. S. Army Research Office (ARO #W911NF-07-2-0019), the U. S. Air Force Office of Scientific Research (MURI #FA9550-06-0312 and AF-TRUST #FA9550-06-1-0244), the Air Force Research Lab (AFRL), the State of California Micro Program, and the following companies: Agilent, Bosch, Lockheed Martin, National Instruments, Thales and Toyota.

dominant design pattern used in conventional concurrent software, satisfactory results are achieved primarily through the use of semiphores, mutex locks, critical sections, and other techniques that prune the expressiveness of the language. To address these problems in cyber-physical systems where correctness, determinism, and reliability trump design language expressibility, more sophisticated design patterns must be used.

Fast hardware can give the illusion of real-time computing, but traditional programming models lack timing constructs and are unable to guarantee deadlines will be met. The sole reliance on a hardware architecture to deliver real-time performance is brittle and unreliable, as illustrated by Richards' Anomaly, where tasks that were once schedulable yield missed deadlines when processor speed is increased [6]. Software models that incorporate timing constructs simplify real-time analysis, and in some cases guarantee that deadlines will be met.

A *model of computation* is a set of allowable instructions used in a computation along with rules that govern the interaction, communication, and control flow of a set of computational components [9]. A formal model of computation defines semantics that often result in greater analyzability and the potential to simulate computation within a physical environment through the use of heterogeneous modeling tools. Models built in a formal model of computation may be easier to analyze with respect to determinism, execution time, reachability, memory usage, and latency [13] [14]. Designing and modeling an application according to a formal model of computation leads to robust software that is simpler to analyze, sidestepping the many pitfalls of threading. These software dynamics alter the evolution of a cyber-physical system, and if modeled may be generalized and used in an MBD workflow to broaden the understanding of a system.

Advantages of using a specific model of computation depend its semantics, if timing constructs are used, and if it is Turing-complete. Timing constructs are introduced by LabVIEW's structured dataflow with timed loops [7], Simulink [8], discrete event (DE) [11], and Programming of Temporally Integrated Distributed Embedded Systems (PTIDES) [12], allowing developers to reason about the timing behavior of their software irrespective of target hardware architectures. While Turing-complete models of computation lack timing semantics [13] and suffer from undecidability of the number of instructions executed, the domain over which the computation halts, the range of values that may be produced by a computation, memory utilization, and more [15], models of computation that are not Turing-complete bypass many of these limitations. Synchronous models of computation such as Esterel [16], Lustre [17], some variants of statecharts [18], synchronous dataflow [19], and synchronous reactive [20] are not Turing-complete and enable developers to statically analyze models to guarantee memory usage, deadlock avoidance, and timing behavior.

2.3 Elements of Model-Based Design

MBD is an iterative process where modeling of a physical system may influence the selection of hardware and model of computation, both of which may be characterized and used to construct a complete system model. We decompose the MBD of a cyber-physical system into ten steps, though we emphasize that steps need not be ordered as they are here.

2.3.1 State the Problem

Use simple language to describe the problem to be solved, without the use of mathematics or technical terminology. This is the "elevator speech" for the project and is a handy reference for the authors, related teams, colleagues and experts, vendors, and machine shops.

2.3.2 Model Physical Processes

Models of physical processes are simplified representations of real systems. Physical processes are typically described by a deterministic or probabilistic evolution of the system versus time [21]. Models of physical processes are usually in the form of systems of differential equations or Laplace transfer functions. Complex physical processes may be represented by multiphysics simulation methods.

The first iteration of this step should give insight into the physical dynamics at play, with the expectation that this step may be revisited if subsequent steps demand more detailed models.

2.3.3 Characterize the Problem

Isolate fixed parameters, adjustable parameters, and variables to be controlled. Identify quantities that characterize physical processes, such as configuration spaces, safety limitations, input and output sets, saturation points, and modal behavior. Understand how a process may interact with a computation, including end-to-end latency requirements, fault conditions, and response to noise and quantization error.

2.3.4 Derive a Control Algorithm

A control algorithm can be thought of as a set of rules that translates samples of an environment into physical quantities via embedded computation. Determine conditions under which physical processes are controllable and derive a suitable control algorithm to be executed by an embedded computer. Specify maximum tolerable latencies and delays from sensing to actuation; this will aid the selection of an embedded computer, since controller bandwidth and end-to-end latency are limited by processing speed. In highly distributed applications, or systems that are globally asynchronous but locally synchronous, it may be necessary to select a model of computation before a control algorithm can be derived.

2.3.5 Select a Model of Computation

The model of computation used depends on the nature of the problem: does the problem characterization suggest real-time constraints? Must the system be deterministic? Are computers, sensors, or actuators distributed across network boundaries? Are formal verification or validation required? Are inputs and outputs periodic? Synchronous? Prioritize each of these concerns and select the model of computation that fits best. Revisit the derivation of the control algorithm to determine the impact of latency jitter or variable sampling rates introduced by an asynchronous model of computation.

2.3.6 Specify Hardware

Select hardware that is capable of solving the problem as characterized. For each component, consider its input and output bandwidths, delay from input to output, power usage, and lifespan. Actuators should be capable of producing forces and torques in excess of minimum values derived from earlier problem characterization. Consider and model the impacts of using cost-effective substitutes for ideal parts.

Selection of an embedded computer may necessitate a deeper understanding of latency and execution time requirements of control algorithms, worst-case execution time measurements of synthesized software, and reasoning as to how software will interact with a specific hardware architecture. This step may require several iterations with software design and simulation before an embedded computer can be selected with confidence.

2.3.7 Simulate & Solve

Solve the problem using a desktop simulation tool that supports the model of computation. Depending on the robustness of the development environment, incorporate models of sensors, actuators, and physical processes. Following the concept of platform-based design [22], separate application logic and architecture-specific software into modular components. If no development tool can capture the dynamics of the cyber-physical system as a whole, use multiple tools to capture the behavior of dynamic subsystems.

While it may not be possible to model every subsystem within a single tool, the exercise can root out logical errors and garner a better understanding of the system as a whole. What is left out of disjoint simulations is a representation of the relationships between signals that cross subsystem boundaries and the behavior of heterogeneous compositions of subsystems. The interactions between heterogenous systems [23] may be simulated in Ptolemy II [9], LabVIEW with LabVIEW Controls, Design, and Simulation toolbox, and to a limited extent, Simulink.

Specify desired input domain and output range of expected values. If warranted, verify and validate the design based on these specifications or contracts between software components. A number of desktop simulation tools feature verification and validation, or models may be transformed and fed into external tools. Model verification tools include Simulink Design Verifier [24] and the Eclipse Modeling Project Validation Framework [25].

Simulation is a powerful tool throughout the iterative design process and at varied levels of granularity. Subsystems may be constructed and tested independently to produce data that may be incorporated in the next iteration of a model. The scheduler for a model of computation may consider worst-case execution time of computations [14], measurements that cannot be made until code has been synthesized. Control algorithms may be adjusted according to the model of computation, especially if parameters of the algorithm depend on step size or latency.

2.3.8 Construct

Build the device according to specifications, taking note where exceptions have been made that may impact earlier modeling. Plan construction in a way that allows individual components or subsystems to be tested against theoretical models, facilitating iteration between simulation and testing.

2.3.9 Synthesize Software

Code synthesizers are usually incorporated into desktop simulation environments, examples of which are LabVIEW, Simulink, and Ptolemy II. Assuming a code generator synthesizes code that faithfully executes the semantics of the model of computation, the synthesized code is correct by construction. Code generators may directly support the embedded computer used, or generic code may be synthesized and tied to handwritten architecture-specific code. If code synthesis is infeasible or unavailable, handwritten code should carefully follow the selected model of computation.

2.3.10 Test

Configure adjustable parameters to create a test environment that is as simple as possible and test each subsystem independently. Computational systems may be isolated from physical systems via *hardware in the loop* testing, where programmable hardware such as embedded computers or field-programmable gate arrays simulate the feedback from physical or other computational processes. Results in execution time and latency can be used to refine previous models, and unexpected test results may point to errors in modeling or implementation.

2.4 MBD Case Studies

We present two studies of MBD in the construction of cyber-physical systems, showing step-by-step design and analysis. For an arbitrary application, the level of detail at each step need not match what we show here; in simple applications, whiteboard drawings may

suffice, while complex applications may require multiple modeling tools, programming languages, embedded targets, custom hardware, and sophisticated physical and mathematical reasoning. No model can ever be complete [15]; MBD models behavior with enough detail to sufficiently understand and solve the problem given the assumptions about its environment.

In the first case study, we augment an existing cyber-physical system with a new capability. The Cal Climber uses a commercially available robotics platform based on the popular iRobot Roomba autonomous vacuum cleaner. The off-the-shelf platform is capable of driving, sensing bumps and cliffs, executing simple scripts, and communicating with an external controller. We extend the platform with an external embedded computer and a sensor that measures tilt with respect to Earth's gravity. The Cal Climber demonstrates the composition of cyber-physical systems, where a robotics platform is modeled as a subsystem and treated as a collection of sensors and actuators located beyond a network boundary. The application does not require real-time processing of data or complex modeling of physical processes. It offers a simple canonical example of MBD in a practical application.

The second case study is inspired by more sophisticated industrial processes. The Tunneling Ball Device is a cyber-physical system whose operation demands hardware and real-time embedded computing that deliver high-precision sensing and actuation. Computations are event driven, and signals present reflect those in an automotive engine control unit for control of fuel injection, ignition timing, and valve retraction of an automotive engine. The system is naturally extensible to a distributed platform, presenting an interesting example for modeling distributed cyber-physical systems. Dynamics are captured iteratively and with greater detail throughout each step of the design process.

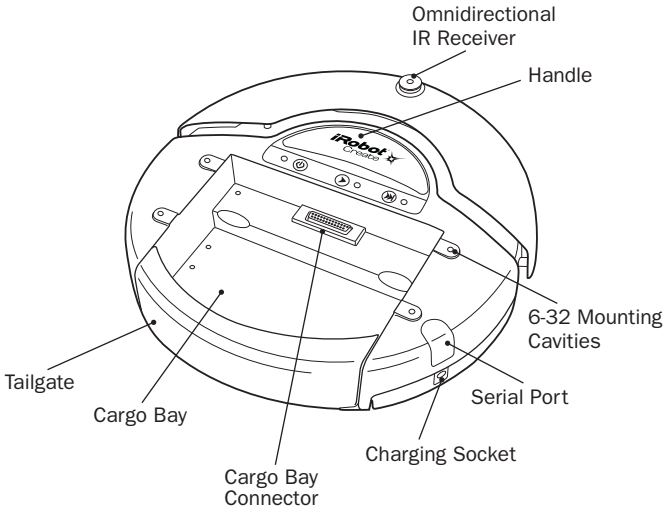


Fig. 1: iRobot Create [27] robotics platform. *Source: iRobot Corporation.*

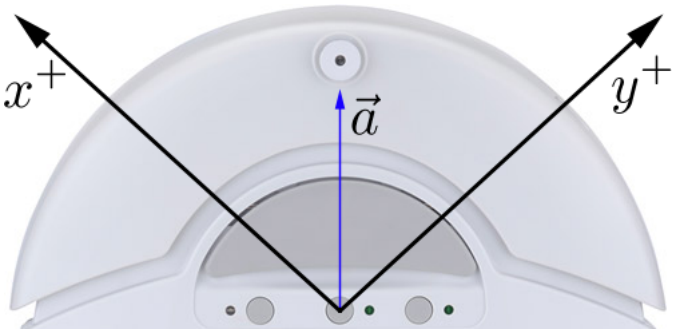


Fig. 2: Coordinate system for iRobot orientation with respect to gravity. The acceleration vector has positive and equal x and y components, indicating uphill orientation.

3 Cal Climber

3.1 State the Problem

Modify the iRobot Create [27] (Fig. 1) consumer robotics platform to autonomously navigate to the top of an incline, avoiding cliffs and obstacles along the way.

3.2 Model Physical Processes

Tilt is a measurement of the orientation of the robot with respect to gravity. We fix a coordinate system so that uphill orientation corresponds to a vector with a $45°$ angle with respect to the positive x and y axes (Fig. 2). We represent tilt as a vector $\vec{a} \in \mathbb{R}^2$, $\vec{a} = (a_x, a_y)$, whose magnitude is proportional to the component of gravitational acceleration in the plane of the robot. If the robot is level then $\vec{a} = \vec{0}$, if the robot is oriented uphill then $\angle\vec{a} = 45°$, if the robot is tilted to the left then $\angle\vec{a} = -45°$, if the robot is tilted to the right then $\angle\vec{a} = 135°$, and if the robot is tilted downhill then $\angle\vec{a} = -135°$.

Four bump sensors distributed in the front of the robot detect when it impacts an object. Four infrared sensors detect cliffs, and wheel drop sensors detect when a wheel is no longer in contact with the ground. No cliff or bump detection is available from the rear of the vehicle.

3.3 Characterize the Problem

We have not constrained the rate with which the robot must ascend an incline, hence the overall rate of the robot is an adjustable parameter. The left and right wheels of the robot may be actuated independently, and the ratio of their rates is a controllable parameter. Each wheel can rotate at a rate of up to $\pm v_{\max}$, where $v_{\max} = 500\text{mm/s}$. The robot does not have out-of-the-box capability to measure acceleration, so we will add a sensor to measure acceleration. Since acceleration due to gravity is a DC signal, we pass tilt measurements through a lowpass filter. Filtering and advanced logic are beyond the capabilities of the factory-embedded computer, prompting us to use an external embedded computer. Wheel actuation commands and sensor feedback can be communicated between the robotics platform and an external embedded computer via UART serial, with a latency of at most 15ms (a fixed parameter).

If the robot is on level ground, then $\|\vec{a}\| = \vec{0}$ and $\angle\vec{a}$ is undefined. In practice, the accelerometer will register some small, nonzero value, and the angle will be erratic. To prevent jitter, we set a magnitude threshold $A_{\text{threshold}}$ to differentiate near-level ground from an incline. In both level ground and climbing modes, we increment wheel speed slowly to reduce accelerometer noise induced by changes in wheel speeds, effectively introducing a second lowpass filter into the system. Measurements with magnitudes larger than acceleration due to gravity will be clipped.

Climbing a hill does not present hard real-time constraints; however, response to cliff and object detection must happen quickly to avoid hard impacts or falling off a cliff. The embedded computer will place a high priority on responding to these sensor events.

3.4 Derive a Control Algorithm

A simple proportional control algorithm will change wheel speeds in response to tilt measurements. If the robot is tilted left of uphill orientation, the left wheel speed should increase relative to the right wheel to correct the orientation, and visa-versa. The left and right wheel speeds, $v_x, v_y : \mathbb{R}^2 \longrightarrow [-v_{\max}, v_{\max}]$, respectively, are given by

$$v_x(\vec{a}) = \begin{cases} v_{\max}\frac{a_x}{\|\vec{a}\|} & \text{if } \|\vec{a}\| \geq A_{\text{threshold}} \\ v_{\max} & \text{if } \|\vec{a}\| < A_{\text{threshold}} \end{cases} \quad (1a)$$

$$v_y(\vec{a}) = \begin{cases} v_{\max}\frac{a_y}{\|\vec{a}\|} & \text{if } \|\vec{a}\| \geq A_{\text{threshold}} \\ v_{\max} & \text{if } \|\vec{a}\| < A_{\text{threshold}}. \end{cases} \quad (1b)$$

The proportional control algorithm correctly orients the robot, with one interesting side effect being an unstable equilibrium point at $a_x, a_y < 0$ and $a_x = a_y$, resulting in the robot attempting to back up the hill. Any deviation from this point will cause the robot to quickly rotate about and return to uphill climbing. A more advanced control algorithm could implement mode changes to prevent this behavior. This is advantageous not only from the standpoint of optimality, but also safety, since bump and cliff sensors are not present on the rear of the robot and hence backing up a hill increases the risk of undetected incident obstacles.

3.5 Select a Model of Computation

The robot can be configured to transmit sensor packets periodically, or only when polled. A synchronous model of computation would be well-suited for periodic sensor updates, and an asynchronous model of computation would suffice for either periodic or aperiodic sensor updates. The LabVIEW programming environment enables desktop simulation and its code synthesis framework natively supports many embedded computers. We select LabVIEW for its ease of use and ability to synthesize code from a desktop simulation.

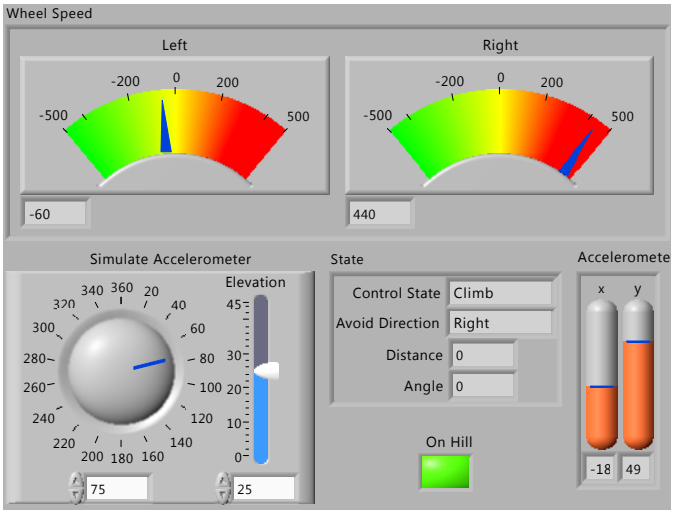


Fig. 3: LabVIEW front panel of the Cal Climber model. The user specifies elevation and rotation in the bottom left control, and control algorithm outputs and ADC sample values are displayed in the remaining indicators. In this simulation, the robot is tilted to the right by 75°, and the wheels adjust to compensate.

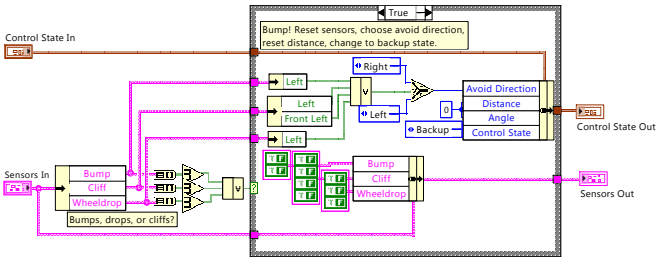


Fig. 4: Block diagram of state evolution block of the LabVIEW model of the Cal Climber. The blocks displayed represent the logic to change state if bump, cliff, or wheel drop sensors are fired.

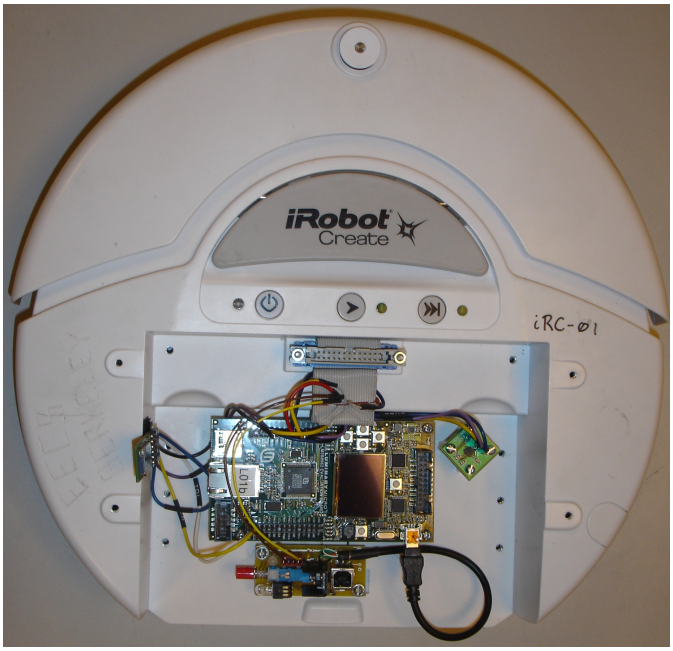


Fig. 5: Modified iRobot Create with Luminary Micro LM3s8962 embedded computer and ADXL-220 accelerometer.

3.6 Specify Hardware

An ADXL-220 [28] two-axis accelerometer outputs an analog voltage proportional to measured acceleration. The Luminary Micro LM3s8962 [37] embedded computer supports UART serial to communicate to the robot, has an analog-to-digital converter (ADC) to sample the accelerometer, a 5V source to power the accelerometer, can be powered from the robot battery, and is easily mounted along with the sensor in the cargo bay of the robot.

3.7 Simulate & Solve

We verify application logic by simulating in a desktop environment. The accelerometer is represented by a dial that may be adjusted to change the simulated tilt from $0°$ to $45°$, and the tilt of the robot by a dial that may be turned from $0°$ to $360°$. The output of the virtual accelerometer mimics the output of the ADC on the embedded computer as it samples the accelerometer, where the analog signal is simulated according to the accelerometer datasheet and the user-provided incline and tilt. Robot sensor packets are generated periodically and passed into the application for processing. Users may manually trigger bump, cliff, and wheel drop events. Portions of the desktop simulation model are shown in Figs. 3–4.

Our application logic mimics a state machine with states for drive, climb, backup, turn, and reset modes. Bump, cliff, and wheel drop events trigger an object avoidance sequence which instructs the robot to back up and turn away from a detected object. The simulation displays the speeds of the left and right wheels, as well as the program state, in response to user input. Once our application logic has been verified in simulation, we construct the device.

3.8 Construct

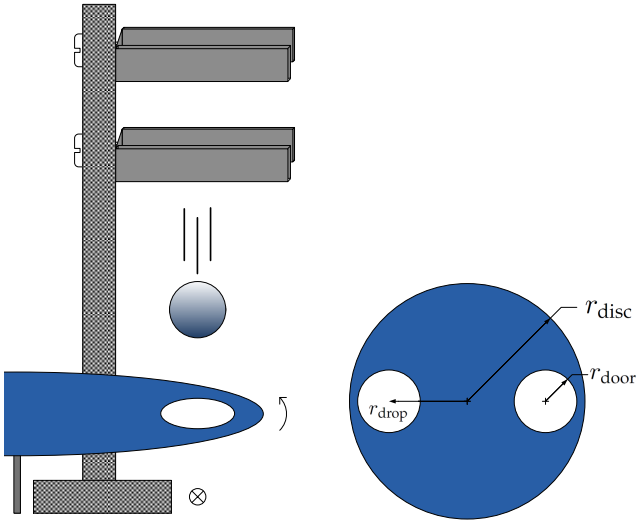
The configuration is shown in Fig. 5, and wiring schematics are shown in Appendix A.

3.9 Synthesize Software

LabVIEW natively supports the embedded computer, and by use of the *conditional disable* structure, we use the same model for desktop simulation and embedded code synthesis. C code is automatically synthesized and downloaded to the embedded computer.

3.10 Test

The robot successfully climbs towards the top of an incline and avoids obstacles and cliffs along the way. Actuation is smooth and the robot returns quickly to the correct orientation when manually displaced.



(a) Illustration of the device. The drop target, located below the optical sensors, is marked by an inscribed circle.

(b) Spinning disc of the Tunneling Ball Device.

Fig. 6: The Tunneling Ball Device

4 The Tunneling Ball Device

4.1 State the Problem

Steel ball bearings are dropped one at a time at sporadic intervals towards a fixed drop target located below a spinning disc (Fig. 6). The disc has been bored through at two opposite ends, and the ball will pass ("tunnel") through untouched if the disc is correctly aligned at the time of impact. Should the disc be improperly rotated, the ball will collide with the disc, signifying failure. The device must sense when a ball is dropped, track the position of the disc, and adjust the trajectory of the disc so that the ball tunnels through the disc unscathed. Only one ball will be above the disc at any time, and between drops the disc should return to a default speed. The disc must not stop at any time, and changes in rate should be minimal.

4.2 Model Physical Processes

4.2.1 Kinematics of a Ball in Freefall

A ball $\beta = (z_0, v_0, t_0) \in \mathrm{B} \subset \mathbb{R}^2_+ \times \mathbb{R}$ is a tuple of its initial altitude, initial velocity, and time at which it is detected above the drop target. Let $z : \mathrm{B} \times \mathbb{R} \longrightarrow \mathbb{R}$ be the vertical distance from the center of the ball to the middle of the disc,

$$z(\beta, t - t_0) = z_0 - v_0 t - \frac{1}{2} g t^2 \tag{2}$$

where g is constant acceleration due to gravity [29].

A ball with radius r_{b} first comes into contact with the disc at arrival time $T_a(\beta)$, is centered in the disc (assuming no impact has occurred) at time $T_c(\beta)$, has departed the disc at time $T_d(\beta)$, and is known to be above the disc for time $\Delta T(\beta)$:

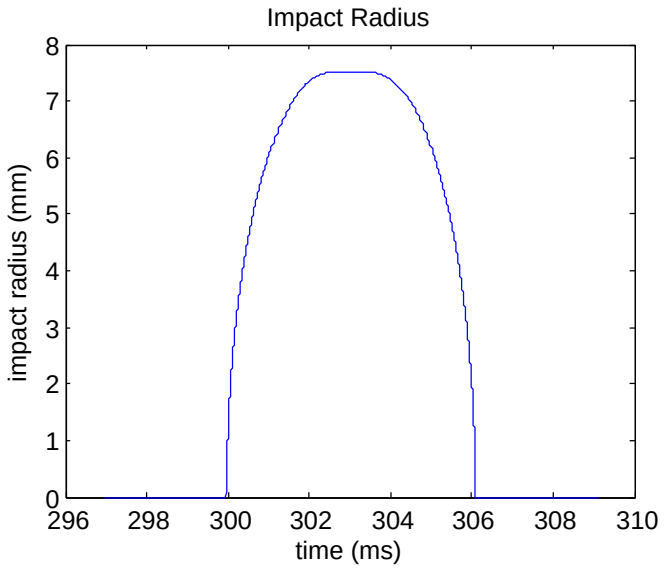


Fig. 7: Impact radius of a 7.5mm radius ball dropped from a height of 400mm. The impact radius is equal to the radius of the ball while the center of the ball passes through the thickness of the disc.

$$\begin{aligned} T_a, T_c, T_d &: \mathrm{B} \longrightarrow \mathbb{R} \\ \Delta T &: \mathrm{B} \longrightarrow \mathbb{R}_+ \end{aligned} \tag{3}$$

$$T_a(\beta) = \frac{1}{g}\left(\sqrt{v_0^2 + 2g\left(z_0 - r_\mathrm{b} - \frac{h}{2}\right)} - v_0\right) + t_0 \tag{4a}$$

$$T_c(\beta) = \frac{1}{g}\left(\sqrt{v_0^2 + 2gz_0} - v_0\right) + t_0 \tag{4b}$$

$$T_d(\beta) = \frac{1}{g}\left(\sqrt{v_0^2 + 2g\left(z_0 + r_\mathrm{b} + \frac{h}{2}\right)} - v_0\right) + t_0 \tag{4c}$$

$$\Delta T(\beta) = T_a(\beta) - t_0 \tag{4d}$$

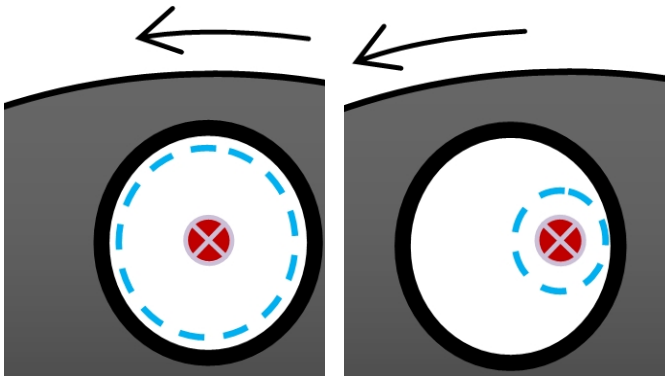
where h denotes the thickness of the disc. The *drop interval* $[t_0, T_a(\beta)]$ is the duration for which a ball is known to be above the disc, and the *impact interval* is the interval over which the ball is passing through the disc, given by $I : \mathrm{B} \longrightarrow \mathcal{P}(\mathbb{R})$ (where $\mathcal{P}$ is the powerset operator)

$$I(\beta) = [T_a(\beta), T_d(\beta)]\,. \tag{5}$$

The *impact radius* is the radius $R_I : \mathrm{B} \times \mathbb{R} \longrightarrow \mathbb{R}_+$ that describes the widest horizontal slice of the ball that is passing through the disc (Fig. 7):

$$R_I(\beta, t) = \begin{cases} 0 & \text{if } |z(\beta,t)| > r_\mathrm{b} + \frac{h}{2} \\ r_\mathrm{b} & \text{if } |z(\beta,t)| < \frac{h}{2} \\ \sqrt{r_b^2 - \left(|z(\beta,t)| - \frac{h}{2}\right)^2} & \text{otherwise.} \end{cases} \tag{6}$$

An equivalent condition for the impact radius to be nonzero is that t fall within the impact interval $I(\beta)$.



(a) Disc rotated with the door centered over the drop target (optimal tunnel).

(b) Disc rotated with the door offset from the drop target (sub-optimal tunnel).

Fig. 8: Disc rotations showing optimal and sub-optimal tunnels. The drop target is centered, and the tunnel is represented by the broken line. The impact radius must be smaller than the tunnel radius for the ball to pass.

4.2.2 Kinematics of a Rotating Disc

Let $\vartheta = [\mathbb{R} \longrightarrow (-\pi, \pi]]$ be the set of functions that describe the rotation of a disc over time. The disc has two doors bored at opposing ends, each with radius r_door. We fix a coordinate system so that the doors on the disc are centered above the drop target at rotation 0 and π. The distance from the center of the disc to the drop target is r_drop and is the quantity that relates the angular velocity of the disc to the linear velocity of the center of the door (Fig. 6(b)).

The Euclidean distance $d : \vartheta \times \mathbb{R} \longrightarrow \mathbb{R}_+$ from the drop target to the center of the nearest of two doors is

$$\begin{aligned} d(\theta, t) =& \sqrt{2} r_\mathrm{drop} \sqrt{1 + \min\left\{\cos\left(\theta(t)\right), \cos\left(\pi - \theta(t)\right)\right\}} \\ =& 2 r_\mathrm{drop} \sin\left(\tfrac{1}{2} \min\left\{|\theta(t)|, |\pi - \theta(t)|\right\}\right). \end{aligned} \tag{7}$$

As the disc rotates, the doors pass over the drop target exposing the tunnel through which a ball may pass (Fig. 8). The *tunnel radius* $R_T : \vartheta \times \mathbb{R} \longrightarrow \mathbb{R}_+$ is the largest allowable impact radius at time t:

$$R_T(\theta, t) = \begin{cases} r_\mathrm{door} - d(\theta, t) & \text{if } d(\theta,t) \le r_\mathrm{door} \\ 0 & \text{if } d(\theta,t) > r_\mathrm{door}. \end{cases} \tag{8}$$

4.2.3 Dynamics of a DC Motor with Load

To rotate the disc, was must connect it to a driving source, such as a motor. DC brushed motors are a reasonable starting point for analyzing the rotation of the disc because they are simple, inexpensive, and widely used. They are commonly driven by pulse-width modulation (PWM) generators (though modeled as if driven by linear-gain amplifiers) and provide position information via attached digital encoders. If later modeling requires more advanced motor technology, we will reexamine this choice. To determine the load placed on the motor, we calculate disc inertia by applying the parallel axis

theorem of inertial masses [29] and relating to disc density ρ and thickness h by

$$J_{\text{disc}} = \pi\rho h \left[\frac{1}{2}r_{\text{disc}}^4 - 2\left(\frac{1}{2}r_{\text{door}}^4 + r_{\text{door}}^2 r_{\text{drop}}^2\right)\right]. \quad (9)$$

The net inertia J of the rotating system is the sum of the disc inertia, motor armature inertia (assuming inertia of an optionally attached encoder is negligible), armature axle inertia, and disc axle inertia,

$$J = K_G^2\left(J_{\text{armature}} + J_{\text{armatureAxle}}\right) + J_{\text{disc}} + J_{\text{discAxle}} \quad (10)$$

where K_G is the gear ratio from the load to the motor.

A standard DC brushed motor with torque constant K_τ, armature resistance R, armature inductance L, damping coefficient b, back-electromotive force constant K_B, and input voltage amplification K_A is modeled by the system of linear differential equations [30]

$$\tau(t) = K_\tau i(t) \quad (11)$$

$$K_G\tau(t) = b\frac{d\theta(t)}{dt} + J\frac{d^2\theta(t)}{dt^2} \quad (12)$$

$$K_A v(t) = Ri(t) + L\frac{di(t)}{dt} + K_B\frac{d\theta(t)}{dt} \quad (13)$$

where $\tau(t)$ is the torque produced by the motor and $i(t)$ is the armature coil current induced by input voltage $v(t)$. Moving to the Laplacian domain, the transfer function of the system is [30]

$$\frac{\Theta(s)}{V(s)} = \frac{K_A K_G K_\tau}{JLs^3 + (RJ + Lb)s^2 + (Rb + K_G^2 K_B K_\tau)s}, \quad (14)$$

where s is the Laplace complex variable.

4.3 Characterize the Problem

We characterize the Tunneling Ball problem by seven fundamental quantities: minimum drop time, maximum correction angle, the minimum torque to effect this angle within the minimum drop time, the minimum voltage required to produce this torque, lower and upper bounds on disc rate, position tolerance, and conditions for success. We translate these quantities into physical parameters used to select appropriate hardware for the device.

4.3.1 Minimum Drop Time

Balls are dropped with forces are present beyond gravity and drag, hence the set of initial velocities is bounded above. While no ball can achieve speed higher than its terminal velocity unless an outside force is applied [29], we assume dropped balls have reasonably small initial velocities and that the drop altitude z_0 is small enough that velocity at arrival time is much less than terminal, so that we may disregard drag. Assuming we can reasonably bound $v_{\max}$ or measure it experimentally, we find from (4a) and (4d) that the drop time ΔT is bounded below by $t_{\min}$, where

$$v_{\max} = \max_{\beta\in\text{B}} v_0 \quad (15)$$

$$t_{\min} = \min_{\beta\in\text{B}} \Delta T(\beta). \quad (16)$$

4.3.2 Maximum Correction Angle

For a ball to pass through the disc, a tunnel must be present at the time of impact, likely requiring the position of the disc when the ball arrives be altered from that of its original trajectory. Any point in a door is never more than one-quarter rotation away from the drop target, which sets maximum position error

$$\theta_{\max} = \frac{\pi}{2}, \quad (17)$$

which is independent of any control or planning algorithm employed. While these algorithms should seek to position the door to form an optimal tunnel (at the expense of greater rotation), $\theta_{\max}$ reflects that at any point in time over the impact interval, the disc can at most be one-quarter rotation from its desired position.

4.3.3 Minimum Torque

Equation (17) establishes that the final position of the disc may need to be altered from its original trajectory by at most $\theta_{\max}$ in at least $t_{\min}$ seconds. The trajectory with the least maximum torque that adjusts for the maximal correction angle is given by Maupertuis' principle of classical mechanics [31], and is the result of applying a constant torque $\tau_{\min}$ over the drop interval. We take the problem statement together with the differential equations governing the physical processes at play and solve to find this torque. Given a ball $\beta \in \text{B}$ and disc rotation $\theta \in \vartheta$ for $t < t_0$, we solve motor equations (11) and (12) subject to $\Delta T(\beta) = t_{\min}$, $\tau(t) = \tau_{\min}$, $\theta(t_0) = \theta_0$, and $\frac{d\theta}{dt}(t_0) = \omega_0$, for $t \geq t_0$:

$$\delta(t) \triangleq 1 - e^{-\frac{b}{J}t} \quad (18)$$

$$\theta(t - t_0) = \theta_0 + \frac{J}{b}\omega_0\delta(t) + \frac{K_G}{b}\left(t + \frac{J}{b}\delta(t)\right)\tau_{\min} \quad (19)$$

$$\begin{aligned}\tau_{\min} &= \min_{\theta_0\in(-\pi,\pi]}\left|\left(-\frac{b}{K_G}\right)\frac{\theta_0 + \frac{J}{b}\omega_0\delta(t_{\min})}{t_{\min} + \frac{J}{b}\delta(t_{\min})}\right| \\ &= \left(\frac{b}{K_G}\right)\frac{\theta_{\max}}{t_{\min} + \frac{J}{b}\delta(t_{\min})}.\end{aligned} \quad (20)$$

The minimum torque equation (20) follows from our assumption that the final position of the disc is be driven towards the nearest door, a rate-optimal approach that minimizes the change in rate over the drop interval by altering the final position of the disc by no more than $\pm\theta_{\max}$. An energy-optimal approach would place preference on slowing the disc to take advantage of damping; however, this results in adjustments larger than $\theta_{\max}$, and by the mean-value theorem of calculus [32], a larger change in rate. $\tau_{\min}$ represents the minimum torque necessary for a nonzero tunnel to be present at ball arrival time while minimizing the change in rate. Applying the minimum torque over the drop interval is necessary for the "worst-case" ball to pass through

the disc, but not sufficient; rather, $\tau_{\min}$ establishes a lower-bound on the motor torque required in order for the Tunneling Ball problem to be solvable. A planning algorithm seeking to create an optimal tunnel would make use of greater rotation at the cost of increased torque.

4.3.4 Minimum Voltage

The minimum voltage $v_{\min}$ is the voltage applied to the motor that is required to produce constant torque $\tau_{\min}$. Substituting $\tau(t) = K_\tau i(t) = \tau_{\min}$ into equations (13) and (19),

$$K_A v_{\min} = \left(\frac{R}{K_\tau} + \frac{K_B K_G}{b}\delta(t_{\min})\right)\tau_{\min} + K_B\omega_0\left(1-\delta(t)\right). \tag{21}$$

We have assumed that $K_\tau i(t_0) = \tau_{\min}$, which anticipates an instantaneous change in current and hence $v(t_0) = \infty$. While this is not realizable in a physical system, we assume motor inductance to be negligible to establish reasonable lower bounds. Tighter bounds may be calculated by solving equations (11) – (13) subject to power supply saturation constraints; these effects are nonlinear, and are beyond the scope of this study.

4.3.5 Rate Bounds

Substituting $\tau_{\min}$ into (19) and solving for $\omega_{\min} = \frac{d\theta}{dt}(t_{\min})$ yields the highest rate achieved over the correction interval,

$$\omega_{\min} = \omega_0 + \left(\frac{K_G}{b}\tau_{\min} - \omega_0\right)\delta(t_{\min}). \tag{22}$$

The motor must be capable of rotating at this rate in order to correct for maximum error, though a control or planning algorithm would make use of higher rates.

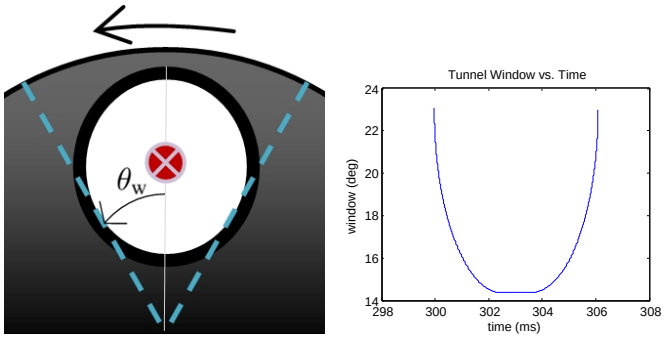
If the disc rotates fast enough, the tunnel is not open long enough for any ball to pass through the disc regardless of initial conditions. The maximum rate $\omega_{\max}$ the disc may travel and still have a nonzero tunnel present over the impact interval is

$$\omega_{\max} = \frac{2\theta_w}{\min\limits_{\beta\in \mathrm{B}}\|I(\beta)\|}. \tag{23}$$

Invoking the mean-value theorem of calculus [32], any rotation of the disc that exceeds an average rate of $\omega_{\max}$ over the impact interval will result in failure. For small values of $t_{\min}$, it may be necessary to rotate the disc quickly through the drop interval to position the door, and then slowly over the impact interval to a rate averaging less than $\omega_{\max}$ to allow the ball to pass. The rate over the drop interval then exceeds $\omega_{\max}$, but its average value is bounded by $\theta_{\max}/t_{\min}$.

4.3.6 Position Tolerance

A tunnel is present if a door is rotated over the drop target. The tunnel angle θ_w is the angle at which a tunnel



(a) Range through which a nonzero tunnel exists, $(-\theta_w, \theta_w)$. (b) Tunnel window as a ball passes through the disc.

Fig. 9: Illustration of tunnel angle θ_w and simulation of tunnel window.

appears (Fig. 9(a)),

$$\begin{aligned}\theta_w =& \cos^{-1}\left(1 - \frac{1}{2}\left(\frac{r_{\text{door}}}{r_{\text{drop}}}\right)^2\right)\\ =& 2\sin^{-1}\left(\frac{r_{\text{door}}}{2r_{\text{drop}}}\right).\end{aligned} \tag{24}$$

The tunnel window $\phi : \mathrm{B} \longrightarrow (0, \theta_w]$ is the absolute amount by which the disc may be rotated away from the drop target and still present a tunnel larger than the impact radius (Fig. 9(b)),

$$\phi(\beta, t) = 2\sin^{-1}\left(\frac{r_{\text{door}} - R_I(\beta,t)}{2r_{\text{drop}}}\right). \tag{25}$$

As a ball approaches and departs the disc, $R_I(\beta, t) \to 0$ and $\phi(\beta, t) \to \theta_w$ as expected.

The position tolerance $\epsilon : \mathrm{B} \times \vartheta \times \mathbb{R} \longrightarrow (-\pi, \theta_w]$ if positive is the largest angle by which the disc may be rotated from its current position in either direction without impacting the ball, if negative is the angle by which the disc must be rotated for the tunnel radius to equal the impact radius, and is a measure of the tolerable error in position over the impact interval (Fig. 10):

$$\epsilon(\beta, \theta, t) = \phi(\beta, t) - \min\left\{|\theta(t)|, |\pi - \theta(t)|\right\}. \tag{26}$$

If the disc is outside the tunnel window, or within the tunnel window but the tunnel radius is smaller than the impact radius, then $\epsilon(\beta, \theta, t) \leq 0$, indicating impact has occurred. If the position tolerance is positive over the impact interval, then the ball passes through untouched.

4.3.7 Success

A ball successfully tunnels through the disc if its impact radius is smaller than the tunnel radius for all times within the impact interval (Fig. 11). The predicate $\mathbf{S} : \mathrm{B} \times \vartheta \longrightarrow \{\text{true}, \text{false}\}$ is the conditional for success given a ball and a trajectory,

$$\begin{aligned}\mathbf{S}(\beta, \theta) &\Leftrightarrow \left[t \in I(\beta) \Rightarrow R_I(\beta, t) \leq R_T(\theta, t)\right]\\ &\Leftrightarrow \left[t \in I(\beta) \Rightarrow \epsilon(\beta, \theta, t) > 0\right].\end{aligned} \tag{27}$$

The success predicate establishes both necessary and sufficient conditions for the ball to pass through the

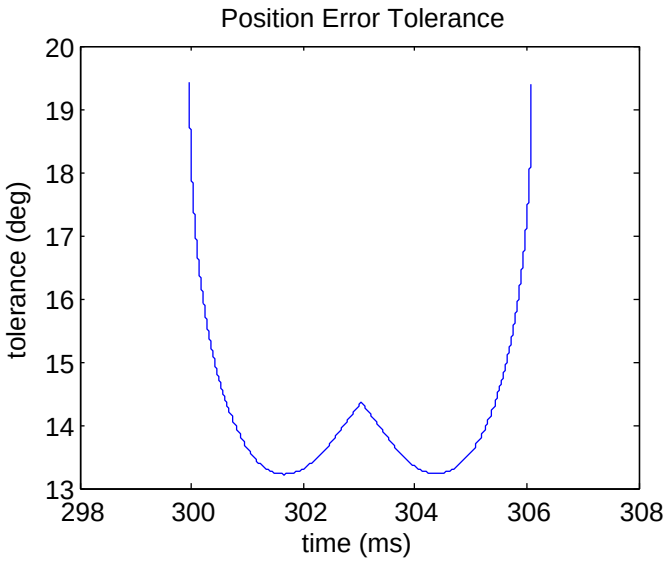
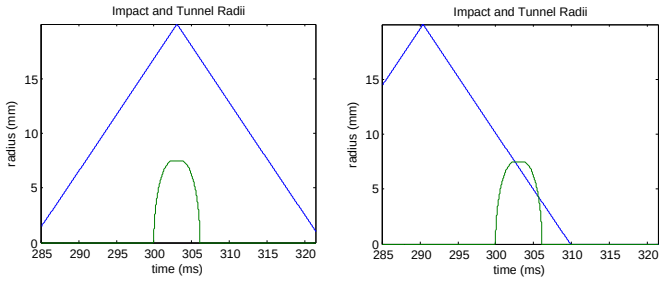


Fig. 10: Simulated position tolerance. The tolerance is always nonzero, indicating the ball does not impact the disc.



(a) Ball passes through the center of the door. The impact radius is always smaller than the tunnel radius, resulting in success.

(b) Ball does not pass through the center of the door. The impact radius becomes larger than the tunnel radius at $t \approx 302$ms, resulting in failure.

Fig. 11: Simulation of successful and unsuccessful ball drops.

disc untouched. There are scenarios where a ball impacts the side of the door but still bounces through the door towards the drop target, in which case equation (27) is sufficient but not necessary. These boundary cases do not significantly extend the conditions that lead to a successful tunnel and introduce additional complexity to modeling and simulation; we have excluded them from this study. We have not restricted the condition of success to disc trajectories that are physically realizable, which disregards physical constraints, such as maximum motor torque. We take these constraints into account in trajectory planning.

4.4 Derive a Control Algorithm

The motor transfer function (14) shows a system pole at $s = 0$, indicating position is uncontrollable in an open-loop configuration [30]. To govern the trajectory of the disc, we place the motor in a feedback loop with an embedded microcontroller. The microcontroller produces output (either analog or PWM) that is amplified and

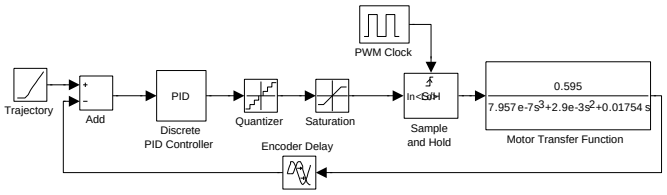


Fig. 12: Simulink model of the motor and PID controller.

applied to the motor, and consumes digital encoder pulses as position feedback.

We synthesize a position controller for position stabilization and trajectory. Position control of a DC motor may be implemented by a proportional, integral, derivative (PID) control algorithm [39], allowing independent development of a planning algorithm. This follows the system design paradigm *orthogonalization of concerns*, or decoupling aspects of design to allow for more effective exploration of alternative solutions [40]. Assuming a PID or similar control algorithm can track a useful range of disc trajectories, trajectory planning may then be implemented independently. This can enable, for example, the PID control algorithm to be executed on a simple microcontroller (or motor controller), while trajectory planning may be executed on a more advanced hardware platform. Logical orthogonalization of software is of greater interest, since trajectory planning can solve the Tunneling Ball problem in a way that is optimal with respect to tunnel radius, while the control algorithm may be derived in a way that is optimal with respect to the transfer function of the motor.

The PID control algorithm may be automatically synthesized from a simulation software, such as LabVIEW, MATLAB, or Simulink. We used Simulink to model of the motor plant in a feedback loop with a PID controller (Fig. 12), a model that we later used to tune the PID constants during the simulation step. After specifying hardware, we revisit this step to automatically tune the PID controller using the MATLAB SISO tool; simulated tracking error of the closed-loop system is shown in Fig. 13, and the resulting proportional, integral, and derivative constants for the discrete PID controller are, respectively, $K_p = 4$, $K_i = 10$, $K_d = 0.2$. With these values, settling time is less than 125ms (half the minimum drop time), and steady-state error is less than 0.1% of a rotation.

4.5 Select a Model of Computation

Three types of signals are present in the Tunneling Ball Device: periodic, *pseudoperiodic*, and *sporadic*. A *pseudoperiodic signal* is a signal whose domain may be partitioned into connected intervals over which the signal is either periodic or a monotonic chirp. A *sporadic signal* [41] is a signal that is nonzero in at most a countably infinite number of points, where the separation between these points is bounded below. Sporadic events may be periodic, aperiodic, or randomly distributed (provided the

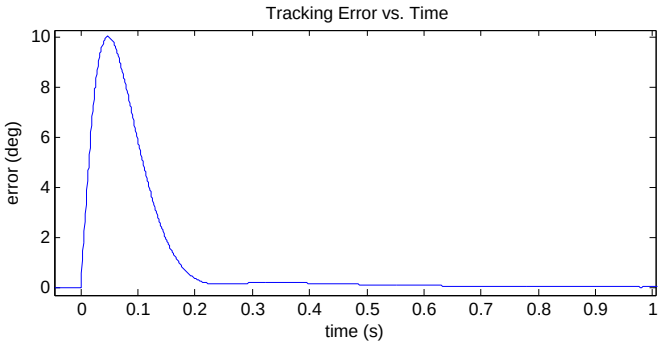


Fig. 13: Simulated error in disc position tracking via PID control. For times $t < 0$, the disc is in steady-state with $\omega_0 = 4$rps. For times $t \geq 0$, the desired position tracks a new rate of 5rps $\approx \omega_0 + \theta_{\max}/t_{\min}$, and the PID controller adjusts to compensate.

minimum time gap is never violated).

If we use a PWM generator that is external to our microcontroller, it will likely sample an analog input and change the duty cycle of its output, both with fixed period. Ball drops are sporadic. A three-phase motor encoder generates a pulse signal for alignment and a second for position. Given a step input, a motor responds with transient and steady-state components [42]; when transient components dominate, encoder pulse arrival is described by an exponential chirp, and when steady-state components dominate, encoder pulses are essentially periodic. Using the imprecise partitioning of transient and steady-state behavior, and that a linear time-invariant system may be characterized by its impulse (equivalently step) response [42], we model the encoder and alignment signals as psuedoperiodic. Though not mathematically rigorous, this partitioning agrees with the intuition that encoder pulses should arrive with increasing frequency when the disc is speeding up, decreasing frequency when slowing down, and roughly constant frequency when in steady-state.

We must choose a model of computation that is well-suited to handle the mixture of periodic, pseudoperiodic, and sporadic signals. A synchronous language would be a poor fit, since pseudoperiodic signals have varied frequencies and sporadic signals may be aperiodic, forcing synchronous models to execute at a high frequency [43]; additionally, synchronous models have difficulty handling non-negligible computation times and sensor latencies. All signals of interest are digital in nature, indicating that continuous models are also a poor fit. Statecharts are an option, though the system is easily represented with only a few simple states, and timing behavior is not easily analyzed.

Two models of computation that appeal us are DE and PTIDES. DE employs a super-dense model of time that captures the passage of physical time as a continuous model would, but its computations are discrete and event-triggered in nature, appealing to the discrete signals present in the device. Time metrics inform a model not just of causal relationships between events (as would be available in many synchronous models), but also delays between them.

PTIDES, which faithfully executes DE semantics, extends DE by specifying timing constraints at sensor and actuator boundaries, which are used to produce real-time guarantees through static analysis of the model. PTIDES models may be statically analyzed to determine causal relationships, relaxing DE constraints by allowing out-of-order execution of event processing as long as the determinism of the underlying DE model is unchanged. This extends the class of models for which feasible DE schedules exist, as out-of-order processing may reduce the delay between sensing and actuation [12].

Unlike DE, which has shown to be a strong language for simulation but is rarely used to synthesize code, PTIDES models are readily deployed to embedded targets while still benefiting from the powerful simulation features of DE. PTIDES is an inherently distributed model of computation, which allows the modeling of systems that are separated by network boundaries. The distribution of computational systems over a network further orthogonalizes a problem through modularization.

A natural distributed extension to our device would be to employ a high-rate embedded system such a field-programmable gate array to decode the motor and to execute the control algorithm, and a more powerful embedded computer for kinematics calculations and trajectory planning. Given the discrete, mixed-signal nature of inputs and the potential opportunities for distributed computation, we find PTIDES to be the best fit to solve the Tunneling Ball problem. We use Ptolemy II [9] to simulate the application and to synthesize code.

4.6 Specify Hardware

A children's "rolling ball" kit that includes a mechanical bucket elevator, ramps, funnels, and marbles, was kicking around our lab. We select this kit as our starting point, and challenge ourselves to derive the remaining components by reasoning about the properties of this kit and by using modeling and simulation to justify our decisions. We fix the drop altitude to the height of the elevator, and we set the ball radius to that of the marbles. We choose a door that is slightly larger than the ball, and size the disc accordingly. We construct the disc from G10F4 garolite, which is strong enough to withstand the impact of the ball and lightweight to reduce inertial load on the disc. The dimensions of the selected components are shown in Table 1, along with the minimum drop time we determined experimentally.

The bounds on rate, torque, and voltage from the problem characterization guide us in motor selection. Motors commonly have a wide range of operation with respect to rate, and gearing may be used to convert motor speed into disc torque [30]. As ω_0 becomes small, torque required from the motor becomes constant and proportional to $\theta_{\max}$, yielding the final rate $\omega_{\min}$ at arrival time. As ω_0 increases, the tunnel window contracts

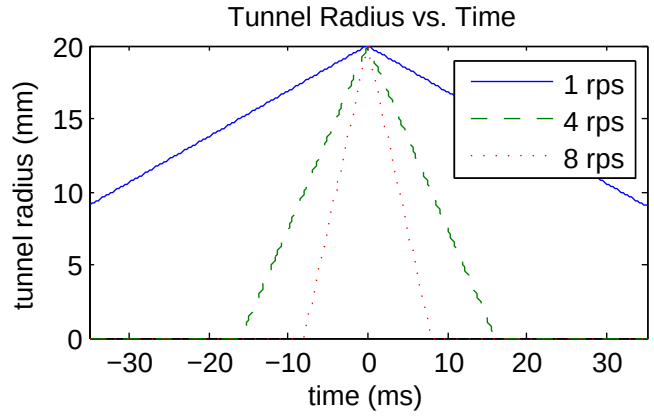


Fig. 14: Tunnel radius $R_T(t)$ for different values of ω_0. Motor constants substituted from the datasheet for a Maxon Re-35 motor [33].

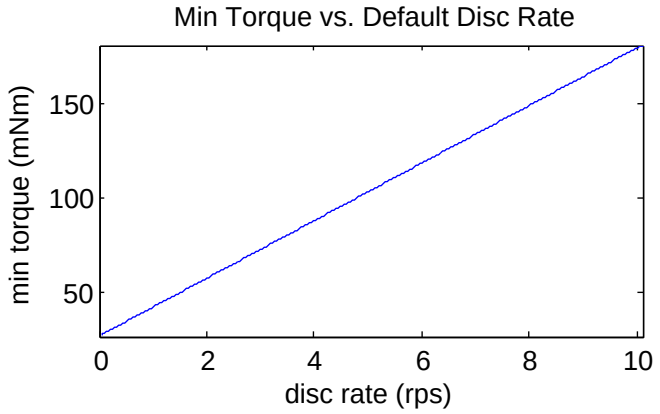


Fig. 15: Minimum torque necessary for $\mathbf{S}$ for varied ω_0. Motor constants were substituted from the datasheet for a Maxon Re-35 motor [33].

(Fig. 14) and the minimum torque grows proportionally, demanding more power from the motor (Fig. 15).

In simple terms, the faster the rate of the disc, the more difficult the Tunneling Ball problem becomes. Since the rate of the disc may be controlled via software (within the bounds of motor capabilities), we choose a motor that can operate over a range that will allow for testing, tuning, and final configuration of the device. We consider the range of between 2 and 16 rotations per second (rps).

Ideally, we would select a motor based on our calculated values for minimum torque, rate, and voltage. Motor requirement equations (20)-(21) show that these values depend on the motor parameters themselves, and that approximation is difficult. If viscosity is neglected ($b = 0$), then motor model equations (11) and (12) may be solved to yield $\tau_{\text{min}} \gg \theta_{\text{max}} K_G J / t_{\text{min}}^2$, which may offer some help in motor selection. The dependency of motor requirements on its characteristics prevents identifying a motor directly; however, specifications for a given motor can be substituted into the motor requirement equations to determine if the motor will be sufficient. The selection process then becomes a process of trial and error. The Maxon Precision Motors Re-35 DC brushed motor (Table 3) meets all of the design requirements when geared to the disc at a load-to-motor ratio of 1:2, and has a form factor on the order of the disc radius. We then calculate values for minimum torque and voltage (Table 2) using motor specifications and the range of motor rates we selected.

Two important quantities we have not considered thus far are the maximum rate ω_{stall} and maximum torque τ_{stall} that may be produced by the motor; we incorporate these quantities into trajectory planning. Position information is provided by a motor encoder, which is characterized by its rise time and resolution (counts per turn). A Hewlett-Packard HEDL 5540 (Table 4) 500 count per turn encoder is our choice, noting that we will verify sufficient resolution by simulation.

We have modeled the motor as if it were controlled by a linear gain amplifier, which is uncommon due to power efficiency and cost concerns. Instead, we use a more conventional pulse-width modulation (PWM) generator. We treat the PWM output signal as if it were the equivalent output from a linear gain amplifier, since PWM generators that produce a high-frequency output have approximately the same effect, and mathematical representations of PWM signals are cumbersome. Control systems with high-frequency closed-loop bandwidths may need more advanced modeling to understand the impact of PWM signals, in which case the reader is referred to [38] for a more precise treatment of PWM generators. Many microcontrollers feature onboard PWM generators, and when combined with an external amplification and switching circuit (commonly referred to as an 'H-bridge' [35]), are sufficient to power a motor.

We construct the drop sensor from two sequential optical sensors at a fixed altitude above the drop target. As the ball passes through the first optical sensor, the time of the event t_{drop} is recorded and compared to the time the ball passes through the second optical sensor, which is the time t_0 when the ball is known to be a fixed altitude above the disc. The initial velocity of the ball is calculated by dividing the distance between the sensors h_{drop} by the difference in time, and adding a term that represents the change in velocity due to constant gravitational acceleration for this duration:

$$v_0 = \frac{h_{\text{drop}}}{t_{\text{drop}}} + \frac{1}{2} g t_{\text{drop}}. \tag{28}$$

The optical sensors must allow a ball to pass through them, work in ambient light, and have a small rise time to reduce sensing delays. The Contrinex LGS-0030-005-502 fork through-beam sensor meets these requirements and can be mounted on an adjustable housing, allowing drop sensor accuracy to be tuned. Increasing the sensor separation increases measurement accuracy, but at the cost of shrinking the drop interval.

Our embedded computer senses drops, decodes encoder pulses, tracks disc alignment, and actuates the motor via PWM. The Luminary Micro LM3s8962 (Table 6) has ample general-purpose input/output (GPIO)

ports, 5 and 15 volt outputs for powering sensors, low interrupt latency, high-frequency PWM generation, and high-precision timers which may be useful depending on the model of computation.

4.7 Simulate & Solve

We define a disc trajectory $\hat{\theta} \in \vartheta$ to be a *correction trajectory for* β if and only if $\beta \in \mathrm{B}$ and $\mathbf{S}(\beta, \hat{\theta})$. At the time a ball drop is detected, the disc must follow a correction trajectory or the ball will impact the disc. Intuitively, the correction trajectory should match the original trajectory for all times up to and including t_0, after which the trajectory may be altered to yield a successful tunnel. Physical limitations of the motor must also be taken into account to avoid exceeding maximum rate and torque. Let $\Omega : \mathrm{B} \times \vartheta \longrightarrow \mathcal{P}(\vartheta)$ map a ball and its original trajectory to the set of correction trajectories, where $\forall t < T_d(\beta)$,

$$\begin{aligned} \hat{\theta} \in \Omega(\beta, \theta) \Leftrightarrow & \left[t < t_0 \Rightarrow \hat{\theta}(t) = \theta(t) \right] \\ & \wedge \left| b\frac{d\hat{\theta}(t)}{dt} + J\frac{d^2\hat{\theta}(t)}{dt^2} \right| < K_G \tau_{\text{stall}} \\ & \wedge \left| \frac{d\hat{\theta}(t)}{dt} \right| < K_G \omega_{\text{stall}} \\ & \wedge \ \mathbf{S}(\beta, \hat{\theta}) \end{aligned} \quad . \tag{29}$$

The above equation assumes zero tracking error, and a more thorough analysis would include the control algorithm in determining members of this set.

The Tunneling Ball problem is solvable if and only if $\Omega(\beta, \theta)$ is nonempty. Since $\Omega(\beta, \theta)$ is generally uncountable if success is possible, we cannot construct it algorithmically [15]. Mathematically deriving an optimal planning algorithm greatly simplifies the question of whether or not the problem is solvable, since if the optimal trajectory is a correction trajectory then it is executed, otherwise it and all suboptimal trajectories fail and the problem is unsolvable. This enables the embedded computer to predict failure and transition into a fault mode. Constructing an optimal planning algorithm that considers torque and rate saturation, motor dynamics, and the control algorithm is a task we leave to the field of optimization.

If tracking error is negligible, we treat the impact radius as symmetric, and we consider only fixed-rate correction trajectories, then the optimal solution to the Tunneling Ball problem is for the correction trajectory to center the disc over the drop target as the center of the ball passes through the disc (Fig. 11(a)). To construct such a trajectory, we record the disc position and rate at drop time t_0. Letting $\omega_0 = \frac{d\theta(t_0)}{dt}$, the correction angle $\theta_c : \mathrm{B} \times \vartheta \longrightarrow (-\pi, \pi]$ is

$$\theta_c(\beta, \theta) = \theta(t_0) + \omega_0 T_c(\beta). \tag{30}$$

TABLE 1: Tunneling Ball Device Dimensions

Quantity	Description	Value	Units
z_0	drop altitude	450	mm
t_{min}	min drop time	250	ms
r_b	ball radius	7.5	mm
r_{door}	disc door radius	20	mm
r_{disc}	disc radius	75	mm
r_{drop}	disc center to door center	50	mm
h	disc thickness	3	mm
ρ	disc density	1800	Kg/m^3

TABLE 2: Motor Requirements

Quantity	Description	Range	Units
ω_0	default disc rate	2 – 16	rps
		12.6 – 100.5	rad/s
ω_{min}	min disc rate	3.5 – 18.5	rps
		21.9 – 116.1	rad/s
ω_{max}	max disc rate	20.9	rps
		131.6	rad/s
τ_{min}	min motor torque	60 – 294	mNm
v_{min}	min motor voltage	2 – 15	V

TABLE 3: Maxon Precision Motors Re-3 DC Brushed Motor [33]

Quantity	Description	Value	Units
ω_{stall}	nominal speed	49.5	rps
		311	rad/s
τ_{max}	max (continuous) torque	105	mNm
τ_{stall}	max (stall) torque	493	mNm
v_{max}	max voltage	48	V
R	armature resistance	11.50	Ω
L	armature inductance	3.16	mH
b	damping coefficient	1.22	mNm/(rad/s)
K_τ	torque constant	119.00	mNm/A
K_B	back-EMF constant	0.12	V/(rad/s)
J_{armature}	armature inertia	65.5	g/cm^2
K_G	gear ratio	0.5	rad/rad

TABLE 4: Hewlett-Packard HEDL 5540 Encoder [34]

Quantity	Description	Value	Units
v_{enc}	max output voltage	5	V
K_{enc}	counts per turn	500	rot^{-1}
d_{enc}	signal rise time	180	ns

TABLE 5: Contrinex LGS-0030-005-502 Optical Sensor [36]

Quantity	Description	Value	Units
v_{beam}	output voltage	15	V
d_{beam}	output signal rise time	250	μs

TABLE 6: Luminary Micro LM3s8962 Microcontroller [37]

Quantity	Description	Value	Units
f_{micro}	max clock frequency	50	MHz
d_{micro}	input interrupt latency (at f_{micro})	240	ns
f_{pwm}	PWM frequency	20	KHz
q_{dac}	PWM quantization level size	0.04	%
d_{dac}	PWM output latency	2	μs

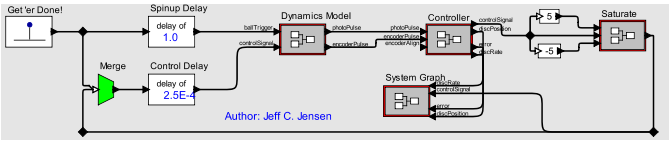


Fig. 16: Top-level view of the Tunneling Ball Device in Ptolemy II.

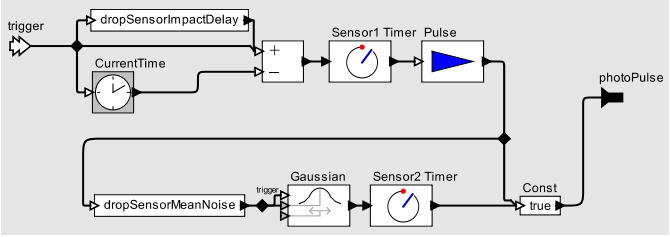


Fig. 17: Ball drop simulator in Ptolemy II.

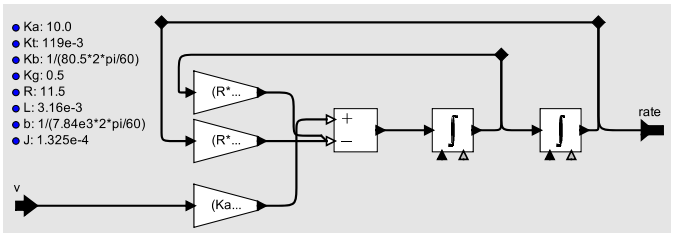


Fig. 18: Continuous motor model in Ptolemy II.

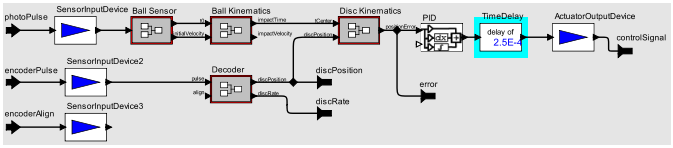


Fig. 19: PTIDES controller in Ptolemy II. This is the top-level actor for code synthesis.

This angle must be adjusted in $T_c(\beta)$ seconds, yielding the correction rate $\omega_c : \mathrm{B} \times \vartheta \longrightarrow \mathbb{R}$

$$\omega_c(\beta,\theta) = \begin{cases} -\frac{\theta_c(\beta,\theta)}{T_c(\beta)} & \text{if } |\theta_c(\beta,\theta)| \leq \frac{\pi}{2} \\ \frac{\pi\,\mathrm{sign}(\theta_c(\beta,\theta)) - \theta_c(\beta,\theta)}{T_c(\beta)} & \text{if } |\theta_c(\beta,\theta)| > \frac{\pi}{2}. \end{cases} \quad (31)$$

The planning algorithm $\Psi : \mathrm{B} \times \vartheta \longrightarrow \vartheta$ that produces the trajectory is

$$(\Psi(\beta,\theta))(t) = \begin{cases} \theta(t) & \text{if } t < t_0 \\ \theta(t_0) + [\omega_0 + \omega_c(\beta,\theta)]\,t & \text{if } t \geq t_0, \end{cases} \quad (32)$$

which is the correction trajectory with constant rate closest to ω_0 that centers the ball in a door, indicated by $(\Psi(\beta,\theta))(T_c(\beta)) = 0$ or π. Comparing the codomain of Ψ to the set of all correction trajectories,

$$\Psi(\beta,\theta) \in \Omega(\beta,\theta) \Leftrightarrow |\omega_0 + \omega_c(\beta,\theta)| < \min\{\omega_{\max}, \omega_{\text{stall}}\}, \quad (33)$$

a condition that is both computable and easily verified. An advantage of a fixed-rate trajectory is that it is easy for the control algorithm to track, since the control voltage approaches a constant value and is directly proportional to the disc rate after transients have decayed.

Turning to simulation, the Simulink modeling environment enables us to simulate nonlinear operations such as quantization, saturation, and sampling. These operations are difficult to model mathematically; however, simulating their effects is straightforward. Our Simulink model for position control of the motor (Fig. 12) incorporates the motor transfer function, sampling rate and quantization of a digital controller, latency and quantization of the PWM generator, and voltage saturation.

We model the entire end-to-end heterogeneous system in Ptolemy II, making use of its DE, PTIDES, and Continuous models of computation. The top-level view (Fig. 16) shows a drop generator, a controller, and physical dynamics actors connected in a feedback loop. Ball drops are simulated by the model in Fig. 17, the Continuous model of the motor is shown in Fig. 18, and the top-level view of the controller is shown in Fig. 19.

4.8 Construct

We construct a housing to enclose the motor, disc, and dropped balls. As we drew schematics for constructing the device (Appendix B), design tradeoffs were easily considered by changing simulation parameters.

4.9 Synthesize Software

We automatically generate software lookup tables for ball and disc kinematics for embedded software via MATLAB scripting. We augment our PTIDES model with delay and latency values for each hardware component.

PTIDES models in Ptolemy II are generated into C code and statically linked against the PtidyOS [12] operating system. PtidyOS is unlike embedded operating systems like OpenRTOS [44], vxWorks [45], and RTLinux [46] which use conventional threaded programming models. PtidyOS instead performs all event processing and context switching in interrupt service routines, minimizing latency between software components and dramatically reducing the number of context switches. PtidyOS follows the event-order execution semantics of PTIDES with traditional scheduling methods such as earliest deadline first to guarantee optimal scheduling when a feasible schedule exists. A simplified PtidyOS application development cycle is shown in Fig. 20, and reflects many of the elements of MBD discussed here.

4.10 Test

Sensors, actuators, electrical components, and software control behave as expected. We tested the drop sensor and found it to correctly detect a ball and its initial velocity. At the time of this report, we observed significant friction between the disc, gears, axles, and motor, prompting us to revisit the physical modeling step. The motor stalls due to static friction for input voltages less than 6% of $v_{\max}$, suggesting the control algorithm should incorporate modal behavior. Mechanical friction is audible and appears to be rate-dependent. The friction is nonlinear and difficult to model. We are using

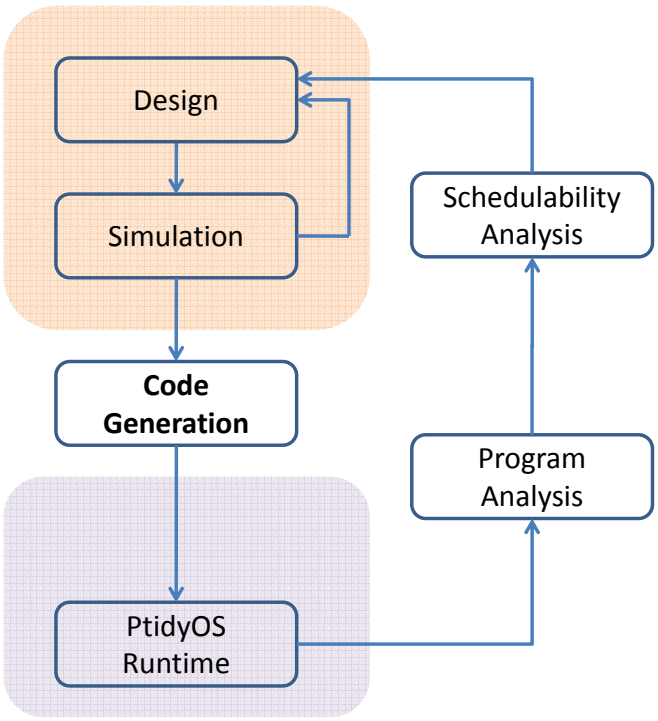


Fig. 20: PtidyOS application development cycle in Ptolemy II. *Source: Jia Zou, University of California, Berkeley.*

data acquisition devices to experimentally characterize the motor plant and to revise simulation models to determine controllability. If the system in its current state is controllable, then we will synthesize new control parameters; if not, we will consider machining options to reduce mechanical friction.

The PID control parameters yield reasonable tracking and steady-state error, but the rise time exceeds the minimum drop time $t_{\min}$. As the Tunneling Ball problem is not solvable under these conditions, we have not conducted tests involving ball drops.

5 Conclusion

5.1 Case Study: Cal Climber

The problem statement for the Cal Climber was simple, though arguably incomplete. When we posed the problem to students in an introductory course in embedded systems, we were asked, "what should the robot do when it reaches the top of the hill?"

The embedded computer does not support floating point operations, and the traditional measurement for tilt requires calculation of an inverse tangent. We derived an alternate solution by revisiting the modeling of the physical processes. We itereated the problem characterization step a second time when the system was rendered unstable due to the lack of lowpass filters and rapid changes in wheel speeds. The control algorithm used to solve the problem is a relatively simple calculation and worked well in an integer-only environment.

Given the flexibility of our design task, selecting a model of computation was not as critical of a step, and we were guided largely by convenience, though statecharts were attractive and may have been used had a suitable code synthesizer been available. Hardware selection was straightforward, though a custom printed circuit board was required to power the embedded computer from the robot battery.

Simulation was extremely helpful in designing embedded software, and LabVIEW supports live debugging of the system including the ability to update the front panel (Fig. 3) directly from the robot sensors. Simulation is what led us to discover the unstable equilibrium point of the control algorithm. Utilizing the display on the embedded controller proved useful, since accelerometer feedback could be displayed numerically and on text sliders, enabling comparison between the sampled accelerometer values and the expected values produced in simulation. Construction took only a few hours, with the most careful task being the alignment of the accelerometer to coincide with the prescribed coordinate system. Had the alignment not been precise, calibration may have been added to the embedded software.

LabVIEW reliably synthesized, compiled, and programmed the software model into embedded C code. In testing, the robot could climb a hill at full speed, avoiding objects encountered along the way and without driving off of a cliff (provided the cliff sensors were able to see the cliff). The robot could even be placed on a table that was held by a person at each corner, and as the table was rotated in three dimensions, the robot climbed towards the highest point. This quickly evolved into a game that resembled the arcade game Pong [47].

We compared the behavior of the Cal Climber as programmed by hand and by synthesized code. The behavior was indistinguishable in most cases. On rare occasions, the robot programmed by handwritten code would deviate from its control algorithm and spin around uncontrollably. This behavior was unexpected (and rather surprising in a late-night laboratory environment), and we later determined it to be a concurrency issue relating to serial communication. In a mode where the embedded computer asynchronously requested sensor packets from the robot, a timed interrupt sent a packet request over the serial port while the main program loop regularly broadcast wheel speeds; when properly interleaved, the main program loop would begin sending the wheel speed broadcast and be interrupted so that the sensor update request was injected into the communication stream. The first few bytes of the sensor request were numerically large, and when injected into the communication stream, instructed the robot to drive one of its wheels at full speed. The handwritten code was revised to use a timed interrupt to flag when an update request should be sent, and the request itself was generated within the main program loop, serializing the two communications. This was not optimal, since a sensor request could be delayed by program execution time. We did not observe this bug when executing code synthesized from LabVIEW.

Challenges in code synthesis arose in development of the Cal Climber, the most notable being code size. The embedded computer has a conservative amount of memory, and the LabVIEW code generator relies heavily on a real-time operating system with a large memory footprint. Linking to floating point libraries or enabling parallel execution produced code that would not fit in the flashable memory space of the embedded computer.

Low-level functionality such as configuring ADC rate, supporting hardware interrupts, and buffer sizes, were hidden by the LabVIEW dataflow abstraction. Drivers for writing to the display on the embedded computer forced the software to block for 50ms, which resulted in serial communication buffers overflowing. We resolved this problem by using an inline-C node in LabVIEW to reference the C-code libraries provided by the manufacturer. While these challenges were difficult to isolate given the abstraction and the bulk of the generated code, dataflow semantics were faithfully executed, so concurrency issues did not arise.

5.2 Case Study: The Tunneling Ball Device

The problem statement for the Tunneling Ball Device was ambiguous about the meaning of 'sporadic,' though we gave a more precise definition in the problem characterization. The requirement that only one ball be in the air at any point in time may be relaxed, though we have yet to consider this.

The modeling of the physical processes at play was time consuming but ultimately necessary, as early simulation gave insight into the physical quantities of interest, error tolerance, and overall difficulty of the problem. Processes we did not consider include drag on the ball in freefall, and perhaps of greater importance, grinding and slipping of gears and friction of axles against their bearings.

Characterizing the Tunneling Ball problem proved the most challenging design aspect, and we were heavily aided by co-iterating with the simulation step. Deriving minimum quantities from the physical dynamics was difficult and required solving a system of linear nonhomogenous differential equations, for which initial conditions were crucial. It was not until we rigorously defined trajectories and planning algorithms in terms of their domains, codomains, and trajectory transformations that we found the correct set of initial conditions that yielded solutions consistent with real-world values. MATLAB calculations often caught errors in sign or values whose magnitude was not practical. A more thorough description of the success predicate might be found through the use of symbolic equation tools, such as Maple [48] or Mathematica [49].

MATLAB scripting was useful when were tuned adjustable parameters such as ball radius, tunnel radius, disc size, and drop altitude. We performed simulations of tunnel window, impact window, and the success predicate. We replicated every equation used to describe the physical processes to ensure calculations agreed with our the understanding of the problem.

In practice, the selection of a model of computation is often based on available tools without regard to the physics of the system, characterization of the problem, or distribution of computations. The signals present in the Tunneling Ball Device clearly aligned with PTIDES or DE, and choosing an incompatible model of computation would have introduced significant complexity in application design. A more thorough modeling of computational processes would incorporate worst-case execution time and sensor to actuator latencies of the software, both of which can be represented in PTIDES.

Rigorous simulations verified calculations and demonstrated the system in its entirety, often catching errors in equations or designs. One such example is that an initial model of our motor omitted viscosity, an error we caught when simulations showed the disc spinning at a constant rate with no voltage applied. We verified tracking error and rise time by simulation. In solving the Tunneling Ball problem, we may have been able to derive optimal planning algorithms from Maple or Mathematica, though we utilized neither.

5.3 Summary and Future Work

We developed two cyber-physical systems using Model-Based Design (MBD). The Cal Climber became the topic of an industry case study on embedded systems design from the LabVIEW environment [50], and has been adopted by several universities as a laboratory embedded systems platform. The Tunneling Ball Device presented us with far more difficult modeling, machining, and software constraints. While modeling proved essential in devising and realizing the device, it is not yet capable of solving the problem at hand, a failure that recapitulates an earlier point: *no model can ever be complete*.

The models for the Tunneling Ball Device are informative, albeit incomplete as they fail to capture the friction that was introduced by imprecise machining. The Center for Hybrid and Embedded Software Systems at the University of California, Berkeley, continues to investigate enhancements to the device including high-precision machining to reduce friction and experimental characterization of the plant for use in simulation and PID tuning.

MBD methodology proved critical in nearly every aspect of the development of these cyber-physical systems, especially in device construction, hardware selection, and selection of the model of computation. The development of each device prompted iteration between design steps in distinct ways, displaying the usefulness and expressiveness of the methodology. The elements of MBD invoke powerful modeling theory in a strongly pedagogical, combined application of mathematics, engineering, and computer science. Each of the elements investigated in this report is only a preview of a vast field

of research, but the formalization into codependent steps offers an innovative approach to the design of cyber-physical systems.

ACKNOWLEDGMENTS

The Ptolemy II framework has been developed for many years by open source developers around the world and researchers from the Center for Hybrid and Embedded Systems at the University of California, Berkeley. The first version of PtidyOS was written by Jia Zou and Shanna-Shaye Forbes in Spring 2009, and extensions were written by Jia Zou, Slobodan Matic, Jeff C. Jensen, and Isaac Liu. The expertise of the University of California, Berkeley, Department of Electrical Engineering and Computer Science Electronics Support Group was instrumental in the construction of both the Cal Climber and the Tunneling Ball Device. Construction of the Cal Climber was overseen by Professor Sanjit A. Seshia, and the research and development leading to this report was advised by Professor Edward A. Lee, at the University of California, Berkeley.

APPENDIX A
SCHEMATICS OF THE CAL CLIMBER

Fig. 21 shows the wiring of the iRobot Create, Luminary Micro LM3s8962 embedded computer, Analog Devices ADXL-322 accelerometer, and optional SparkFun Electronics BlueSMiRF [51] Bluetooth serial modem. Schematic for the iRobot Create carbo bay connector was taken from the iRobot Create Manual [27], schematic for the Luminary Micro was taken from the LM3s8962 datasheet [37], schematic for ADXL-322 accelerometer was taken from the ADXL-322 datasheet [28], and image of the BlueSMiRF Bluetooth radio was taken from the BlueSMiRF datasheet [51].

APPENDIX B
SCHEMATICS OF THE TUNNELING BALL DEVICE

Schematics for the Tunneling Ball Device are shown in Fig. 22 – 28. All measurements are in millimeters or degrees, and all components are drawn to scale.

The device comprises a drop basin (Fig. 22 – 23), which is the base of tunneling ball device, a motor housing, and a sensor housing (Fig. 27 – 26). The drop basin is composed of a floorboard which collects dropped balls and returns them to a bin, a topboard (Fig. 25) which prevents balls which have impacted the disc from leaving the basin, and a disc (Fig. 28). The housing material is made of clear 10mm polycarbonate, and the disc of 3mm G10F4 garolite.

The motor shaft is rigidly connected to a metal gear to take advantage of gearing ratio to the load and to prevent balls impacting the disc from applying significant torque to the motor shaft. Axles are fit into cylindrical bearings mounted in the housing material.

Schematics for the optical sensors were provided from the Contrinex Fork Through-Beak Sensors datasheet [36], schematics for the motor were taken from the Maxon Precision Motors, “Re 35, 35mm, Graphite Brushes, 90 Watt,” datasheet [33], and schematics for the encoder were taken from the Hewlett-Packard, “Encoder HEDL 5540, 500CPT, 3 Channels” datasheet [34].

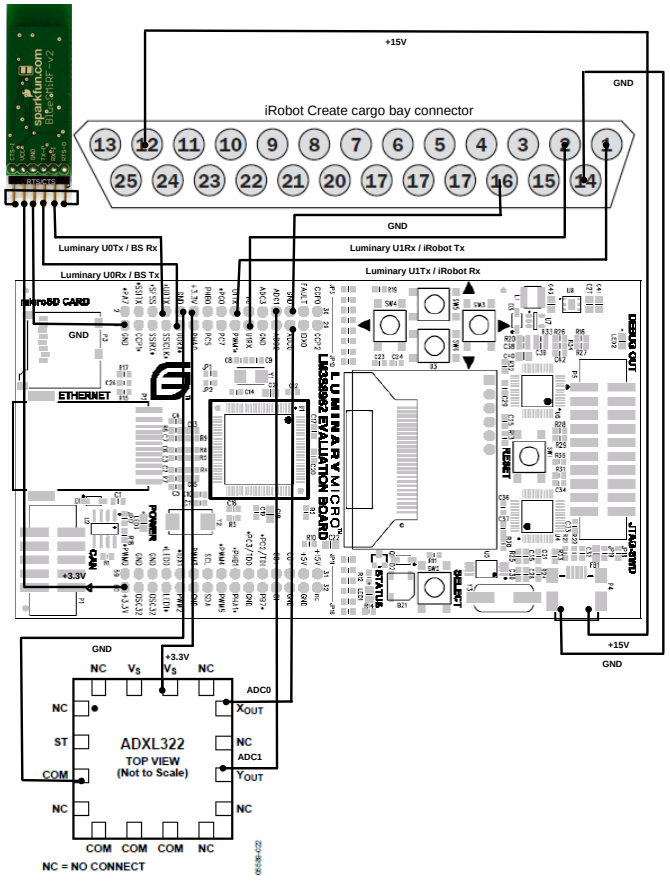


Fig. 21: Wiring diagram for the Cal Climber

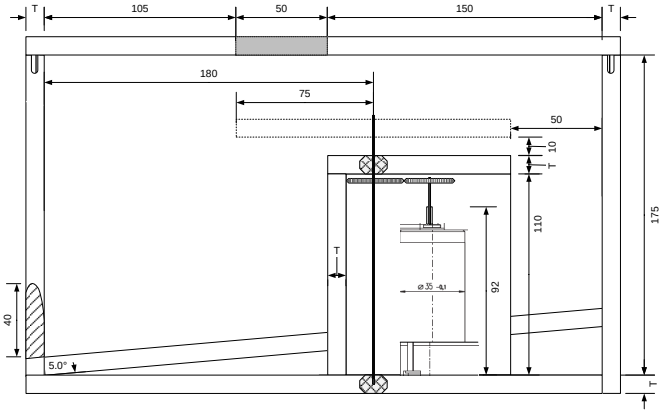


Fig. 23: Tunneling Ball Device drop basin side view.

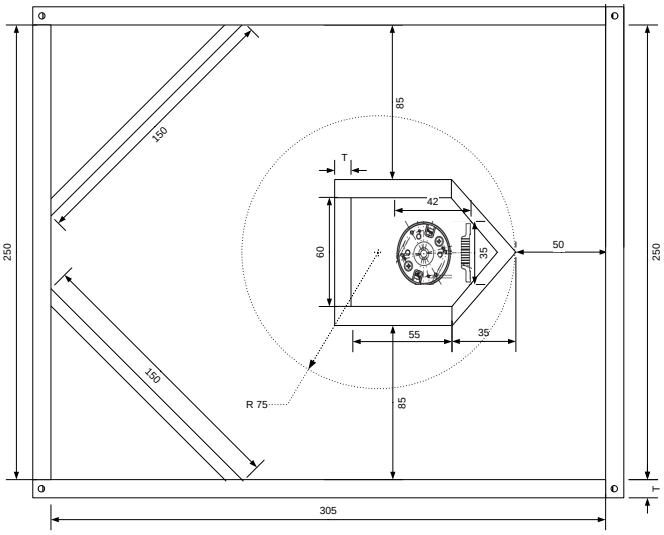


Fig. 22: Tunneling Ball Device drop basin top view.

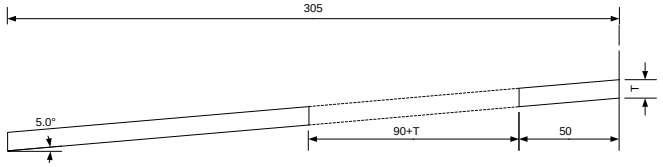


Fig. 24: Tunneling Ball Device floorboard side view.

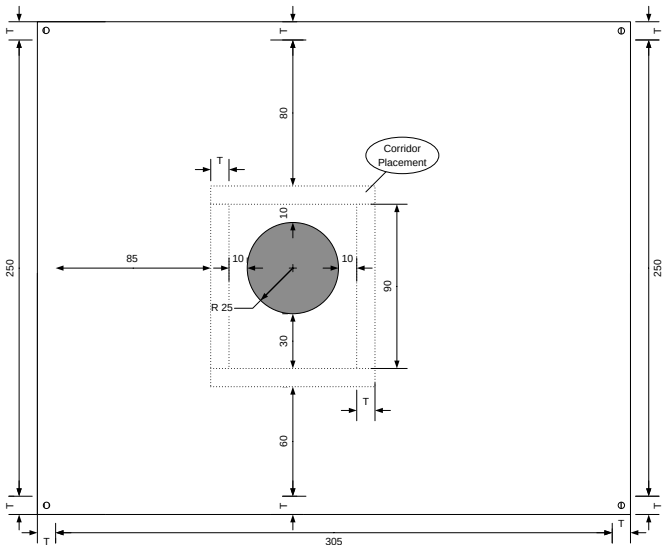


Fig. 25: Tunneling Ball Device topboard top view.

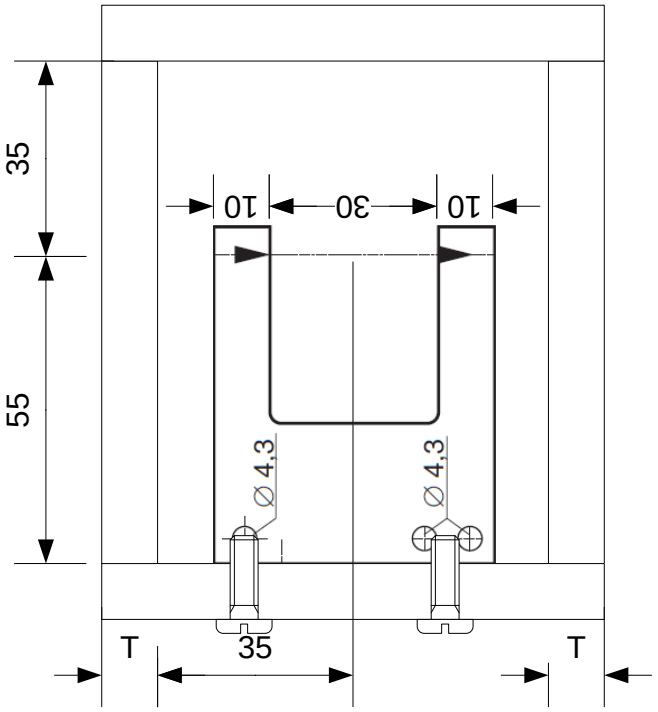


Fig. 27: Tunneling Ball Device drop sensor top view.

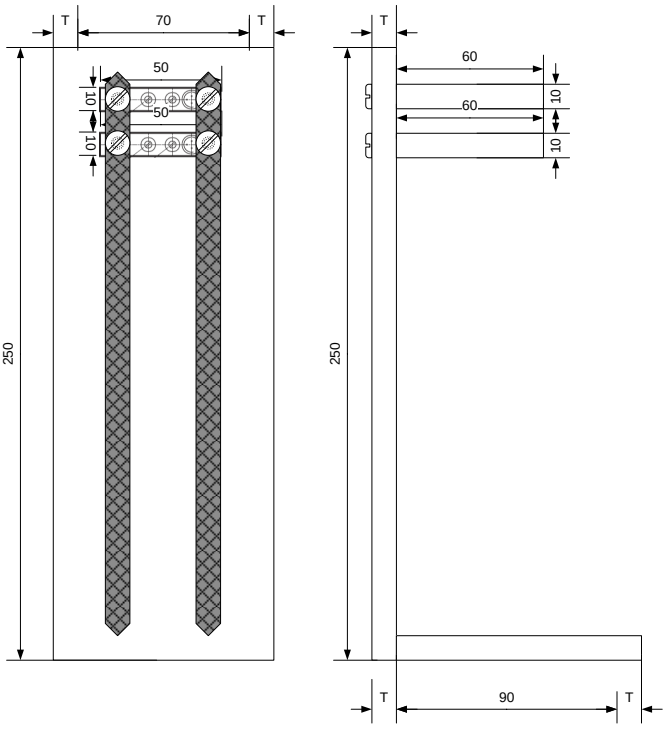


Fig. 26: Tunneling Ball Device drop sensor side view.

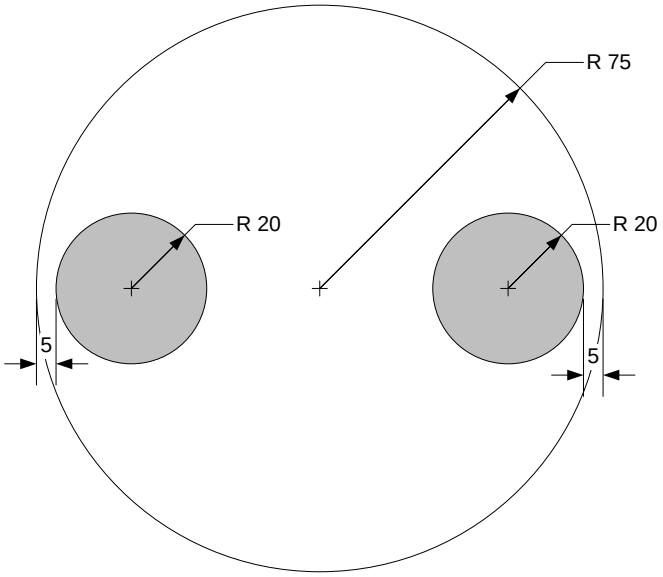


Fig. 28: Tunneling Ball Device disc.

REFERENCES

[1] E. Lee, "Cyber Physical Systems: Design Challenges," in International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), May 2008.

[2] C. Brooks, C. Cheng, T. Feng, E. Lee, and R. von Hanxleden, "Model Engineering Using Multimodeling", in 1st International Workshop on Model Co-Evolution and Consistency Management (MCCM 08), September 2008.

[3] K. Balasubramanian, A. Gokhale, G. Karsai, J. Sztipanovits, and S. Neema, "Developing Applications Using Model-Driven Design Environments," *IEEE Computer*, vol. 39, no. 2, pp. 33, February 2006.

[4] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty, "Model-Integrated Development of Embedded Software," in Proceedings of the IEEE, vol. 91, no. 1, January, 2003.

[5] E. Lee, "The Problem with Threads," *IEEE Computer*, vol. 39, no. 5, pp. 33-42, May 2006.

[6] G. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 2nd ed. Berlin: Springer, 2005.

[7] LabVIEW, *LabVIEW User Guide*, National Instruments, June 2009.

[8] Simulink, Control Design, *Simulink Control Design 3 User's Guide*, The MathWorks, Inc., September 2009.

[9] C. Hylands, E. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, Y. Zhao, and H. Zheng, "Overview of the Ptolemy Project," University of California, Berkeley, Technical Memorandum. UCB/ERL M03/25, July 2003.

[10] MATLAB, *MATLAB 7 User's Guide,* The MathWorks, Inc., September 2009.

[11] G. Fishman, *Discrete-Event Simulation: Modeling, Programming, and Analysis*. Springer-Verlag, 2001.

[12] P. Derler, T. Feng, and E. Lee, "PTIDES: A Programming Model for Distributed Real-Time Embedded Systems," University of California, Berkeley, EECS Technical Report. EECS-2008-72, May 2008.

[13] E. Lee, "Computing needs time," *ACM Communications*, vol. 52, no. 5, pp. 70-79, May 2009.

[14] J. Eidson, E. Lee, S. Matic, S. Seshia, and J. Zou, "Time-centric Models for Designing Embedded Cyber-Physical Systems," University of California, Berkeley, Technical Memorandum. UCB/EECS-2009-135, October 2009.

[15] N. Cutland, *Computability: An Introduction to Recursive Function Theory*. Cambridge, MA: Cambridge University Press, 1997, pp. 100-112, 149-156.

[16] G. Berry, "The Foundations of Esterel," in *Proof, Language, and Interaction: Essays in Honour of Robin Milner*. Cambridge, MA: MIT Press, 2000.

[17] P. Caspi, D. Pilaud, N. Halbwachs, J. Plaice, "LUSTRE: A Declarative Language for Real-Time Programming," in Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, January 23rd 1987, pp. 178-188.

[18] D. Harel and A. Naamad, "The STATEMATE semantics of statecharts," *ACM Transactions on Software Engineering Methodology*, vol. 5, no. 4, pp. 293-333. October 1996.

[19] E. Lee and D. Messerschmitt, "Synchronous Dataflow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235-1245, September 1987.

[20] A. Benveniste and G. Berry, "The Synchronous Approach to Reactive and Real-Time Systems," *Proceedings of the IEEE*, vol. 79 no. 9, September 1991.

[21] J. Shearer, B. Kulakowski, and J. Gardner, *Dynamic Modeling and Control of Engineering Systems*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 1997, pp. 49-72.

[22] A. Sangiovanni-Vincentelli and G. Martin, "Platform-Based Design and Software Design Methodology for Embedded Systems," *IEEE Design and Test of Computers*, vol. 18, no. 6, pp. 22-33. December 2001.

[23] J. Eker, J. Janneck, E. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming Heterogeneity - The Ptolemy Approach," *Proceeedings of the IEEE*, vol. 91, no. 1. January, 2003.

[24] Simulink Design Verifier, *Simulink Design Verifier 1.5 User Guide,* The MathWorks, Inc., Natick, MA, September 2009.

[25] Eclipse Modeling Project, "The Eclipse Modeling Framework (EMF) Overview," *Eclipse Foundation*, June 2005. Available: http://www.eclipse.org. [Accessed: February 1st 2010].

[26] S. Sendall and W. Kozaczynski, "Model Transformation: The Heart and Soul of Model-Driven Software Development," *IEEE Software*, vol. 20, no. 5, pp. 42-45. October 2003.

[27] iRobot, *iRobot Create Owner's Guide,* iRobot Inc., January 2006.

[28] Analog Devices, "Small and Thin ±2g Accelerometer" ADXL322 Datasheet rev. 0, January 2007.

[29] H. Young, *Sears & Zemanskys University Physics: with Modern Physics*, 11th ed. San Francisco, CA: Pearson, 2004, pp. 58-60, 178-181, 334, 345-346.

[30] S. Shinners, *Modern Control System Theory and Design*, 1st ed. New York, NY: Wiley Interscience, 1992, pp. 143-159, 256-258.

[31] E. Corinaldesi, *Classical Mechanics for Physics Graduate Students*. Boston, MA: World Scientific Publishing, 1999, pp. 12-13.

[32] W. Rudin, *Principles of Mathematical Analysis*, 3rd ed. New York, NY: McGraw-Hill, 1976, pp. 107-108.

[33] Maxon Precision Motors, "Re 35, 35mm, Graphite Brushes, 90 Watt," 273759 datasheet. May 2009.

[34] Hewlett-Packard, "Encoder HEDL 5540, 500CPT, 3 Channels" datasheet. May 2009.

[35] Signal Consulting LLC, "Si20HPB4-50V-20A-HB2 High Power H-Bridge" datasheet.

[36] Contrinex, "Fork Through-Beam Sensors" LGS-0030-005-502 datasheet.

[37] Texas Instruments, "LM3s8962 Microcontroller" datasheet. July 2008.

[38] B. Axelrod, Y. Berkovich, and A. Ioinovici, "A new dynamic discrete model of DC-DC PWM converters," *HAIT Journal of Science and Engineering*, vol. 2, no. 3-4, pp. 426-451. July 2005.

[39] R. Kelly and J. Moreno, "Learning PID Structures in an Introductory Course of Automatic Control," *IEEE Transactions on Education*, vol. 44, no. 4. November, 2001.

[40] K. Keutzer, A.R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System-Level Design: Orthogonalization of Concerns and Platform-Based Design," *IEEE Transactions*, vol. 19, no. 12. December 2000.

[41] S. Baruah, A. Mok, and L. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in 11th Real-Time Systems Symposium (IEEE). December 1990.

[42] C. Phillips and R. Harbor, *Feedback Control Systems*, 4th ed. Upper Saddle River, NJ: Prentice Hall, 2000, pp. 116-124.

[43] H. Nyquist, "Certain Topics in Telegraph Transmission Theory," *AIEE Transmissions*, vol. 47, pp. 617-644. April 1928.

[44] The FreeRTOS.org Project, "OpenRTOS," *The FreeRTOS.org Project*. Available: http://freertos.org. [Accessed February 1st 2010].

[45] Wind River, "VxWorks: Embedded RTOS with support for POSIX and SMP," *Wind River*. Available: http://www.windriver.com/products/vxworks. [Accessed February 1st 2010].

[46] Wind River, "RTLinuxFree," *Wind River*. Available: http://www.rtlinuxfree.com. [Accessed February 1st, 2010].

[47] *Pong* [Arcade]. Atari Inc., 1972.

[48] Waterloo Maple Inc., "Maple", *Waterloo Maple Inc.*. Available: http://www.maplesoft.com. [Accessed February 1st 2010].

[49] Wolfram Research, Inc., "Mathematica", *Wolfram Research*. Available: http://www.wolfram.com. [Accessed February 1st 2010].

[50] J. Brettle and J. Jensen, "UC Berkeley Teaches Embedded Systems with NI LabVIEW Embedded Module for ARM Microcontrollers," *National Instruments*, April 2009. Available: http://sine.ni.com/cs/app/doc/p/id/cs-12246. [Accessed February 1st 2010].

[51] SparkFun Electronics, "BlueSMiRF Gold" RN-v1 datasheet. February, 2008.

Jeff C. Jensen is a graduate student in the Department of Electrical Engineering & Computer Science and a member of the Center for Hybrid and Embedded Software Systems at the University of California, Berkeley. He holds an A.A. in General Science and an A.A. in Liberal Arts from Santa Monica College, and a B.S. in Electrical Engineering & Computer Science from the University of California, Berkeley. Research interests include model-based design, models of computation, and cyber-physical systems.