PICL: Portable In-Circuit Learner

Adam Fourney, Michael Terry
Cheriton School of Computer Science
University of Waterloo
{afourney, mterry}@cs.uwaterloo.ca

ABSTRACT

This paper introduces the PICL, the portable in-circuit learner. The PICL explores the possibility of providing standalone, low-cost, programming-by-demonstration machine learning capabilities to circuit prototyping. To train the PICL, users attach a sensor to the PICL, demonstrate example input, then specify the desired output (expressed as a voltage) for the given input. The current version of the PICL provides two learning modes, binary classification and linear regression. To streamline training and also make it possible to train on highly transient signals (such as those produced by a camera flash or a hand clap), the PICL includes a number of *input inferencing* techniques. These techniques make it possible for the PICL to learn with as few as one example. The PICL's behavioural repertoire can be expanded by means of various *output adapters*, which serve to transform the output in useful ways when prototyping. Collectively, the PICL's capabilities allow users of systems such as the Arduino or littleBits electronics kit to quickly add basic sensor-based behaviour, with little or no programming required.

Author Keywords

Toolkits; Machine Learning; Circuits; Tangible User Interfaces

ACM Classification Keywords

H.5.m. Information Interfaces and Presentation (e.g. HCI): input devices and strategies; interaction styles; prototyping

General Terms

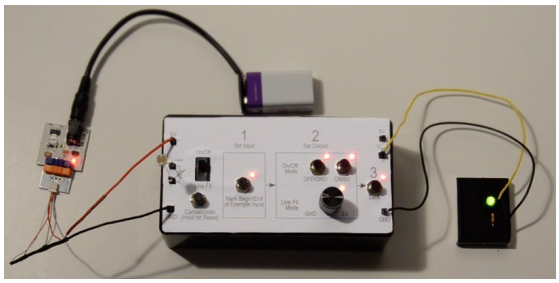
Design, Human Factors

INTRODUCTION

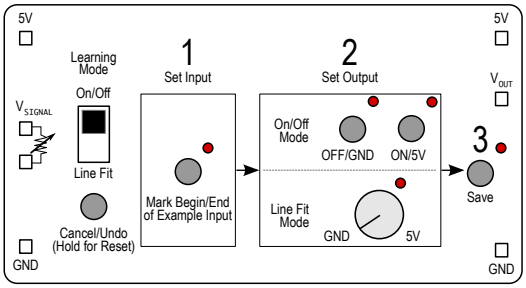
The move from desktop to ubiquitous computing has led to the development of a variety of hardware-based toolkits to aid with prototyping and design. These toolkits range from those that depend on computers for both programming and deployment (e.g., Phidgets [10] and d.tools [14]), to those that are completely standalone, such as littleBits [4, 5], a recent hardware platform with individual pieces that snap together. Each of these toolkits lowers the barrier to prototyping hardware-based interfaces by reducing the time and effort required to work with physical sensors, actuators, and electrical circuits in general.

In recent years, interest in machine learning toolkits has also increased. Some of these toolkits are designed to be relatively independent of problem domains (e.g., Weka [11]), while others, such as Exemplar [13] or Crayons [7], are intended to aid in building user interfaces that respond to real-world sensor data. A hallmark of these latter toolkits is that they enable a form of programming-by-demonstration [18] by virtue of allowing the designer to demonstrate desired input.

Inspired by this past research, we envision a more complete integration of these two research trajectories whereby ma-



(a) The PICL prototype in a simple circuit involving a battery, a photocell, and an LED output.



(b) A rendering of the PICL's user interface. The interface consists of five momentary buttons, one rotary dial, one switch, and five small LEDs for feedback. Users attach sensors to the V_{SIGNAL} inputs on the left side. Sensors outputting a voltage connect directly to the top V_{SIGNAL} input port; resistive-based sensors are attached to V_{SIGNAL} and the input directly below it, which attaches internally to a voltage divider.

Figure 1. The PICL prototype.

UIST'12, October 7–10, 2012, Cambridge, Massachusetts, USA.

chine learning capabilities are reified in portable, standalone learning hardware that can integrate with circuits, hardware toolkits, and other computational devices. These learning units should allow designers and developers to attach arbitrary sensors, then teach the learning hardware how to respond when given specific types of input. Ideally, we would like to achieve this goal without requiring a computer for either programming or deployment.

As a step towards this larger vision, we introduce the PICL, the Portable In-Circuit Learner (Figure 1(a)). The PICL is an in-circuit learning unit that accepts arbitrary analog sensors (resistive-based or voltage-based) as input. The sensed input is then mapped by the PICL to an output voltage via a learned function. End-users train the PICL by providing examples of input and the desired output for each input example.

Our current PICL can learn either a linear classifier or a linear regression model. It outputs 0V or 5V for the linear classifier, and a voltage in the range of 0-5V for the linear regression learning mode. These learning algorithms continuously sample input at a rate of 1200Hz on the 16MHz Atmel processor used in our PICL, yielding a 0.83ms response time. These two learning modes, operating at this response rate, enable one to prototype a wide variety of systems without any programming.

To simplify training, the PICL includes a number of *input inferencing* techniques. These input inferencing techniques reduce the number of examples the user must provide when training the PICL, and also infer which parts of an input signal are likely of interest. As an example, consider a scenario whereby the user wishes to turn on a light by clapping their hands. The PICL's linear classifier can be used in this case, but it must be trained on the short-lived, larger amplitudes of the clap's sound wave so it does not accidentally activate at normal, ambient sound levels. To aid with this type of training, the PICL tracks the initial, minimum, and maximum sensor values when it is training. It then chooses the value that leads to the best separation of the training data. This mode of input inferencing is useful when training on transient events, like the flash of a camera, or the previous hand clapping example. It also allows the PICL to be trained without requiring a separate graphical user interface to specify which parts of the signal are of interest.

To further facilitate rapid prototyping, the PICL also includes a number of *output adapters* that transform the output voltage in useful ways. These output adapters are separate components that connect to the PICL. For example, we provide a "toggle" output adapter that acts as a basic flip-flop. The output adapters build on, and extend, previous work exploring programming with tangible eBlocks [19].

The specific contributions of this work are as follows:

- An in-circuit, standalone machine learning hardware component with two learning modes, linear classification and linear regression.
- A programming-by-demonstration style of training that includes a set of input inferencing techniques to minimize training time, and to support training on transient events.
- A series of output adapters that transform the output in meaningful ways, to address a number of common situations and needs when building systems.
- A simplified interface for attaching sensors. Specifically, the PICL includes a built-in voltage divider to allow direct connection of resistive sensors. Active, voltage-based sensors can also be directly attached.

PICLs are aimed at designers, developers, and hobbyists who use platforms such as Arduino or littleBits to develop sensor-based systems. The PICL reduces and simplifies the code necessary to react to sensed input, while eliminating the need to measure sensor values, then write code that reacts to those sensor values. Its use of machine learning algorithms also provides a measure of error tolerance, allowing noisy or imperfect training data. Finally, for users of high-level systems like littleBits, PICLs provide a way to add intelligence to a snap-together system, without requiring programming.

In the rest of this paper, we review related work, then describe the design and implementation of the PICL. We present a set of sample applications, and conclude with a discussion of limitations and possibilities for future work.

RELATED WORK

The PICL draws upon prior research in hardware toolkits; programmable, tangible interfaces; and machine learning toolkits.

Numerous hardware toolkits have been produced to enable rapid prototyping of systems that can sense and interact with the physical world. Phidgets [10] encapsulate sensors and actuators into USB-modules that can be accessed via a rich programming API. In a similar vein, iStuff [3] and the Calder Toolkit [17] attach microcontrollers to physical devices, such as switches and potentiometers, to provide a higher-level interface to these low-level components. d.tools [14] takes an analogous approach, providing a standardized interface for low-level components to attach to a computer. It also includes a software environment to further facilitate programming with the physical sensors. The recent .NET Gadgeteer system [26] provides a similar modular system of sensors, actuators, and a mainboard, and can be programmed using the .NET language. Like these hardware toolkits, the PICL shares the goal of lowering the barrier to designing interactive systems with real-world sensors and actuators, but differs from these previous projects by not requiring a computer for programming or deployment.

A number of embedded systems have been developed that require a computer for programming, but which can otherwise be deployed as self-standing units. Smart-Its [9] are a general purpose development platform for creating context-aware applications using embedded systems. Smart-Its consist of a

core board and sensor board, allowing flexibility in the sensors used. The Arduino [1] provides a basic platform for prototyping embedded systems, though it requires basic knowledge of electronics to attach sensors and actuators. Like these systems, the PICL is a self-standing system that can accept arbitrary sensor inputs. However, it is not intended to be a general-purpose system like these other platforms. Instead, its sole function is to learn input-output mappings.

The recent littleBits architecture [4, 5] allows various sensors, controls, and actuators to be chained together, with sensors and controls directly affecting actuators that come after them in the chain. This architecture enables those with no prior electronic experience to snap together basic circuits, but at present, it offers nothing more than AND and OR gates for adding computation to the circuits. The PICL provides a natural way to add intelligence to the littleBits system. We describe this possibility more fully later, and in the accompanying video figure.

The Buttercup system [21] is an embedded system that maps sensor input to effector output via an input-output curve. Users configure the Buttercup on a PC, where they can manually adjust the input-output curve or choose from predefined curves. The PICL performs a similar input-output mapping, but in contrast to Buttercup, PICL users create this mapping by providing example input and the corresponding desired output. The examples are then interpreted in the context of the chosen learning mode (i.e., the linear classifier or linear regression mode).

Research in tangible computing [16] investigates how computation can be embedded in physical objects. In the spirit of tangible user interfaces, a number of research projects have explored how physical objects can be used for programming. For example, AlgoBlocks [24] creates a "tangible programming language" with each block representing a programming construct in a Logo-like language. McNerey's Tangible Programming Bricks [20] provide similar capabilities, allowing individuals to compose basic programs by snapping Lego blocks together, with each Lego block representing a specific operation. While the PICL does not support the traditional form of programming represented by these systems, it does nonetheless allow users to specify desired behavior through demonstration.

More similar to the PICL, eBlocks [19] provide sensor-based blocks that transform their input to logic levels. The output of these blocks can then be connected to logic blocks to perform Boolean logic. Also included in this system are state-based blocks, such as flip-flops (called a "Toggle" block in the system), a "Yes Prolonger" block (which extends the emission of a "yes"/true-valued input), a latch block, and a pulse generator block. The PICL's output adapters are analogous to these state blocks, and serve similar functions when designing circuits. We replicate and extend these blocks with our output adapters, adding line-select and PWM output adapters (described later).

Transitioning to the realm of machine learning toolkits, Hartmann et al.'s Exemplar system [13] provides a bridge between sensor-based toolkits and software that recognizes patterns in sensor data. Exemplar offers a standardized input interface for sensors that allows a designer to illustrate desired sensor input, choose the portion of the sensor signal that is of interest, then train to recognize that type of input. Its use of dynamic time warping enables users to provide a single input to train on. Software can then be notified via an event when that sensor input is later seen. Our PICL shares a similar goal of allowing users to demonstrate desired input with as few examples as possible. However, training the PICL is done entirely with the device itself and does not require a standalone graphical user interface. Furthermore, it possesses a number of input inferencing techniques to lessen training time and effort. Another important difference between these systems is that the PICL does not consider a signal over time, while Exemplar does.

Other toolkits have been developed to lower the barrier to utilizing machine learning techniques, especially by those with little experience developing or using machine learning algorithms. Weka [11] is a general-purpose, low-level machine learning toolkit, while Gestalt [22] provides a number of facilities to assist in developing robust classifiers. While these toolkits help bootstrap the creation of systems with a machine learning component, they are aimed primarily at software developers.

Crayons [7] and CueFlik [8] are both examples of interactive machine learning systems, where end-users interactively label data to train a computer vision classifier. MAGIC [2] similarly allows end-users to create a gesture classifier by providing examples. PICL shares a similar vision to these systems by allowing users to explicitly provide example input and the desired output for that input.

Finally, there have been numerous efforts to embed machine learning capabilities directly in chips such as field-programmable gate arrays (FPGAs) (e.g., [27, 12]). We are inspired by these efforts, but are focused on how to make such functionality accessible to the end-user.

DESIGN DESIDERATA

In designing the PICL, our aim was to construct a learning unit that could enable designers and developers to interface arbitrary sensors with simple electronic circuits using a programming-by-demonstration metaphor. Our specific goals are captured in the following design desiderata:

- **A Standalone Unit**: The PICL should be a standalone unit that does not require a computer for training or deployment.

- **Sensor Agnostic**: The system should be sensor agnostic, and users should not need to know exactly how a given sensor responds to its environment. For example, the PICL should not require users to differentiate between positive temperature coefficient (PTC) thermistors, whose resistance increases with temperature, and negative temperature coefficient (NTC) thermistors, whose resistance decreases under the same conditions.

- **Programming-by-Demonstration**: Users should not need to know anything about machine learning, but rather simply feel like they can demonstrate example input and output, correcting the unit as necessary.

- **Simple and Streamlined Training**: The PICL should require as few training examples as possible, preferably a single training example.

- **Minimalist UI**: The PICL should strike a balance between a minimalist design and a rich, verbose interface that enables easy learning and debugging of the system. The interface should also allow for rapid specification of training data. The desire for a minimalist design is motivated on two grounds. First, a minimalist design can help keep costs down with respect to the required hardware. Second, most of the time, the unit will be responding to input, rather than being trained by the user.

- **Minimal Knowledge of Electronics**: Users should be able to operate a PICL and derive value from it without requiring extensive knowledge of electronics.

Collectively, these goals and constraints are intended to explore a specific area of the design space for hardware-based machine learning, namely, low-cost, standalone learning units that are programmed by demonstration alone.

PICL'S USER INTERFACE

The PICL must support two primary tasks: capturing example input and labeling that input. In support of these tasks, its user interface (Figure 1(b)) consists of five momentary buttons, a rotary dial, a mode switch, and LEDs to signal state. The mode switch allows the user to choose the learning mode (i.e., linear classifier mode or linear regression mode). The choice of learning mode also affects which subset of the interface is active, as described below. This design was a result of an iterative design process that explored dozens of interface possibilities. It represents a balance between a sparse interface with overloaded controls, and an interface requiring more expensive components (such as graphical displays).

Linear Classification

When in linear classification mode (referred to as "On/Off Mode" in the PICL's interface), users train the PICL by providing example inputs and labeling those inputs with a desired output of either "ON/5V" or "OFF/0V". The training process proceeds as follows:

1. The user presses the button labeled "Mark Begin/End of Example Input" to begin recording the training example. The LED next to this button begins flashing to indicate the system is monitoring the input.

2. The user demonstrates the example input by performing some action that is captured by the attached sensor. For example, the user might cover a photocell with their hand, or clap near a microphone attached to the PICL.

3. The user terminates capture of the input by pressing the "Mark Begin/End of Example Input" button for a second time. The LEDs near the "ON/5V" and "OFF/0V" buttons begin to flash to signal to the user that they must specify the desired output for the captured input.

4. The user presses either the "ON/5V" or "OFF/0V" button depending on the desired output. After choosing an output, the PICL begins outputting the corresponding electrical voltage. This allows the user to verify that she has chosen the correct output given her needs. Upon pressing a button, the LED near the chosen value remains on, and the LED near the "Save" button begins flashing to signal the next step of the process.

5. Finally, the user presses the "Save" button. The training example is recorded and the PICL's internal models are retrained. The PICL resumes mapping input voltages to output voltages using the newly trained model.

Linear Regression

The PICL's second mode of operation, linear regression (referred to as "Line Fit Mode" in the PICL's interface), maps input to output through a linear model learned from training data. In this mode of operation, the PICL's output is analog, and varies between 0V and 5V. Training in this mode is identical to the previous mode, except that the end-user specifies the desired outputs using the rotary dial.

Cancel, Undo, and Reset

The PICL has the ability to cancel or undo input via the "Cancel/Undo (Hold for Reset)" button. Pressing this button during any stage of the training processes just described (prior to pressing the "Save" button) discards that training example. If the PICL is not currently being trained, pressing this button undoes the prior training example. This allows users to easily recover from erroneous input. We currently support one level of undo, in part because errors should be readily noticed. As the button's name suggests, holding it down resets the device to its initial state.

Interface Shortcuts and One-Touch Training

To expedite training, the PICL provides a number of training shortcuts. For example, after first pressing the "Mark Begin/End of Example Input" button to start capturing a training example, the user can immediately press either the "ON/5V" or "OFF/0V" buttons, or turn the rotary dial (depending on the learning mode), to simultaneously end capture of the input example and to specify the desired output.

The PICL also supports *one-touch training*, which, when coupled with the input inferencing methods described below, greatly simplifies training of the system. To perform one-touch training, the user can simply press the "ON/5V" or "OFF/0V" button to capture the current input and associate it with the chosen output. That is, the user does not need to explicitly press the "Mark Begin/End of Example Input" or "Save" buttons – they can simply press the desired output button if the sensor is currently reading the desired input. In the

linear regression learning mode, users can similarly begin rotating the output knob to set the desired output given the current input. However, the user must explicitly save the training example by pressing the "Save" button, since the PICL cannot infer when the user is done adjusting the output value.

LEARNING MODES AND IMPLEMENTATIONS

We now discuss how the PICL's learning modes are implemented internally. We also describe how the PICL can infer intended training values when capturing input, and how it can learn a linear classifier from a single training example.

Linear Classifier Mode

When operating in linear classifier mode, the PICL examines the input signal voltage and determines if the output voltage should be fully ON (5V) or fully OFF (0V), depending on which side of a learned threshold the input falls. This threshold, V_{THRESH}, is learned from the (x, y) pairs making up the training examples. Here, x can be any input voltage in the range 0-5V, and y is selected to be either 0V or 5V. When there exists a V_{THRESH} that perfectly separates ON examples from OFF examples, the data is said to be linearly separable, and finding a satisfactory V_{THRESH} is trivial. However, in many real-world scenarios, the sensor data is noisy, or otherwise fails to be linearly separable.

In order to better cope with noisy input, and to learn a robust threshold, the PICL employs a one dimensional soft-margin support vector machine (SVM) [6]. While support vector machines are beyond the scope of this paper, it suffices to say that in the one dimensional case, training involves selecting an optimal pair of ON and OFF examples, x_{ON} and x_{OFF}, between which the threshold is placed (i.e., $V_{\text{THRESH}} = (x_{\text{ON}} + x_{\text{OFF}})/2$). These optimal examples are known as the "support", and are chosen so as to maximize their mutual distance from the threshold (i.e., the margins), while minimizing the error contributed by training examples that fall on the wrong side of their respective support. This tradeoff is represented in equations 1 and 2.

$$\text{Cost} = \frac{2}{(x_{\text{ON}} - x_{\text{OFF}})^2} + C \sum_{(x,y)\in\text{Training}} H(x, y) \qquad (1)$$

Where C is a constant which determines how forgiving the support vector machine is of errors, and H is the hinge loss function defined as follows:

$$H(x, y) = \begin{cases} \max(0,\ x - x_{\text{ON}}) & \text{if } x_{\text{ON}} < x_{\text{OFF}} \text{ and } y = 5\text{V} \\ \max(0,\ x_{\text{ON}} - x) & \text{if } x_{\text{ON}} \geq x_{\text{OFF}} \text{ and } y = 5\text{V} \\ \max(0,\ x_{\text{OFF}} - x) & \text{if } x_{\text{ON}} < x_{\text{OFF}} \text{ and } y = 0\text{V} \\ \max(0,\ x - x_{\text{OFF}}) & \text{if } x_{\text{ON}} \geq x_{\text{OFF}} \text{ and } y = 0\text{V} \end{cases} \qquad (2)$$

Linear Regression Mode

When operating in linear regression mode, the PICL employs a simple linear model to map the input signal voltage to an output voltage. Again, the training examples take the form of (x, y) pairs. This time, both the x and the y values can be any voltage in the range 0-5V. From these training examples, standard *simple linear regression* [15] is used to learn a model that maps inputs x to outputs $f(x)$ via the equation $f(x) = \alpha + \beta x$. Here α and β are the usual regression parameters defined as follows:

$$\beta = \ r_{xy}\frac{s_y}{s_x} \qquad\qquad \alpha = \ \bar{y} - \beta\bar{x}$$

Where $\bar{x}$ and $\bar{y}$ are the sample means of the x, and, respectively, y components of the training examples. Likewise, s_x and s_y are the sample standard deviations of the x and y components. Finally, r_{xy} is the corresponding correlation coefficient.

When parameters are chosen as described above, the model is said to minimize the sum of squared errors (SSE):

$$\text{SSE} = \sum_{(x,y)\in\text{Training}} (y - f(x))^2 \qquad (3)$$

Input Inferencing

In both the linear classifier and the linear regression modes, users provide training data to the PICL by pressing a button to start and end a recording session. During the time between button presses, the PICL may sample the input many thousands of times. A decision must therefore be made to determine which sample best represents the user's intended training example.

In many cases, the user can demonstrate the desired input the entire time the PICL is capturing the input example. For instance, if the user wishes to capture a sensor value for a photocell that is in a shadow, it may be relatively easy to sustain the shadow during the time the input is being captured. In such situations, while there will undoubtedly be small variations in the input signal, these variations are minor and will not significantly affect training the model.

In other cases, the desired input is transient and difficult for the user to sustain or isolate when providing example input. For example, the user may wish the PICL to trigger its output when the user makes a loud sound (like hands clapping) or when it detects the flash of a camera. In these situations, the desired input value may be close to the minimum or maximum values observed while training. While we could introduce new capture modes to the PICL to allow the user to specify the type of values of interest (e.g., the minimum or maximum sensed values), doing so requires the user to understand how the sensor varies with different input. It would also introduce additional interface requirements. To avoid these requirements, we developed a means to infer the desired training value.

When training, the PICL employs a strategy we call *input inferencing*. Specifically, for each input example, the PICL learns three training examples. Each example has a distinct x value, but an example set shares a common y value (the desired output specified by the user). These three training examples are as follows:

- $(x_{\text{INITIAL}},\ y)$, where x_{INITIAL} denotes the input at the moment the input capture started

- $(x_{\text{MIN}},\ y)$, where x_{MIN} denotes the *minimum* input value observed during the input capture

- $(x_{\text{MAX}},\ y)$, where x_{MAX} denotes the *maximum* input value observed during the input capture

After each input example, the three training examples are added to three corresponding training sets T_{INITIAL}, T_{MIN}, and T_{MAX}. If the PICL is in linear classifier mode, then the three training sets are used to train three distinct support vector machines: $\text{SVM}_{\text{INITIAL}}$, SVM_{MIN} and SVM_{MAX}. The SVM that minimizes the cost represented by equation 1 is then promoted to the final model, while the remaining two SVMs are discarded. This process is similar to feature selection, and selects the input interpretation that yields an SVM with the best margin width-to-error tradeoff (which best separates the classes in the training data).

If the PICL is in linear regression mode, a similar process is undertaken, but with equation 3 in place of equation 1. Intuitively, this has the effect of selecting the training set exhibiting the strongest linear relationship between observed input and desired output. In the special case of only two training examples, the linear model is fit without residuals, and equation 3 evaluates to zero for all three models. These ties are broken by selecting the training set that covers the largest range of input values, relying on the assumption that the user is likely to demonstrate extremes when building regression models.

Baseline input and training with a single example

As a final convenience to users, the PICL records a *baseline* sensor measurement when the device is first powered on, or when it is reset by the user. This baseline measurement is used as a "stand in" for missing training examples when in linear classifier mode. For example, suppose the user resets the device, then provides a training example that is labeled as ON. After receiving this single training example, the PICL will assume the baseline measurement represents the OFF state, and trains the model accordingly. If the user then provides an OFF example, this inferred baseline example is discarded, and the actual user example is used instead. The CueFlik system [8] employs a similar strategy when the end-user has provided only positive examples to the system. In our experience, this simple heuristic often matches the way we think about training the PICL: we first demonstrate exceptional input, then demonstrate more common input. With this mode of input inferencing, it is often possible to train the PICL to classify input with only a single training example and a single button press.

ELECTRICAL AND HARDWARE INTERFACE

The PICL's electrical interface is inspired by the littleBits electrical interface, which provides power rails and an input signal that is transformed to an output signal.

Input electrical interface

At the most basic level, the PICL requires a single input, V_{SIGNAL}. V_{SIGNAL} takes an input voltage in the range 0-5V, which is output by an active sensor (e.g., an ultrasonic rangefinder or amplified microphone). Internally, V_{SIGNAL} is measured by a 10-bit analog to digital converter.

The PICL also includes a second input: *Divider*. When *Divider* is unconnected (floating), V_{SIGNAL} is treated as described above. However, if *Divider* is connected to V_{SIGNAL} through a resistive sensor (e.g., CdS photocell, thermistor, etc.), the PICL employs an internal voltage divider. The voltage divider enables end-users to directly attach resistive sensors without needing to construct their own voltage divider. While a voltage divider is a relatively straightforward circuit, resistive sensors are so common that we felt it worthwhile to include this circuitry in the PICL itself. This built-in divider also lowers the barrier for end-users new to electrical circuits.

Output electrical interface

The PICL provides a single output, V_{OUT}. V_{OUT} provides an output voltage in the range 0-5V, which corresponds to the evaluation performed on V_{SIGNAL} by the internal logic. The V_{OUT} signal is analog, and is generated by an 8-bit digital to analog converter.

Peripheral Compatibility

The PICL's inputs and outputs range from 0V to 5V, making it compatible with a wide range of logic families and microcontrollers (e.g., Arduino [1], PICAXE [23], etc.). Furthermore, with the addition of littleBits' physical magnetic connectors, the PICL can derive power from a littleBits chain, utilize the littleBits sensors as input, and interface with littleBits output modules. Thus, with no other changes beyond the addition of the littleBits physical connectors, the PICL can add basic computational capabilities to that system.

OUTPUT ADAPTERS

In its standard configuration, the V_{OUT} signal generated by the PICL is useful for many applications, such as triggering an external camera flash when detecting a loud noise. Occasionally, more complex output behaviours are desired. To facilitate the construction of more complex responses to input, we designed a set of *output adapters*. As mentioned in Related Work, these output adapters build on and extend the state blocks found in the eBlocks system [19]. They are intended to attach directly to the output line of the PICL.

Adapters for Binary Output

When the PICL is operating in linear classification mode, its output is either 0V or 5V. In many cases, it is useful to condition this output in some fashion. For example, the input may be extremely transient and rapidly vary (e.g., when hands clap), causing the output to change rapidly. To address these, and similar circumstances, we designed the following binary output adapters:

- **Single Pulse**: In its baseline state, the single pulse output adapter emits a steady 0V. This output adapter monitors the PICL's V_{OUT} for instances where the signal transitions from OFF (0V) to ON (5V). Upon observing this rising edge, the adapter raises its output voltage to 5V for a user-configurable period of time (specified by adjusting a

dial on the face of the output adapter). This output adapter is useful for extending what would otherwise be very brief high-output signals, and makes interfacing with existing hardware (such as remote controls) more reliable.

- **Toggle**: The toggle output adapter is a basic flip-flop: a rising edge in the PICL's V_{OUT} signal causes the output adapter to flip states (if the adapter is outputting 0V, a rising edge on its input line causes it to output 5V, and vice versa).

- **Latch**: The latch output adapter changes its output state from 0V to 5V upon observing a rising edge in the PICL's V_{OUT} signal. Unlike the toggle and pulse adapters, the latch adapter remains in this high state until manually reset by the user. This reset is achieved by pressing a small button on the face of the adapter.

- **Line Select**: Unlike the aforementioned adapters, the line select adapter has two output lines. The first line simply echoes the PICL's V_{OUT} signal. The second line, $\overline{V_{\text{OUT}}}$, transmits the negated signal. This output adapter allows the user to create a system that selects between two circuits.

Adapters for Analog Output

In addition to the binary output adapters, we have developed two output adapters that work with the PICL's linear regression mode.

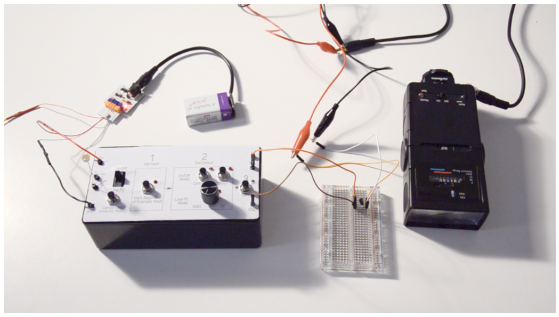
- **Oscillator**: As the name suggests, the oscillator output adapter outputs a clock signal, whose frequency varies as a function of the PICL's V_{OUT} signal. Specifically, the clock rate is mapped to voltage, with higher voltages yielding higher frequencies. When connected to a speaker, the oscillations are audible, and the result is a tone generator with output tones controlled by the PICL.

- **Pulse-width modulation (PWM)**: The PWM output adapter converts the PICL's V_{OUT} signal to a pulse-width modulated signal. This signal quickly switches on and off, but the percentage of time spent in each state (duty cycle) varies with V_{OUT}. PWM control signals are useful in that they can be used to efficiently control high-current devices. For example, this output adapter allows the PICL to control the speed of a motor, or the brightness of high-power lighting.

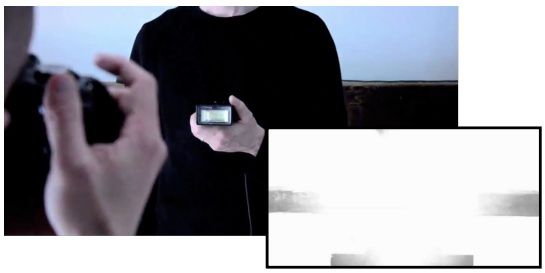
Benefits of Output Adapters

In designing the PICL, one of the questions we faced was whether we should include these types of output conditioning on the PICL itself. In the end, we decided to separate this functionality out of the PICL for a number reasons.

First, doing so maintains a minimalist design of the PICL, whose default input and output electrical interfaces are useful in many situations. Second, some forms of output conditioning require setting additional parameters. By offloading output conditioning to separate modules, output adapters can provide interfaces that are specifically designed for manipulating the relevant parameters. This enables very targeted interfaces, while avoiding a cluttered, overloaded interface on the PICL itself. Finally, externalizing the output adapters enables adapters to be chained together to compose more complex behaviours. For example, a Toggle output adapter and Line Select output adapter could be chained together to create a system that switches between two circuits every time the user claps their hands.



(a) The PICL attached to an external camera flash via a transistor. A photocell also connected to the PICL triggers the flash in response to other bright lights.



(b) The external flash is triggered by the prior flash of a photographer's camera. Here the external flash is used to defensively overexpose photographs (inset).

Figure 2. Implementation of capture-resistant environments.

SAMPLE APPLICATIONS

Having described the PICL and its output adapters, we now present various sample applications that can be easily developed with our system. All of these examples can be viewed in the accompanying video figure.

Capture-Resistant Environments

Truong et al. [25] describe how spaces can be augmented with a system of directed, pulsing lights to prevent the recording of images by unauthorized camera equipment. With the PICL, we can recreate this effect by detecting camera flashes, and quickly responding by activating a defensive flash to overexpose the recorded image (Figure 2(b)). To construct this system, we attach a passive CdS photocell to the PICL and record instances of camera flashes from various locations in the room. We also record several examples of ambient lighting conditions. The PICL's fast sampling rate and input inferencing features enable the PICL to develop an SVM model that accurately detects the camera flashes. The PICL's output

can then be used to interface with a simple switching transistor, which can be used to trigger external camera flashes (Figure 2(a)).

Laser Tripwire Alarm

The PICL can also be configured to create a simple laser tripwire alarm. Using the same photocell as in the previous example, the PICL can be trained to output a low value when the cell is illuminated by a laser beam, and to output a high value when the beam is broken. By attaching a piezoelectric buzzer, the system can be made to output a loud beeping alert whenever the alarm is tripped. If the buzzer is connected through the Latch output adapter, the buzzer will continue to sound even after the laser illumination has been reestablished. To stop the alarm, the Latch adapter needs to be manually reset by pressing a reset button.

Clap Activated Outlet

As a final example, the PICL paired with a remote controlled wall outlet can be used to construct a sound-activated appliance. We accomplish this by training the PICL to output a high signal whenever a clap is detected by a microphone. The PICL's output can then be used to activate a mechanical relay, which closes a connection across the remote's "On" button. In this example, the Single Pulse adapter must be used to prolong the short-lived output signal so that the relay has time to close, and so that the remote detects the input as a proper button press (after debouncing). Now, when the microphone detects a clap, the remote outlet is activated.

In all of these examples, the user can construct a useful circuit with no programming required, and minimal interfacing hardware. For example, a transistor or relay is all that is typically needed to turn an external circuit on or off. In comparison, constructing these applications using a platform like the Arduino would require the user to first take measurements, then write code to trigger an output line when the desired event is observed by the microcontroller.

DISCUSSION

As discussed above, the goal of the PICL is to explore the notion of a low-cost, minimalist, standalone, programming-by-demonstration hardware learning unit. As can be expected, there are limitations associated with the imposition of such severe design constraints. In this section, we discuss the consequences of these design constraints and also outline possibilities for future work.

Limited Interaction

Without an interactive graphical user interface, it potentially can be challenging for the end-user to specify the particular sensor values they wish to train on. For example, they may wish to train on the maximum sensed value, rather than the minimum. In our own uses of the PICL, we have found the input inferencing to work well when inferring the intended training values. However, there may be cases where particularly noisy or widely varying input may lead to the wrong inferences being made. In these cases, the ability to directly select values of interest could be useful (as in the Exemplar system [13]).

Limited Feedback

The lack of a GUI can also make it more difficult to debug the system as a whole. For example, if the system does not seem to be training as expected, the end-user may wish to monitor the current sensor values, the current output values, and the current learned (internal) model. While both the input and output values can be directly measured using tools such as multimeters and oscilloscopes, there is currently no way for the end-user to view the internal model of the PICL. In our own experiences, we have not encountered situations where we were unable to train the PICL, but end-users who are less familiar with sensors and machine learning may run into situations in which they would benefit from a visualization of these values.

Changes in Environmental Context

One particular way in which the PICL may work in unexpected ways for those new to machine learning is when its surrounding context changes. For example, if the PICL is connected to a light sensor and trained indoors, the resulting model will likely fail when taken outdoors. Again, richer feedback could help end-users debug such scenarios and understand why the system does not work as expected.

For this particular problem, we also developed and tested the possibility of allowing the user to shift the learned model. However, we found the resultant interface too clunky. Instead, we opted for a more basic conceptual model of simply retraining the PICL.

Learning temporal patterns

At present, the PICL samples input at a rate of approximately 1200 Hz, and evaluates each sample independently. For example, it trains on inputs such as camera flashes by sensing bright lights, and hand claps by sensing loud noises. While the amplitude of these signals is a strong indication of each event, other key features of these signals are overlooked: their brevity, spectral components, etc. In the future, we look forward to incorporating more temporal features and learning modes into the PICL. Specifically, we are interested in being able to train on features in the frequency domain, as well as the possibility of using dynamic time warping to train on specific patterns. Dynamic time warping is advantageous because it can function with a single training example (e.g., [13]), which is very much in the spirit of the simple training desiderata expressed in the PICL's design.

Input Adapters

As an alternative to enabling temporal models of input, it may be possible to achieve a subset of this functionality through pre-processing of the signal. In this vein, we envision constructing a set of input adapters that are similar in spirit to output adapters, but which filter or transform sensor input. As a simple example, one input adapter may implement a bandpass filter to allow the PICL to detect sounds only within a given frequency range. Similarly, a beat-counter input adapter may allow the PICL to measure tones, or to handle sensors that communicate readings by varying the frequency of an output signal (e.g., some humidity sensors).

Alternatively, it would be useful to develop input adapters that communicate with sensors using common serial interfaces, such as the I^2C serial bus. Such an input adapter would greatly expand interoperability with complex sensors (e.g., accelerometers), but would likely require the input adapters to be preloaded with the communication protocols employed by common I^2C sensors.

Learning with parallel inputs

The PICL's hardware interface accepts a single input signal, V_{SIGNAL}, from which a single feature (discretized voltage) is used to build the PICL's internal models. In the future, we would like to explore the possibility of allowing multiple input signals to train on. For example, one sensor may measure barometric pressures, and another humidity, in order to predict the likelihood of rain. By expanding the feature space, we expand the complexity of the PICL's internal models, and thus increase the need for training data. As such, there are challenges in implementing such models in inexpensive microcontroller hardware. There is also a challenge in maintaining a simplified, low-cost interface to the system.

Cooperating PICLs

As an alternative to accepting multiple parallel inputs, it may be desirable to configure many individual PICLs to work together. In such a scenario, each PICL would monitor a single input or feature. The devices would then communicate with one another to map the ensemble of inputs into a single output signal. Such ensembles are trivial to create by interconnecting current PICLs with standard logic gates or voltage comparators, which is essentially the strategy employed by Exemplar [13]. However, we envision that cooperation could occur earlier in the evaluation process to better model the full input space. For example, one could assemble a group of PICLs to implement a single naive Bayes classifier. In this scenario, each PICL would be independently trained, and would monitor a single feature. The ensemble of PICLs would then communicate with one another to jointly infer the class of an unknown set of sensor readings. Once trained, the individual PICLs could then be swapped in and out, allowing a tangible means of selecting and composing features.

Miniaturization

The current PICL prototype, measuring $16 \times 8 \times 5$ cm, is much larger than it needs to be. This size is mostly a result of our use of an Arduino Mega 2560 microcontroller for prototyping, which itself measures $10 \times 5 \times 1.5$ cm. The enclosure also contains a prototyping breadboard and digital-to-analog converter chip, but much of the PICL's interior is empty space. We have begun the process of miniaturization, and have used surface mount versions of the various electronics to reduce the size of the core assembly (excluding controls) to approximately $7 \times 2 \times 2.5$ cm, for a PICL with only a linear classifier. Miniaturizing the interface will require some simplification of the controls, and remains future work.

CONCLUSION

In this paper, we have presented the PICL, an electronic component that supports basic, in-circuit machine learning. Users wire it into a circuit, then train it by demonstrating example input and desired output. Importantly, the PICL has a minimalist interface, can perform well with minimal training (in many cases, with a single training example triggered with a single button press), is able to infer desired input when recording transient signals, and requires very little knowledge of electronics or machine learning. With the PICL, users can quickly develop interesting applications by simply demonstrating input and specifying the desired output.

This paper also outlined how various simple output adapters can be utilized to expand the behaviour repertoire of the PICL. For example, the Latch or Toggle adapter enables users to develop circuits that maintain state after receiving input, while the Single Pulse output adapter allows users to interface with devices sporting mechanical momentary buttons. Finally, the PICL's electrical interface is compatible with most 5V electronics, including the very popular Arduino microcontroller and the littleBits electronic construction kit. When integrated with such systems, users can quickly interface sensors and transform their input to useful output values, with no programming required.

ACKNOWLEDGEMENTS

We would like to thank Krzysztof Pietroszek for his help in preparing the video figure accompanying this paper. This work was supported in part by an NSERC Discovery Grant and an NSERC Alexander Graham Bell Canada Graduate Scholarship.

REFERENCES

1. Arduino Team. Arduino - homepage. `http://www.arduino.cc/`, Retrieved April 2012.
2. Ashbrook, D., and Starner, T. Magic: a motion gesture design tool. In *Proc CHI '10*, ACM (New York, NY, USA, 2010), 2159–2168.
3. Ballagas, R., Ringel, M., Stone, M., and Borchers, J. istuff: a physical user interface toolkit for ubiquitous computing environments. In *Proc CHI '03*, ACM (New York, NY, USA, 2003), 537–544.
4. Bdeir, A. Electronics as material: littlebits. In *Proc TEI '09*, ACM (New York, NY, USA, 2009), 397–400.
5. Bdeir, A., and Rothman, P. Electronics as material: littlebits. In *Proc TEI '12*, ACM (New York, NY, USA, 2012), 371–374.
6. Cortes, C., and Vapnik, V. Support-vector networks. *Machine Learning 20* (1995), 273–297. 10.1007/BF00994018.
7. Fails, J., and Olsen, D. A design tool for camera-based interaction. In *Proc CHI '03*, ACM (New York, NY, USA, 2003), 449–456.
8. Fogarty, J., Tan, D., Kapoor, A., and Winder, S. Cueflik: interactive concept learning in image search. In *Proc CHI '08*, ACM (New York, NY, USA, 2008), 29–38.

9. Gellersen, H., Kortuem, G., Schmidt, A., and Beigl, M. Physical prototyping with smart-its. *IEEE Pervasive Computing 3*, 3 (July 2004), 74–82.
10. Greenberg, S., and Fitchett, C. Phidgets: easy development of physical interfaces through physical widgets. In *Proc UIST '01*, ACM (New York, NY, USA, 2001), 209–218.
11. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. The weka data mining software: an update. *SIGKDD Explor. Newsl. 11*, 1 (Nov. 2009), 10–18.
12. Hammerstrom, D. A VLSI architecture for high-performance, low-cost, on-chip learning. *International Joint Conference on Neural Networks 2* (1990), 537–544.
13. Hartmann, B., Abdulla, L., Mittal, M., and Klemmer, S. R. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. In *Proc CHI '07*, ACM (New York, NY, USA, 2007), 145–154.
14. Hartmann, B., Klemmer, S. R., Bernstein, M., Abdulla, L., Burr, B., Robinson-Mosher, A., and Gee, J. Reflective physical prototyping through integrated design, test, and analysis. In *Proc UIST '06*, ACM (New York, NY, USA, 2006), 299–308.
15. Hogg, R., and Tanis, E. *Probability and Statistical Inference*. Prentice Hall, 2001, ch. 7, 402–416.
16. Ishii, H., and Ullmer, B. Tangible bits: towards seamless interfaces between people, bits and atoms. In *Proc CHI '97*, ACM (New York, NY, USA, 1997), 234–241.
17. Lee, J. C., Avrahami, D., Hudson, S. E., Forlizzi, J., Dietz, P. H., and Leigh, D. The calder toolkit: wired and wireless components for rapidly prototyping interactive devices. In *Proc DIS '04*, ACM (New York, NY, USA, 2004), 167–175.
18. Lieberman, H. *Your Wish is My Command: Programming By Example*, 1st edition ed. Morgan Kaufmann, 2001.
19. Lysecky, S., and Vahid, F. Enabling nonexpert construction of basic sensor-based systems. *ACM Trans. Comput.-Hum. Interact. 16*, 1 (Apr. 2009), 1:1–1:28.
20. McNerney, T. S. From turtles to tangible programming bricks: explorations in physical language design. *Personal Ubiquitous Comput. 8*, 5 (Sept. 2004), 326–337.
21. Merlo, M., and Bachman, M. A rapid prototyping tool for interactive device development. In *Proc HCII '11*, Springer-Verlag (Berlin, Heidelberg, 2011), 107–113.
22. Patel, K., Bancroft, N., Drucker, S. M., Fogarty, J., Ko, A. J., and Landay, J. Gestalt: integrated support for implementation and analysis in machine learning. In *Proc UIST '10*, ACM (New York, NY, USA, 2010), 37–46.
23. Revolution Education Ltd. PICAXE. `http://www.picaxe.com/`, Retrieved April 2012.
24. Suzuki, H., and Kato, H. Interaction-level support for collaborative learning: Algoblock – an open programming language. In *Proc CSCL '95*, L. Erlbaum Associates Inc. (Hillsdale, NJ, USA, 1995), 349–355.
25. Truong, K., Patel, S., Summet, J., and Abowd, G. *Preventing Camera Recording by Designing a Capture-Resistant Environment*, vol. 3660 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2005, 903–903.
26. Villar, N., Scott, J., Hodges, S., Hammil, K., and Miller, C. .NET gadgeteer: A platform for custom devices. In *Proceedings of Pervasive 2012*, LNCS, Springer-Verlag (2012), 216–233.
27. Zhu, J., and Sutton, P. FPGA implementations of neural networks - a survey of a decade of progress. In *Proc FPL '03*, Springer-Verlag (2003), 1062–1066.