Punch-O-Matic

$1,000,000,000

Patrick Dillon (pdillon), Blade Olson (bladeols)

Abstract

The Punch-O-Matic boxing sensor is a wearable data capture and analysis system for punches, jabs, uppercuts, hooks, and any kind of gesture that ends in an impact on the knuckles of the Punch-O-Matic glove. The Punch-O-Matic sensor incorporates a 9-DoF inertial measurement unit (IMU), a pressure sensor that is connected via a 12-bit analog-to-digital converter (ADC), and light-emitting diodes (LEDs) for user feedback. The ADC and IMU both communicate over I^2C. The threshold for the pressure sensor is set such that an impact is only triggered during a user-intended punch. A circular buffer FIFO queue continuously stores incoming IMU data at a rate of about 100Hz. When an impact is detected, the last 50 frames stored in the FIFO queue are compared against a set of recorded punches to classify the most recent punch gesture; the Punch-O-Matic's best guess is presented to the user via LEDs for feedback.

1. Introduction

Diagnostic and online monitoring of physical activity is a growing demand for physical therapists, entertainment technology, sports trainers, and postoperative monitoring for surgeons. In response to the need for a low-cost, low-profile, versatile, extensible, wearable activity sensor, the Punch-O-Matic boxing sensor is a proof-of-concept device that demonstrates that on-board gesture recognition and high-throughput data monitoring are possible on a wearable sensor that can withstand violent impacts. In fact, when demoing the device, the Punch-O-Matic glove stitching broke before the sensor or wiring failed. Because of the Punch-O-Matic's ability to accurately classify different punches during a single training session, the Punch-O-Matic would make for an ideal telemetry gathering device for fitness enthusiasts or athletic trainers. Additionally, its ability to gather raw sensor values and run calculations at a high frame rate make the Punch-O-Matic an ideal diagnostic device for physical therapists who want to see slight perturbations across a user's gestures in a single recording session or look at discrepancies between ideal motions/gestures from a healthy individual and the user's current motions/gestures.

The Punch-O-Matic sensor incorporates a Gumstix Verdex Pro running Linux, a 9-DoF inertial measurement unit (IMU), a pressure sensor that is connected to the Gumstix via a 12-bit analog-to-digital converter (ADC), and LEDs for user feedback. The ADC and IMU both communicate over I^2C. The LEDs communicate to the Gumstix through general purpose input/output (GPIO). All software was written in C and runs exclusively on the Gumstix Verdex Pro. A linux kernel module was written to interact with the LEDs from the user-space program that performs data capture and analysis. IMU data was smoothed and corrected online ('online' meaning actively analyzed as the data is being recorded versus 'offline' where the data is analyzed long after recording) with an open source attitude and heading reference system (AHRS) provided by Mahony[1][2]. A circular buffer queue was used to store and retrieve sensor data for recording and analysis. Punch classification compares accelerometer values at each data point and chooses the gesture with smallest discrepancy.

Each of three LEDs on the Punch-O-Matic glove represents a different gesture type. After the punchomatic program is started, the user is prompted to record three gestures by way of three flashing LEDs. In the background, IMU data is continuously being recorded. The first, yellow LED flashes until an impact is registered, at which point the last 50 frames of IMU data are used as the 'fingerprint' for the gesture. This gesture 'fingerprint' is stored for the rest of the session. Two additional gestures are recorded in an identical manner using the red and blue LEDs for the subsequent punches. After three gestures have been recorded, the user can punch in any form and the Punch-O-Matic will classify the new punch according to the three recently recorded punch gestures. Feedback on the most closely related punch is presented by lighting up the corresponding LED of the originally recorded gesture when a new punch occurs.

The Mahony AHRS smoothing algorithm provides euler orientation data from the raw 9-DoF IMU data with a Kalman-like filter that can be used for further off-board analysis of the punch. Accelerometer, gyrometer, and magnetometer data can be used for analysis as desired or the euler data can be used to orient the device in 3D space. If the base sensor of the Punch-O-Matic was replicated across the user's different joints, such as the elbow, shoulder, and waist, a full skeletal capture of the user's movement could be replicated in 3D with visualization programs such as Unity3D or Maya. Similar motion-tracking devices on the market, such as the Rokoko suit, are much more expensive while providing the same functionality. As explained in the design flow below, the final version of the Punchomatic is able to correctly identify user punches with high accuracy by combining the pressure sensor as a 'trigger' event and the IMU data as the event 'fingerprint'. However, despite having euler angles, accelerometer data, gyrometer data, and magnetometer data from which to generate and check an event 'fingerprint', only the accelerometer data was needed to develop a sufficient classification accuracy during testing. Though untested, adding gyrometer data to the classification algorithm should improve the classification accuracy even more without forcing a noticeable slowdown in computational time.

2. Design Flow

The Punch-O-Matic consists of a user-level program which reads input from the IMU and pressure sensor and communicates user feedback through LEDs. The user-level program reads from the IMU directly over I^2C while the pressure sensor is connected to an ADC, which is then read through I^2C (Figure 1). A kernel module was written to control the LEDs over GPIO and the user program communicates with the kernel module through /dev/led.

- Blade wired the Gumstix to all peripheral hardware.
- Patrick and Blade worked together to position the pressure sensor and breadboard on the glove and tape the loose wires to the glove.
- Blade retrofitted a raspberry pi case for the Gumstix and secured the case to the glove.
- Blade debugged and setup I^2C communication with the IMU and ADC used for the pressure sensor.
- Patrick organized Blade's IMU and ADC code file into structured files to adopt an appropriate long-term development architecture.
- Patrick wrote the initial user-level software/algorithm to collect data/detect punches

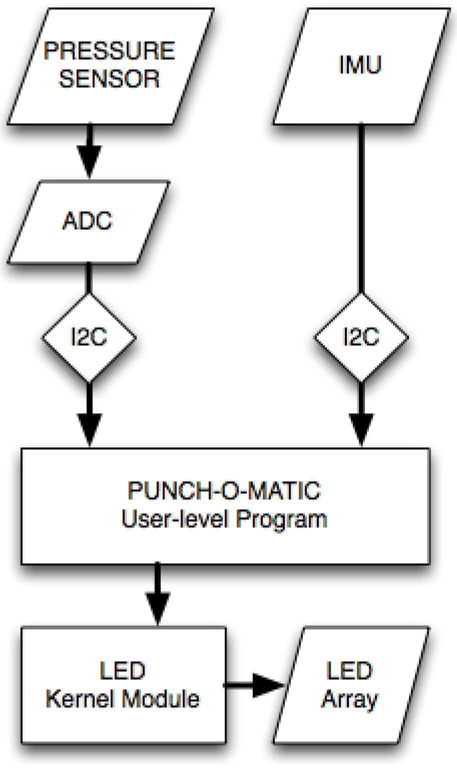


Figure 1: System architecture for hardware and kernel modules

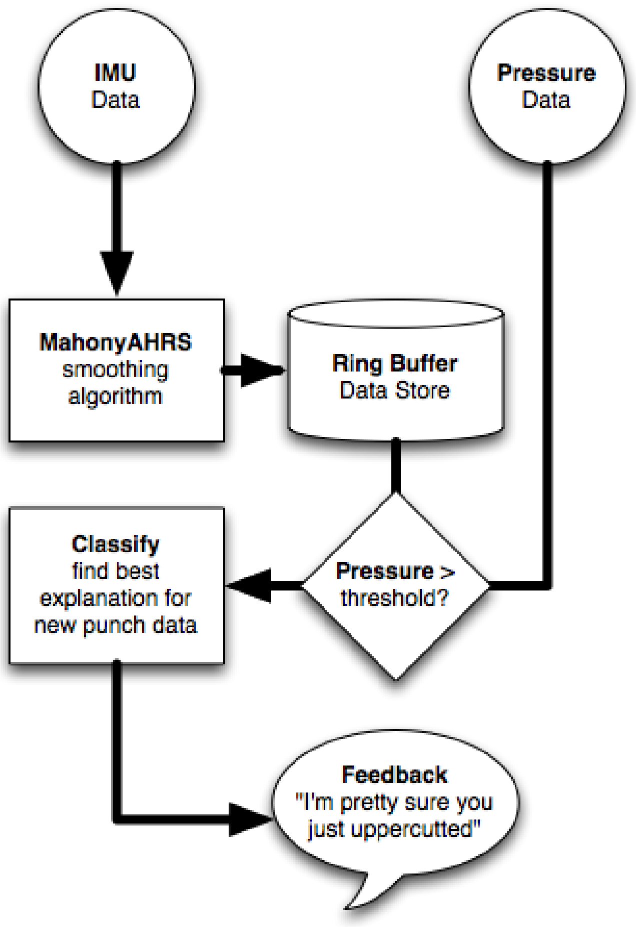
using the data coming from the IMU and pressure sensor.
- Patrick wrote the bare-bones LED kernel module.
- Patrick wrote the ring buffer data store that housed and returned the ongoing stream of IMU data.
- Blade extended the LED kernel module to allow for delayed start timers and 'flash' timers.
- Blade wrote the user feedback LED code within Patrick's main punch-check algorithm.
- Blade converted the arduino MahonyAHRS algorithm to work on the Gumstix with our IMU.
- Blade wrote the motion-analysis/punch-matching algorithm.

~40% of work was Patrick, ~60% of work was Blade

3. Project Details

a. **User-Level Program**

The user-level program initializes I^{2}C communication through our I^{2}C library and begins continuously putting sensor data into ring buffers for each type of sensor data (see below for descriptions of each of these libraries). The main algorithm loop is largely the same with a slight variation between recording initial punches and identifying punches. Once the pressure sensor value passes a certain threshold, a copy of the ring buffers containing spatial coordinates is either stored for reference or, when analyzed, evaluated against the stored copies to determine the best match. In order to identify the correct punch, we compare the new punch with three saved punches and select the one with least error. Feedback to the user is presented through the LED module, which the user-level program communicates to by writing to the /dev/led file. See Figure 2 for a diagram that helps explain the process described here. We ran into a problem communicating with the kernel module, as this needs to be done very carefully when requests are being made at very fast speed, and constantly. Ultimately, our results were successful. The Punch-O-Matic user level program was able to properly identify punches with an even simpler evaluation method than we originally anticipated.


IMU Data
Pressure Data
MahonyAHRS smoothing algorithm
Ring Buffer Data Store
Classify find best explanation for new punch data
Pressure > threshold?
Feedback "I'm pretty sure you just uppercutted"
Figure 2: Overview of the software architecture for translating IMU and Pressure data to user feedback

b. **I^{2}C Library** (see i2c_punchomatic.c, i2c_punchomatic.h, & i2c_test.c)

We created an I^{2}C library to easily allow initialization of I^{2}C communication and reading from the I^{2}C line. Our sensor library utilizes this library for the lower-level I^{2}C communication. Communication over I^{2}C was a major difficulty that we overcame during this project. Initial communication with the ADC was problematic. The address specified in the datasheet [3] did not work as expected. Despite not having one of the 'slave address' ID pins soldered, the IMU slave address responded as if that pin was high. We ultimately resolved the issue by scanning all potential slave addresses with a bash script and noticing that only one of the addresses responded appropriately. In retrospect, checking the general call address would have alleviated many of the initial debugging directions that were explored as we were open to either the software or hardware being the source of the problem. When scanning for slave addresses, both the general call address and the unexpected, unique slave address for the IMU worked. In the future, it would be nice to have known, working I^2C code or known, working I^2C hardware before trying to debug both directions at the same time. A USB logic analyzer would have proven very useful here[4]. While less strenuous of a debugging process, there was also difficulty trying to communicate with the 12-bit ADC. With the original IMU I^2C code that was written, registers were read 1 byte at a time (after sending the slave address and target register address); however, every register on the ADC required 2 bytes to be read for a full data transfer (after sending the slave and register addresses). Again, information on the ADC datasheet did not match the data retrieved from I^2C register reads, which again made us question whether our multiple-byte I^2C read code was running correctly. As one example, the default startup settings on the ADC that we read off the config register were different than the settings listed on the datasheet. By writing and reading to a 2-byte threshold register, we were able to prove that we were in fact writing and reading the registers on the ADC correctly. To further complicate our development, we wanted to use a built-in, writable threshold value on the ADC, along with a latching ALERT/RDY pin, to know when to read values off the ADC as the ADC has a ~1ms delay for converting analog values to digital values. After a series of confusing bugs involving the config register, we found that we could set the ADC to 'continuously' convert incoming analog signals; in our final code we continuously read from the ADC's conversion value register to get the pressure sensor data. After these issues were resolved, the sensor communication worked very well, providing reliable, high-speed data transfer at about 100 Hz.

c. **Sensor Library (**see sensor.c & sensor.h)
We wrote a sensor library which handles retrieval of data from both the IMU and the pressure sensor. At this level, the pressure sensor is straight-forward: we simply read a short from the address of the pressure sensor register. For spatial coordinates, a coords struct holds x, y, z as float values. This struct can be used by get functions to return mag, accel, or gyro. To get roll, pitch, & yaw we converted outside code, MahonyAHRS[1], to work with our sensors library.

d. **Ring Buffer Library (**see ring_buffer.h & ring_buffer.c)
The primary data structure for our program is a ring buffer. We wrote this library in order to hold the constant stream of sensor data. The design is relatively simple while efficient enough to handle the high throughout of sensors constantly writing. The ring buffer provides a circular FIFO queue so that once the queue has been filled the oldest data is overwritten by the newest data. A copy of the current state of the buffer can be made by passing a pointer to an array. Copying the ring buffer is linear. The size of the buffer is adjustable via a macro. The buffer holds spatial coordinates and represents a sliding time window. Therefore, changing the size of the ring buffer will represent an increase or decrease in the length of time. The sensors capture data at 100 HZ so a ring buffer size of 50 would represent half a second. Our simple data structure held up very well to the demands of the program. After some early bug fixes, we never experienced any crashes.

e. **LED Module** (see led.c)
The LED module provides output to the user through LEDs connected by GPIO. The LED module reads input from the user-level program, which tells the module to begin certain states. The module triggers LEDs to be either off or on for a certain duration (accomplished through kernel timers), how long to delay the start of the LED turn-on/flash, and whether the pattern should be continuous or only occur once. The user-level program enables these states to provide user feedback, such as a quick flash of all lights to show a punch has been saved, circulating lights to show the Punch-O-Matic is ready for a punch, or a constant light for the identified punch.

f. **Pressure Sensor - ADC Circuitry**
Because the pressure sensor communicates voltage values as a variable resistor and because we were originally looking to build a “best punch” detector, analog values needed to be retrieved. Given that we had already wired the IMU over I^2C, we decided to use an I^2C-capable 12-bit ADC to connect the pressure sensor to the Gumstix[5]. Unfortunately, the pressure sensor saturates very easily, that is: almost all punches max out the sensor at 1024. This was existentially problematic because our initial goal was to train users about how to perform their hardest punch, but we quickly pivoted to simply using the pressure sensor as a trigger for evaluating punches through the IMU data.

g. **IMU Circuitry**
The IMU is wired on its CS, SCL, SDA, GND, and VCC pins. The SCL, SDA, GND, and VCC pins are self-explanatory. The CS pin needs to driven high for I^2C communication to be enabled. An unwired SDO pin, which remains unwired in the final Punch-O-Matic, was the cause for much of the frustration when originally diagnosing problems with I^2C communication. SDO is responsible for the least significant bit of the slave address for the IMU. Driving SDO low or high would theoretically allow two IMUs to be wired to the same I^2C line and be uniquely addressable. Unfortunately, even though this pin was never soldered (in fact, a pin is completely absent on the breakout board), the slave address for our IMU considers this pin to be driven high. We speculate that the breakout board must have a wired connection to SDO from one of the other lines that is not evident by eye as it is unlikely that the datasheet is inaccurate.

h. **Product Design**
The Punch-O-Matic uses a cheap, consumer-grade adult boxing glove from Amazon as its main device body. To protect the Gumstix from thrashing and impact damage, an off-the-shelf raspberry pi case was used to house the development board (Figure 4); duct and electrical tape restrained the raspberry pi case to the bottom wrist strap of the boxing glove and bubble wrap filled the loose space within the raspberry pi case surrounding the Gumstix board. To provide ongoing access during development, the raspberry pi case was cut open at key areas to give cable access to the Gumstix serial and power ports. The breadboard that connected the ADC and IMU to the Gumstix was taped down to the top of the boxing glove wrist strap (Figure 3); stable attachment of the IMU was necessary for reproducible and reliable gesture recognition. Loose wires were taped together and eventually taped down to the glove to secure wires from coming unplugged during punching. The pressure sensor was first embedded in the glove itself, but was later placed on the exterior of the glove and covered in tape. When embedded in the glove, the pressure sensor often traveled to other non-ideal impact sites on the glove. On the glove’s exterior, the pressure sensor performed optimally and served as a guide for where the user should make contact during a punch. Two different pressure sensors were trialed. The larger pressure sensor was abandoned because its

sensor value reached saturation too easily and too often. The final glove uses only the small pressure sensor.

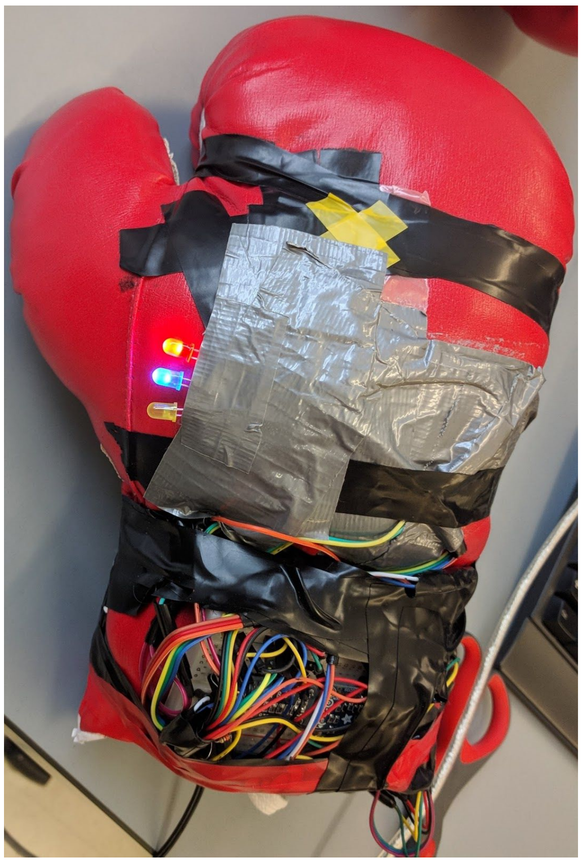


Figure 3: Top view of the Punch-O-Matic. The pressure sensor is located under the yellow tape. The bread board containing the IMU and ADC can be seen on the bottom.

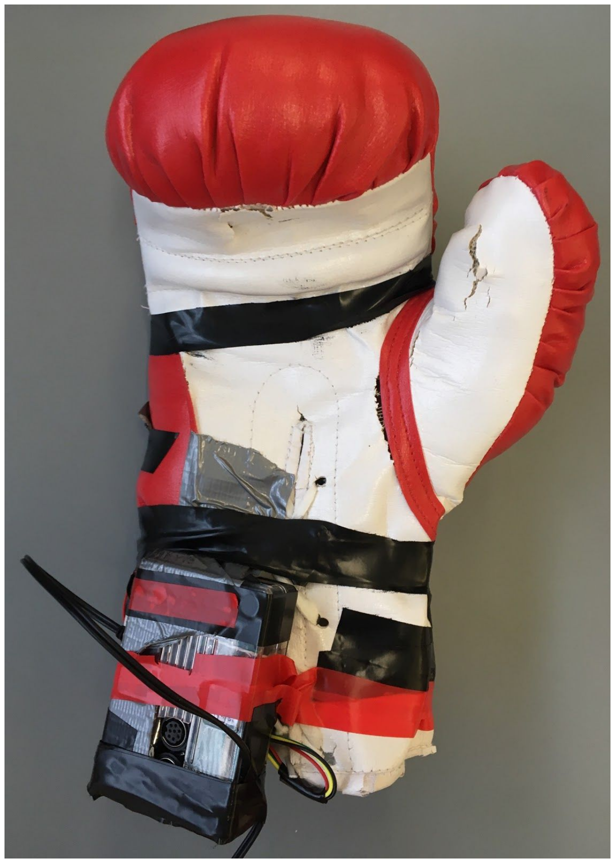


Figure 4: Bottom view of the Punch-O-Matic. The Gumstix is housed in a Raspberry Pi case with makeshift holes cut for the serial, I^{2}C, GPIO, and power cables.

4. Summary

The Punch-O-Matic represents a successful implementation of a wearable, completely self-contained embedded system. We proved the viability of collecting and processing IMU and pressure sensor data with highly responsive feedback despite repeated forceful impacts. The Punch-O-Matic system effectively delegates tasks from user-space to kernel-space. We implemented a successful codesign of hardware and software, where the hardware comprised high-throughput sensors communicating over I^{2}C and the software was developed with these demands in mind.

Some future directions for the project including battery-power for the device, which we were unable to test in the final version. We only tested the device for a few days, but a realistic production model would need to withstand tens of thousands of punches, at least, to be considered a usable product by athletes and fitness trainers. Overall, the final version of the Punch-O-Matic should be able to train a user, without outside human guidance, to punch their best. Audio feedback instead of LED feedback would be ideal so that the user does not need to interrupt their boxing in order to discern the state of the Punch-O-Matic. Ideally, the final glove would feature a pressure sensor that can discern impacts between professional athletes and amatuer trainers. Lastly, the overall hit classification and feedback system would need a large time investment of user experience design and coding so that users would receive the appropriate feedback to intuitively correct their punch form for the better.

References

[1] “Mahony AHRS algorithm”, https://github.com/dccharacter/AHRS/blob/master/MahonyAHRS.c, (retrieved in Dec. 2017)

[2] x-io Technologies Limited, “Open source IMU and AHRS algorithms,” source code, http://x-io.co.uk/open-source-imu-and-ahrs-algorithms/ (retrieved in Dec. 2017)

[3] STMicroelectronics “iNEMO inertial module: 3D accelerometer, 3D gyroscope, 3D magnetometer,” Adafruit LM9DS0 Datasheet, https://cdn-shop.adafruit.com/datasheets/LSM9DS0.pdf, Aug. 2013 (retrieved in Dec. 2017)

[4] Sparkfun, “Logic Pro 8 - USB Logic Analyzer“ https://www.sparkfun.com/products/13196 (retrieved in Dec. 2017)

[5] Texas Instruments, “Ultra-Small, Low-Power, 12-Bit Analog-to-Digital Converter with Internal Reference”, ADS1015 Datasheet, https://cdn-shop.adafruit.com/datasheets/ads1015.pdf, May 2009 [Revised Oct. 2009] (retrieved in Dec. 2017)