Virtual Touch Sensors
Jeremy Rand
Team SNARC (Sooners / Norman Advanced Robotics Coalition)
jeremy.rand@ou.edu

Virtual Touch Sensors

1 Introduction

Touch sensors are a standard element of a Botballer's toolbox. But what if I told you that in many cases, the same functionality provided by touch sensors can instead be provided without any mechanical complexity, using only the same hardware that Botballers typically use for dead-reckoning? It's true, and I've been doing exactly that since 2008. This paper will discuss the technology and history of Virtual Touch Sensors, explain which implementations are superior, and offer tips for practical applications in Botball today.

2 The Theory

Typically, touch sensors are used to detect a collision. This could be a collision between the robot and its environment, or a collision between components of the robot. In both cases, certain built-in sensors may be able to detect this collision, even if no touch sensor is installed.

3 Motor Current Sensing

Current sensing is one of the oldest Virtual Touch Sensors in Botball. When DC motors stall or experience heavy load, they quickly begin consuming a high amount of current (much higher than in light load). The Xport Robot Controller (XRC) and Xport Botball Controller (XBC), which was used in Botball between 2005 and 2008, was capable of reading the current draw of each motor, and could use this to detect the load. XRC/XBC developer Charmed Labs produced an excellent demo video of the XRC wandering around a room, without any touch sensors [1]. When the XRC detected a heavy load on its drive motors, it would conclude that it had hit a wall, play an "Ahh!" or "Ouch!" sound effect, and back away.

The libxrc C++ function for accessing the current draw of a motor is:

```
short CAxesOpen::GetCurrent(unsigned char axis);
```

Users of Interactive C can access this function through the XBC IC Firmware Student Edition:

```
// Returns current draw for motor, unknown units
int get_current(int m)
{
	return(callml(1581, m));
}
```

The XRC/XBC updates this data every 5ms, and current draw doesn't oscillate very much, making this an excellent method of measuring load on a motor.

The "baseline" value for GetCurrent varies based on your robot's construction, but simple testing will usually show a huge difference between normal driving and driving into a wall. It is often possible to also use this to detect unusually high loads which haven't completely stalled a motor, e.g. detecting that the robot is traveling up a hill.

Unfortunately, motor current sensing circuitry was not present in the CBC Botball Controller, and so far hackers have not found any such feature in the KIPR Link. The dated CPU speed and extremely limited availability of the XRC/XBC means that most Botballers will not be able to use this type of Virtual Touch Sensor (and certainly not in Botball tournament settings, where only the Link is legal).

4 PWM Power

The XRC/XBC, CBC, and Link all support PID motor control (e.g. `mav()`). PID control attempts to keep motors running at a constant velocity by varying the PWM duty cycle (power) given to the motor. If a motor encounters heavy load while under PID control, the PWM power will increase as the PID controller tries to maintain the requested velocity. This can be used to detect the increased load.

4.1 XRC/XBC

On the XRC/XBC, libxrc's function for this is:

```
int CAxesOpen::GetPWM(unsigned char axis);
```

XBC IC Firmware Student Edition supports a callml for this function:

```
// Returns PWM output for motor, range = [-255,255]
int getpwm8(int m)
{
	return(callml(1580, m));
}
```

4.2 CBC

On the CBCv2, the function is available as well:

```
int getpwm(int motor);
```

However, the CBC simulator does not support `getpwm`, so the following hack is necessary when compiling with KISS-IDE:

```
// Insert this at the top of your program to make KISS-IDE work with getpwm
#ifndef __arm__
int getpwm(int motor) {return 0;}
#endif
```

4.3 Link

At the moment, `getpwm` is not implemented on the KIPR Link's user libraries. However, I suspect that a reasonably competent hacker could add a working implementation, as there is no hardware limitation involved. Any takers?

4.4 Other Notes

Because PID controllers maintain average velocity over a few seconds but not necessarily instantaneous velocity, the PWM power can vary widely in normal operation, even sometimes hitting the maximum value of 100%. As a result, GetPWM is less robust as a Virtual Touch Sensor than GetCurrent. However, CBC and Link users don't have the luxury of choosing, so GetPWM is a useful tool to master. A particularly useful technique is to average the GetPWM results over time, which improves accuracy (at the expense of introducing a delay).

GetPWM can be helpful as a diagnostic tool as well. Have you ever noticed that if you try to `mav()` at too high a speed, your robot will start drifting away from the straight-line path you intended? This is because `mav()` will max out your motors at 100% if it cannot reach the requested velocity, which eliminates the benefits of PID control. Contrary to popular myth, 1000 ticks/sec is not the "maximum safe speed"; the maximum speed depends on the robot's mechanical design. So how do you figure out the maximum safe speed? You can find it by monitoring the GetPWM output from the drive motors. If both motors are dipping below 100% regularly, your speed is safe and you can try to increase the speed. If either motor is stuck at 100%, your speed is unsafe and you should decrease the speed. You want the highest possible speed which allows both motors to dip below 100%. (Even if the motors sometimes hit 100%, that's fine, as long as they also dip below 100% at times.)

5 Accelerometers

Collisions with walls produce strong accelerations, which can be detected by the accelerometers in recent Botball controllers.

On both the CBC and Link, the functions `accel_x()`, `accel_y()`, and `accel_z()` will spike when a collision occurs in the corresponding axis.

Accelerometers differ from GetPWM and GetCurrent in that accelerometers only can detect the collision event. After the robot has come to a stop, accelerometers will no longer detect anything out of the ordinary. GetPWM and GetCurrent will continue to detect a high motor load after the robot comes to a stop, until power to the motor is switched off.

6 The iRobot Create

The iRobot Create Open Interface allows access to a number of useful variables which can serve as Virtual Touch Sensors. For example, these could be used to simulate a rear bump sensor. Unfortunately, these functions are unimplemented on the Link, although a competent hacker should be able to fix this. For more details, consult the iRobot Create manual.

6.1 Motor Overcurrents

The CBC function `get_create_overcurrents()` returns a bitfield, with each bit indicating whether a motor has hit its current limit. Stalled Create motors typically will show up as a 1. According to the Create manual, the bits are, starting with the least significant bit: LSD1, LSD0, LSD2, Right Wheel, Left Wheel.

6.2 Battery Current Flow

Motor overcurrents are useful when you want to distinguish between different motors, but they are digital rather than analog. When you want an analog reading, the CBC function `get_create_battery_current()` returns the current flowing into or out of the Create's battery. (The value will be negative under normal operation; positive values indicate that the battery is charging.) When a robot is stalled (e.g. due to a collision), a much higher amount of current is pulled from the battery than under standard conditions.

7 Conclusion

Virtual Touch Sensors provide an excellent software alternative to hardware touch sensors. Although the Link's user libraries do not currently support some of these features, this will most likely improve, either due to work by KIPR or by hacking. For teams which prefer software solutions over hardware solutions, or who are just worried about a touch sensor falling off, becoming unplugged, or otherwise failing, Virtual Touch Sensors may be the ideal solution.

I can be reached at the Botball Community [2]; feel free to stop by if you have any questions about this paper. Happy Sensing!

References

[1] Charmed Labs. Wander. http://www.charmedlabs.com/index.php?option=com_docman&task=doc_download&gid=12&Itemid=44 , 2006.
[2] Botball Youth Advisory Council. Botball Community. http://community.botball.org , 2013.