CBC Hacking 2011: Vision Enhancements and Sensor Speedups (Part 2)
Jeremy Rand
Norman Advanced Robotics
jeremy.rand@ou.edu

CBC Hacking 2011: Vision Enhancements and Sensor Speedups (Part 2)

8 Welcome to Part 2!

Welcome back to CBC Hacking 2011. In Part 2, we'll cover additional hacks which can speed up camera vision and sensor access, which didn't fit into Part 1.

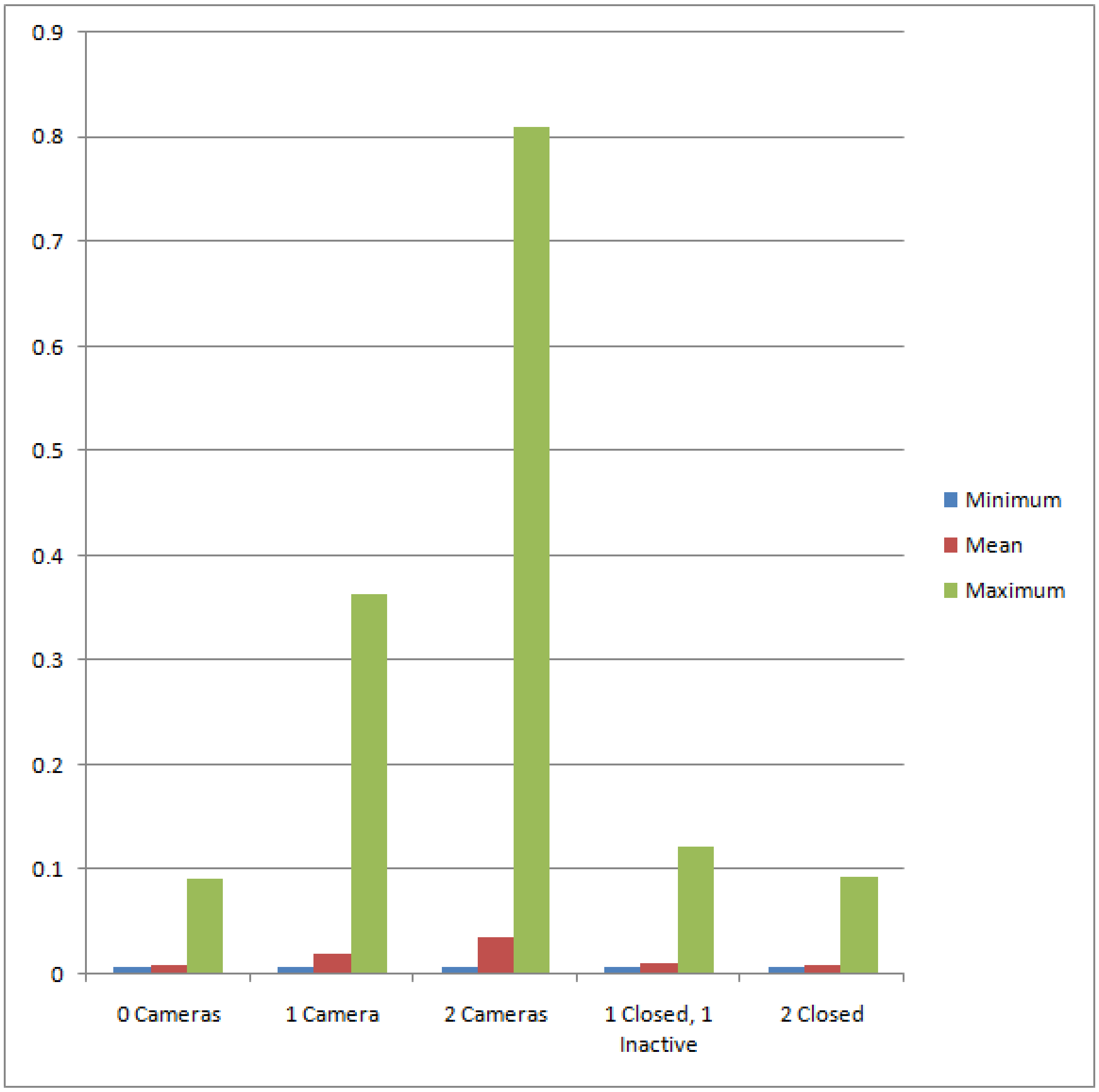
9 Disabling Cameras

Botball teams have frequently noted that their CBC responds more slowly when a camera is plugged in. This is not entirely due to the tracking calculations; the camera driver itself generates a large quantity of interrupts which occasionally lags the Chumby. If you've seen your robot occasionally overshoot a distance or turn, you may have noticed that this behavior decreases or completely disappears when the camera is unplugged. In 2010, Matthew Thompson discovered that closing the camera device eliminates these interrupts, and added a GUI option to the NHS Patchset which completely disables the camera. Matthew used CBCLua [6] to implement his own vision system, allowing him to selectively enable/disable the camera for that custom vision system.

However, for Botballers who wish to use KIPR's vision system without unnecessary lag, I have added a feature to temporarily close a camera device. Just use the Multiple Cameras FIFO, but add 0x80 to the camera number to close that camera device. This works with multiple camera setups as well; an example usage might be:

1. Switch to camera 0x81 (close camera 1).
2. Switch to camera 0x02 (open camera 2).
3. Do stuff with camera 2.
4. Switch to camera 0x82 (close camera 2).
5. Switch to camera 0x01 (open camera 1).
6. You're back at the default camera configuration.

A graph of the lag (in seconds) associated with cameras is below:



Notice that closing cameras is roughly equivalent to unplugging them. Remember that opening a camera takes several seconds before a picture is visible, so only close a camera if you're willing to wait several seconds before using it again.

10 CBOB Batch Access: Speeding Up Sensor/Motor Access

The CBOB firmware itself is very risky to hack, due to the bricking risk. As a result, no such hacking has been attempted. However, its source code is available, and reading it yields some useful knowledge which can unlock undocumented features. Each time the user requests data to be retrieved from or sent to the CBOB, a 6ms SPI transaction occurs. If the user is repeatedly refreshing all the sensor and motor I/O, this can cripple the CBC's CPU usage, since the 6ms transaction busy-waits. (All attempts so far to make the 6ms transaction use less CPU have resulted in randomly crashing the Chumby, so that method appears to be a no-go.)

However, it turns out that the CBOB can support reading or writing multiple data types simultaneously. Specifically, the following useful undocumented packets exist:

- CBOB to Chumby (input)
    - `sensors` packet, which contains:
        - All digitals
        - All analogs
        - Battery voltage
        - All accelerometers
        - Analog pullup states
    - `pid` packet, which contains all 4 motor positions
- Chumby to CBOB (output)
    - `pwm` packet, which assigns PWM speeds for all motors

Each of these packets only takes 6ms to execute, compared to much higher delays if the commands were sent individually. For example, the `sensors` packet would take 150ms if read individually, which could be unusable depending on the application.

The `pwm` packet may look useless since PID control (`mav`/`mrp`) is preferable to PWM control (`motor`). However, it can actually be very beneficial. Ever notice that when the two drive motors are turned on, one occasionally starts before the other? This causes significant drift which interferes with odometry. However, if you know roughly what PWM values your drive motors need, you can start both motors at that PWM value using the `pwm` packet, which is guaranteed to start both motors at exactly the same time, and then run the `mav` commands as usual. Since the motors don't accelerate much due to the `mav` (they're already going close to that speed), the drift will be much less.

The following code will allow you to access these undocumented CBOB commands:

```
#include <fcntl.h>

int cbob_batch_inited = 0;

int g_sensorsBatch;
int g_pwmBatch;
int g_pidBatch;

void init_cbob_batch()
{
	if(! cbob_batch_inited)
	{
		g_sensorsBatch = open("/dev/cbc/sensors", O_RDONLY);
	g_pwmBatch = open("/dev/cbc/pwm", O_RDWR);
	g_pidBatch = open("/dev/cbc/pid", O_RDONLY);
```

```
      cbob_batch_inited = 1;
      }
}

void exit_cbob_batch()
{
      if(cbob_batch_inited)
      {
      close(g_sensorsBatch);
      close(g_pwmBatch);
      close(g_pidBatch);

      cbob_batch_inited = 0;
      }
}

void sensors_batch(short *digitals, short *analog0, short *analog1, short
*analog2, short *analog3, short *analog4, short *analog5, short *analog6,
short *analog7, short *battery_voltage, short *x, short *y, short *z, short
*analog_pullups)
{
      short sensorsData[14];

      read(g_sensorsBatch, sensorsData, 28);

      *digitals = sensorsData[0];

      *analog0 = sensorsData[1];
      *analog1 = sensorsData[2];
      *analog2 = sensorsData[3];
      *analog3 = sensorsData[4];
      *analog4 = sensorsData[5];
      *analog5 = sensorsData[6];
      *analog6 = sensorsData[7];
      *analog7 = sensorsData[8];

      *battery_voltage = sensorsData[9];

      *x = sensorsData[10];
      *y = sensorsData[11];
      *z = sensorsData[12];

      *analog_pullups = (~sensorsData[13])&0xFF;
}

void write_pwm_batch(signed char pwm0, signed char pwm1, signed char pwm2,
signed char pwm3)
{
      signed char pwmData[4];
```

```
	pwmData[0] = pwm0;
	pwmData[1] = pwm1;
	pwmData[2] = pwm2;
	pwmData[3] = pwm3;

	write(g_pwmBatch, pwmData, 4);
}

void read_motor_positions_batch(long *pos0, long *pos1, long *pos2, long 
*pos3)
{
	long positionsData[4];

	read(g_pidBatch, positionsData, 16);

	*pos0 = positionsData[0];
	*pos1 = positionsData[1];
	*pos2 = positionsData[2];
	*pos3 = positionsData[3];
}
```

11 Conclusion

CBC hacking may have slowed down since 2009, but is by no means dead. If you have a cool idea for a CBC hack, let me know! I can be found on the Botball Community [4]. Happy Hacking!

References

[1] J. Rand, M. Thompson, B. McDorman. Hacking the CBC Botball Controller: Because It Wouldn't Be a Botball Controller if It Couldn't Be Hacked. Proceedings of the 2009 Global Conference on Educational Robotics, July 2009.
[2] J. Rand, M. Thompson. NHS Patchset. https://github.com/JeremyRand/cbc , June 2011.
[3] Wikipedia Contributors. De Morgan's laws. http://en.wikipedia.org/wiki/De_Morgan%27s_laws , May 2011.
[4] Botball Youth Advisory Council. Botball Community. http://community.botball.org , June 2011.
[5] J. Rand, M. Thompson, B. McDorman. CBC Hacking 2010. Proceedings of the 2010 Global Conference on Educational Robotics, July 2010.
[6] M. Thompson. CBCLua: Bringing Lua Scripting to Competitive Robotics. Proceedings of the 2009 Global Conference on Educational Robotics, July 2009.