

Smart Autonomous Java™ Technology Robot Arm

Teaching a robot arm to see, find,
and grasp an object

Bruce Boyes
Ben Holt

Systronix Inc
www.jcx.systronix.com

SYSTRONIX
Embedded Java Spoken Here

Smart Robot Arm

Can a useful arm be made for \$1000?

Use off the shelf mechanics and JCX electronics to create an arm for educational and experimental use.

This is a report on Arm1, our first attempt.

What's different about this arm?

- Robust – metal, not plastic. Motors must last too. Withstand steady use, 8 hours/day at conferences.
- Easy to program for students
- Use standard interfaces and APIs.
- Write all the code in Java
 - Prove it can be done, esp realtime aspect
 - Easy to maintain
- Develop Java algorithms for this task

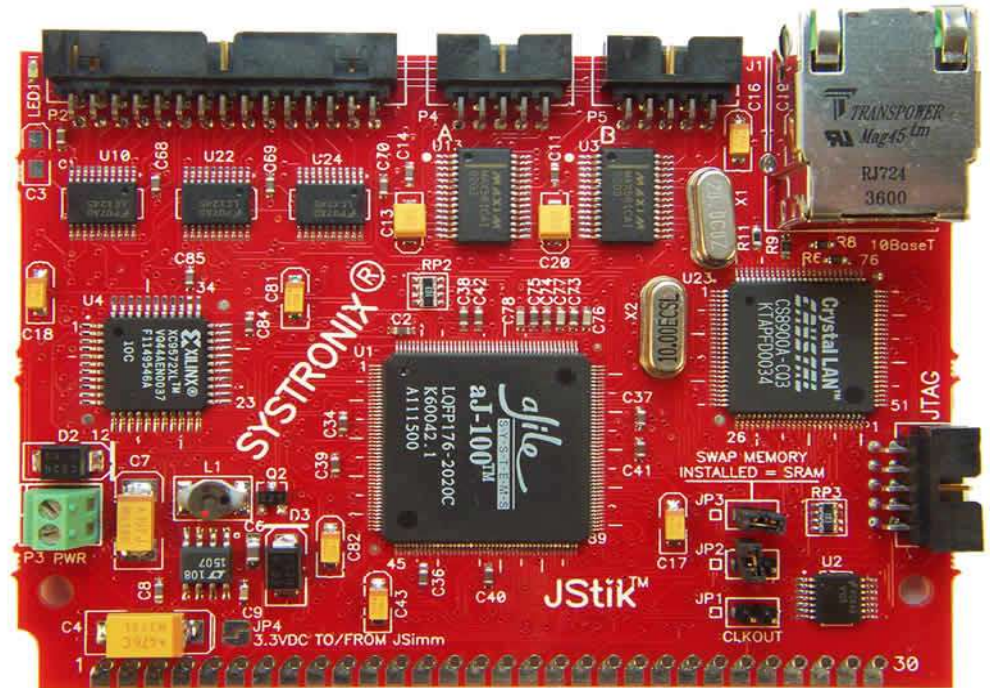
Smart Robot Arm

What we used

JCX

JStik controller

- Native Java
- 20 Mbcps
- 2 Mb SRAM
- 4 Mb flash
- JSimm bus on bottom edge
- HSIO bus
 - 50 Mbytes/sec, variable timing
 - We don't use it on this arm
 - Would be ideal for a “real” camera



Robix Hardware

Links and Servos

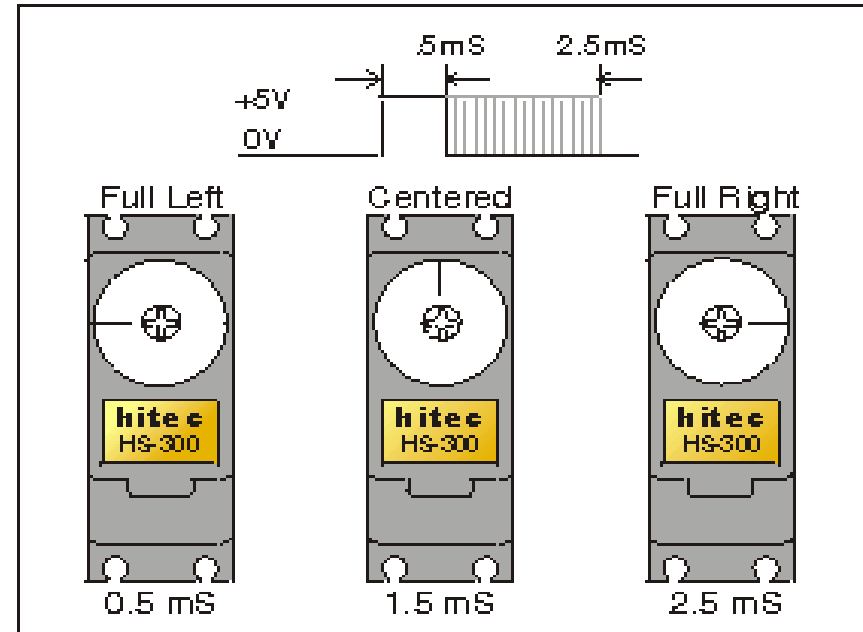
- Aluminum Links
 - Clamps & mounts
- R/C Servos
 - Ball-bearings
 - Hand-picked
- Center pivot & base
- New Java-based PC controller
 - We didn't use this here
- www.robix.com



R/C Servos

PWM, with a twist

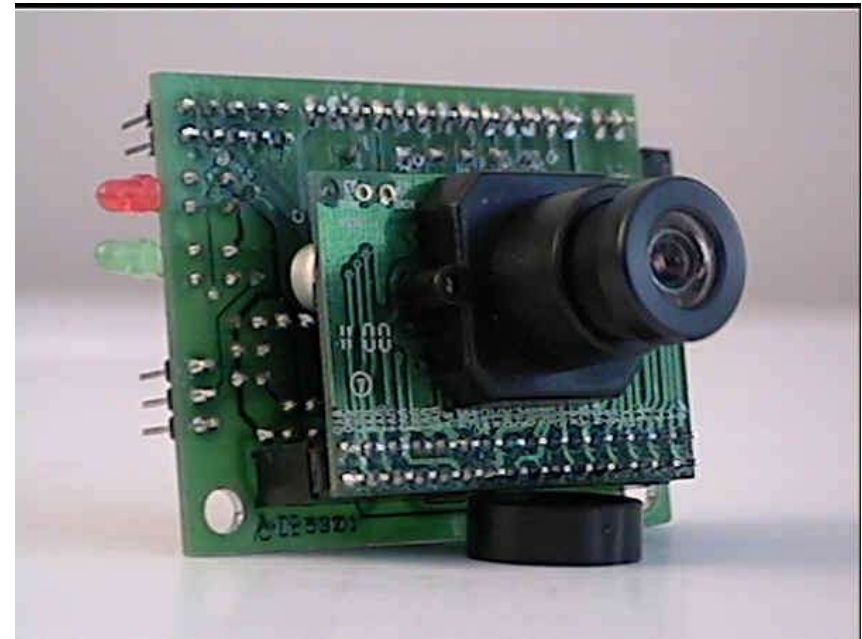
- 0.5 – 2.5 msec pulse
 - 0.7 to 2.3 realistic
- Rep rate > 50 Hz
 - Determines torque
- Quirky behavior
 - Deadband keeps servos from hunting but also adds, well, a *dead* band... no response there
 - Hysterisis – you don't get back to the same place by retracing your exact steps
 - Nonlinearity if > +/- 45 degrees
 - Hard stops before +/- 90, current goes way up
 - No soft start or external feedback



CMUcam1

Color vision sensor

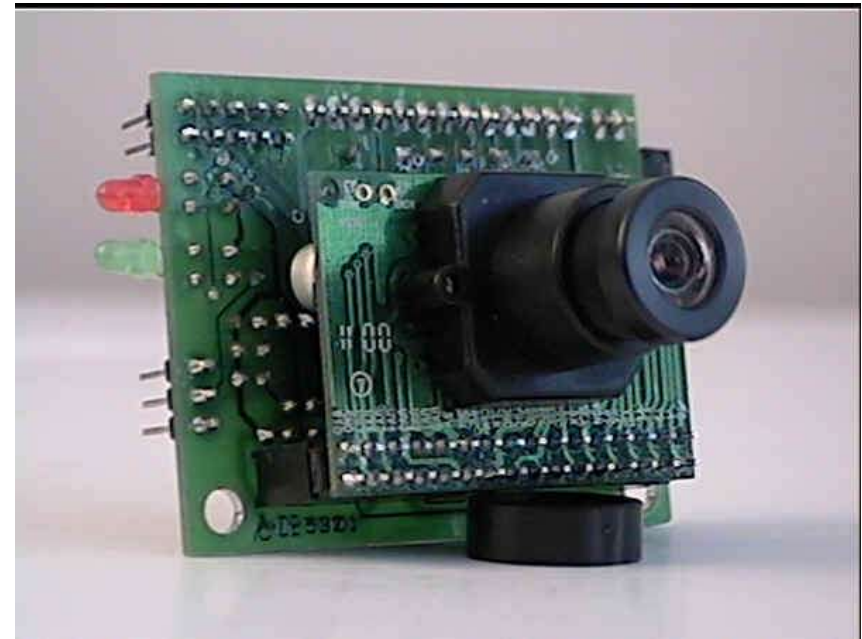
- Good news/bad news
 - It's \$110
 - Its a toaster – 200 mA
- Serial interface
 - 115 kbaud
 - PC demo/exploration software
- Helpful primitives
 - Color blob tracking, auto gain, white balance
- Shortcomings
 - Lens mount off center, lens distortion, colors are not what we see but what a CMOS sensor does



CMUcam1

Performance Limits

- Tracking
 - 17 fps
- Image transfer
 - Several seconds
- Resolution
 - 80 x 140 as a practical matter
 - This is enough for many tasks
 - Resolution vs processing – at least N^2
 - How smart is a bee? (pretty smart it turns out)



Graphic LCD

Matrix Orbital

- Graphic LCD
 - 122 x 32
- Serial interface
 - 19200 baud
 - Javaxcomm stream
- Helpful primitives
 - Local image screen storage
 - Fonts, mixed images/text, etc
- Other models
 - Bright vacuum fluorescent
 - Pushbuttons, etc



Software

Software

- Java
 - Javaxcomm serial I/O for CMUcam and LCD
 - Ajile runtime – J2ME/CLDC 1.0
- Robot APIs
 - “Cielguard” CMUcam API
 - JCX APIs for tagging memory
 - SPI access package
 - ServoMaster - uses PT with interrupts

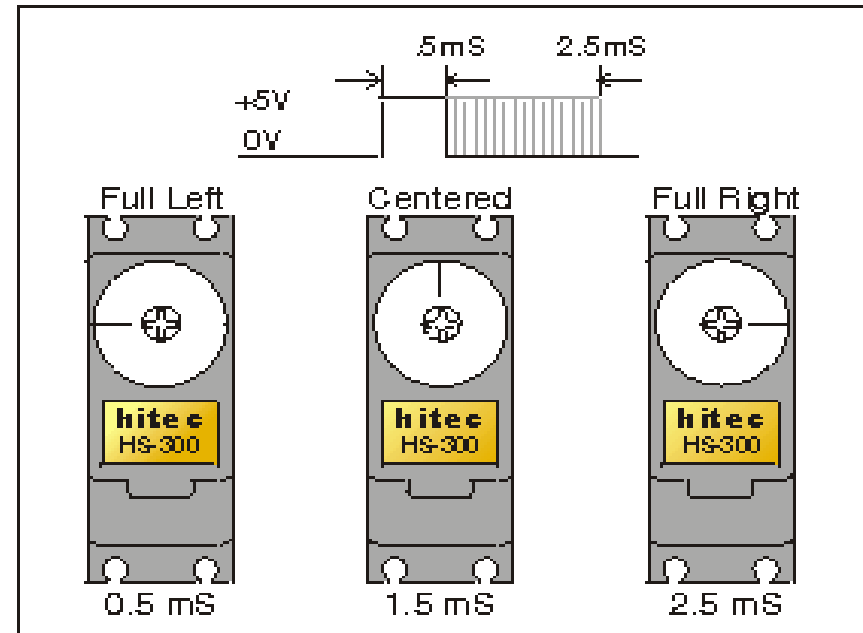
Smart Robot Arm

Approach to the problem(s)

R/C Servos

What they need

- Resolution
 - 10-12 bits is optimal, this is the granularity of the pulse width
- Rep rate
 - 50 Hz needed for good torque. More than about 70 Hz has no additional benefit.
- What matters
 - Exact rep rate is not critical. Precise pulse width and smooth changing is very important.



Servo Control

Some Real -Time Java

- Bounds
 - 2.5 msec tick x 8 servos = 20 msec, so we have 50 Hz refresh of all 8 servos
 - 2.3 msec max pulse width gives 200 usec for processing time at end of pulse
- How we did it
 - Array of pulse widths, one per servo.
 - PT with 2.5 msec period, start the output pulse, start timer of pulse width, then cycle()
 - Interrupt when timer overflows, handler terminates pulse and increments array to next pulse width value, then exits.
 - Lather, rinse, repeat.

PianoRoll – list of PeriodicThreads

```
/**
 * 2.5 msec duration and 2.5 msec beat
 * But on the scope this comes out with a 2.44 msec period,
 * this is under by 60 usec.
 */
PianoRoll pr = new PianoRoll(2, 500000, 2, 500000);

PianoRoll.start();

// "Start" periodic threads.
servo.start();

System.out.println("Periodic thread started");
}
```

PeriodicThread

```
/**
 * This class implements a thread which fires every 2.5 msec,
 * then we set a timer to generate a single pulse. Then we incr a counter and cycle().
 * When we wake up again we generate a pulse for the next servo.
 * As long as its timing is deterministic, we can null out the time it takes
 * to update the SPI register outputs. Testing verifies that this is indeed true.
 */
class PT1 extends PeriodicThread implements InterruptEventListener {

    /** We want a 2.5 msec period */
    static final int PERIOD_MSEC = 2;
    static final int PERIOD_NSEC = 500000;
    static final int INIT_DELAY_MSEC = 0;
    static final int PRIORITY = 12;          // preempt the GC, internal prio = 12*2 + 1

    // The timercounter we use to generate the pulse width
    static TimerCounter widthCounter;

    // Array to hold the width of pulses for eight servos
    static int servoPulse [] = new int [8];

    PT1 ()
    {
        makePeriodic(PERIOD_MSEC, PERIOD_NSEC, 0, 0, PRIORITY, this);
        // use PCLK as timer master clock, PCLK is system clock divided by two
        TimerCounter.setPrescalerClockSource( TimerCounter.INTERNAL_PERIPHERAL_CLOCK );
    }

    // Lots of other stuff left out, such as the interrupt handler (on a following slide)
}
```

PeriodicThread run()

```
/**
 * This method is invoked every 2.5 msec by PianoRoll
 * Each time we awake we step to the next servo in our array of servos
 * This code must complete in less than 2.5 msec.
 */
public void run() {

    System.out.print("Thread pT1 running...");

    for (;;) {
        // so we can time things on the scope
        ServoTest.pin.setPinState(true);

        servoPulse[currentServo] -= incr;

        widthCounter.setCurrentTimeRegisterValue(servoPulse[currentServo]);
        widthCounter.setMasterTimerEnabled(true);

        // Set the bit of the servo which we are driving at this moment
        // This is grossly simplified and a lot of detail is missing here
        spidata = 0x01 << currentServo;

        ServoTest.spi.sendData(spidata);
        ServoTest.pin.setPinState(false);
        cycle(); // Block until next wakeup.

    }
} // end of run method
```

Interrupt Handler

This code is actually part of Class PT1

```
/**
 * Turn off the servo pulse output.
 * Increment the currentServo value so we're ready for the next one
 */
public void interruptEvent () {

    currentServo++;
    if (currentServo >= activeServos)
    {
        currentServo = 0;
    }

    //clear the interrupt
    widthCounter.resetTimeoutInterruptStatus();

    //turn off the servo pulse pin (actually all servo outputs are off)
    ServoTest.spi.sendData(0x0);

} // end of interrupt event
```

Mechanical Layout

What goes where

- Camera Placement
 - Depth of focus
 - Angular resolution = $k * \text{distance from target}$
 - Keep arm from blocking camera's view
- Arm Design
 - 'Picker' vs. 'Scooper' arms
- Trapezoid lifter
 - 2-Servos, here deadband is actually good
- LCD Placement
- Lighting
 - avoid backscatter into camera

Modeling a human

Simple Algorithms

- Memorization
 - Humans use predefined motions
- Decisions
 - What should I do now?
 - Did I complete the objective?
- Adjustments
 - Do I see straight?
 - Controlled Movements
- Interaction
 - How do I communicate?
 - How do you communicate with me?

Modeling a human

How we did it

- 6 predefined Arm pick up positions
 - Linear interpolation in between
- Decisions by Priority
 - Subsumption Architecture
 - Priority based State Machine
- Adjustments as needed
 - On startup, verify centroid of known object.
 - Trapezoidal Velocity
- Communication
 - LCD Display which spells out current task.
 - Robot's "head" will shake no, if out of reach

Smart Robot Arm

Results

What went right

I find that the harder I work, the more luck I seem to have - John Heywood

- Java benefits
 - Low-level camera and servo APIs were completed first, then became transparent to rest of code
 - Leveraged existing CMUcam software
 - Exception handling used to our advantage
 - Rapid development vs C and assy
- Our behavior modeling proved correct (whew)
- Modular hardware
 - Add / remove components to track down problems
 - Build one block at a time
- Plug and play subsystems enable sophistication of the complex whole

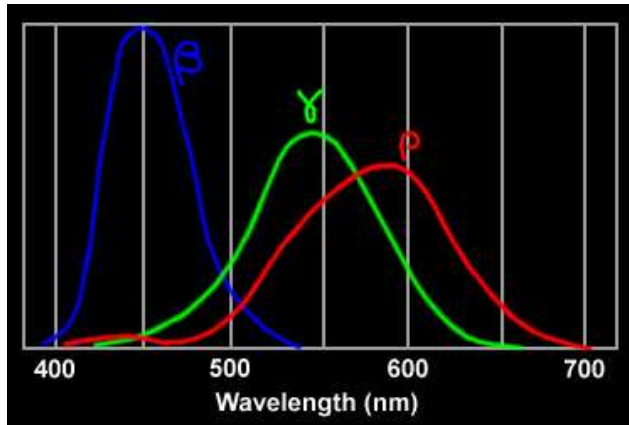
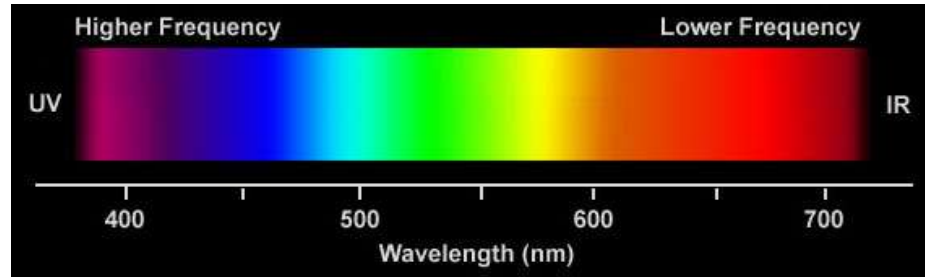
... and not so right

The difference between theory and practice, while small in theory, is large in practice

- CMUcam1
 - what can you expect for ~ 100 USD?
 - Optics are mechanically unstable
 - Lighting – our world is not always 18% gray
 - White LEDs aren't
 - Our eyes are not CMOS sensors (WYSAWYG)
 - CMUcam is a single point of failure
 - Easily fooled – orange paint on table or a sticker
- Those !@\$%\$ wires
- Feature creep
 - Just say “no”

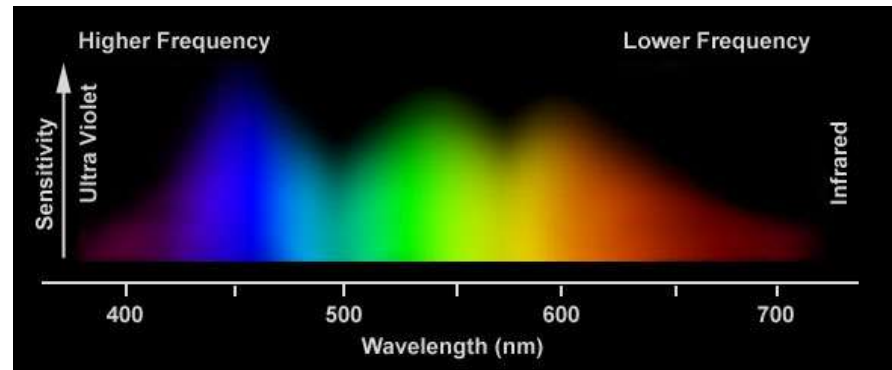
How our eyes perceive color

Visible color spectra:



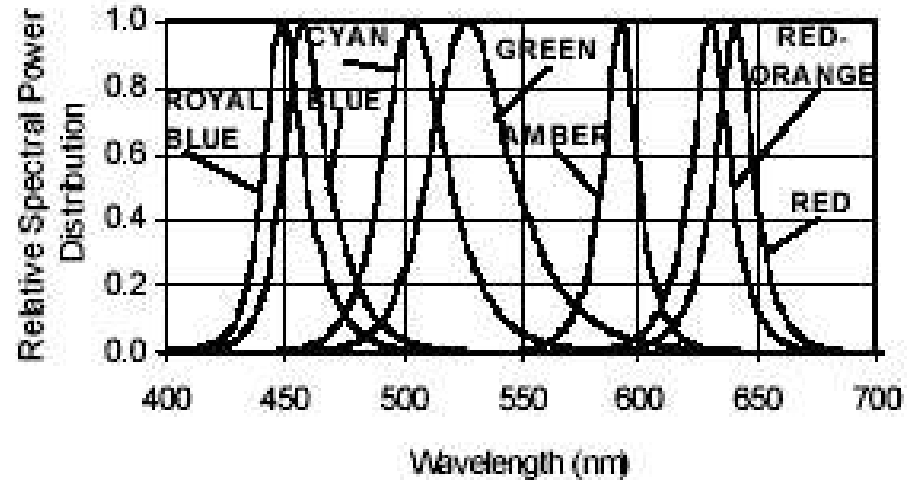
Eye's sensitivity

Color spectra as seen by human eye:

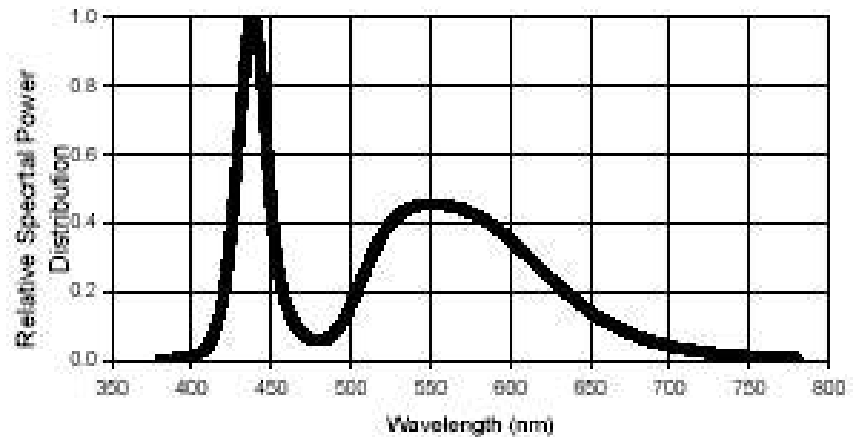


Why white LEDs aren't

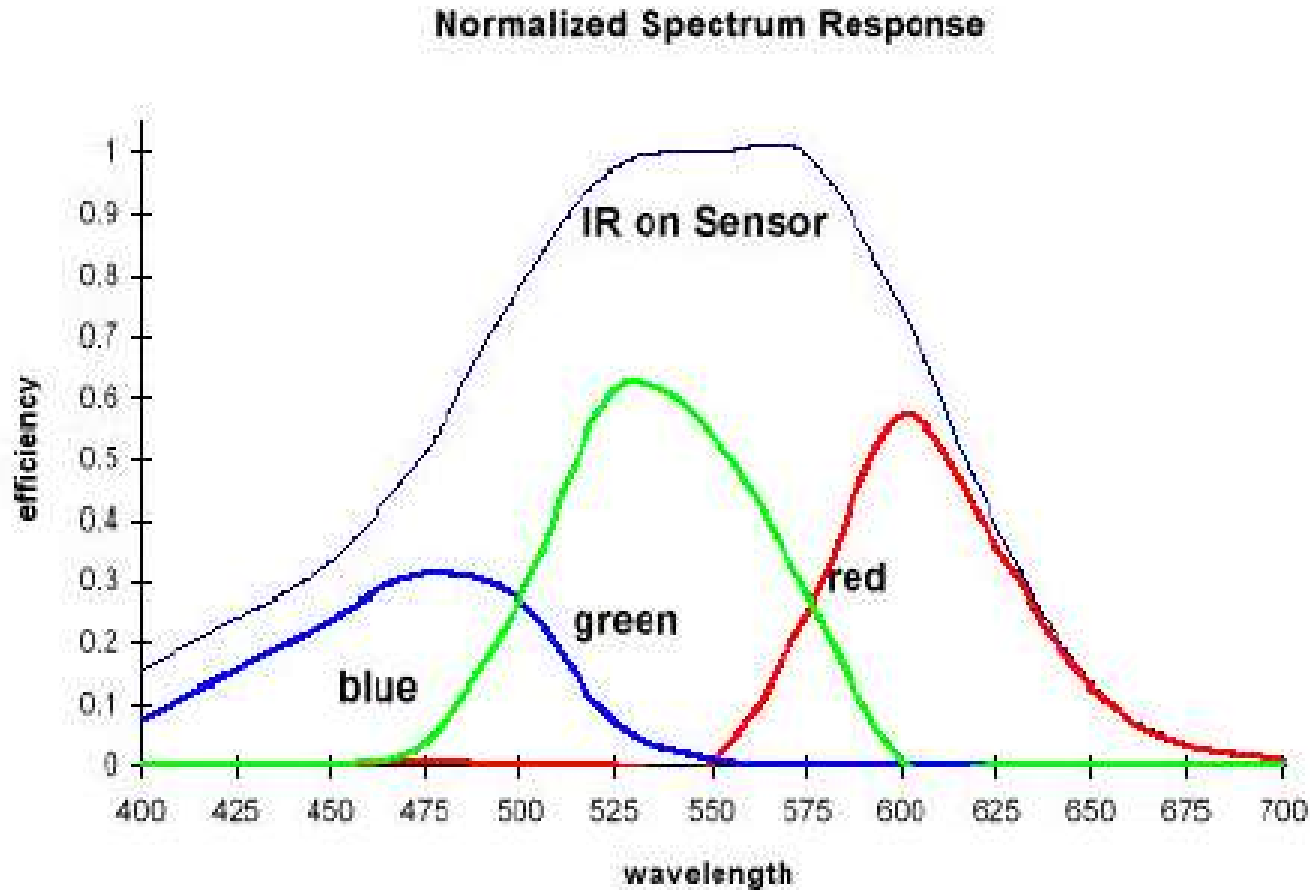
Typical LED spectra:



“White” LED spectrum:



What the Camera sees



OV6620 spectral response

(This diagram is offered by the manufacturer without explanation of what the “IR on sensor” curve means.)

What we learned

Exploring the world through “his” eyes

- Interaction is 90% of the cool factor
 - Smart is relative
 - Robot's “head” shaking is more impressive to most people than complex, hidden math
 - People refer to a disembodied arm as “he”?
 - Simple done well is better than sophisticated done poorly
- Vision is difficult...
 - It's easy to come to the wrong conclusions as you are analyzing and debugging. (Experimenting and reading data sheets helps combat this.)
- ...and so is robotics

Arm2 – Try Harder

We're working on this now -- JavaOne 2005?

- Sensor additions
 - CMUcam2
 - Sonar on the head
 - Short-range IR rangefinder on gripper
 - Force sensitive resistors on gripper
- AI
 - Auto calibration
 - Adaptive learning
- New interaction
 - Search Algorithm
 - “Bored” functionality (track someone)
 - Pick an item from the air in 3d space

Demo

OK, show me the arm!



For More Information

- Technologies
 - JCX: <http://www.jcx.systonix.com>
 - Arm links: <http://www.robix.com>
 - CMUcam: <http://seattlerobotics.com/>
 - LCD: <http://www.matrixorbital.com>
 - Native Execution Java: <http://www.ajile.com>
 - CMUcam API: <http://www.cielguard.com>
- Misc Techniques
 - <http://www.PracticalEmbeddedJava.com>

Smart Autonomous Java™ Technology Robot Arm

Comments, queries, quips, questions?

SYSTRONIX
Embedded Java Spoken Here