



Sony's Aibo

Vision

Robotics Seminar CSI445/660
Spring 2009, MW 4:15-5:35
Instructors: Tomek Strzalkowski,
Nick Webb & Michael Ferguson
University at Albany, SUNY



Administrative

- HW2 due 2/25 (Wednesday after break)
- Robot lab open next week W10-2
- Final project info out Monday 2/23, quick overview so you can think over break...



Overview of Vision

- Need to find “objects” of interest
- Throw out unnecessary information
- Computationally expensive.

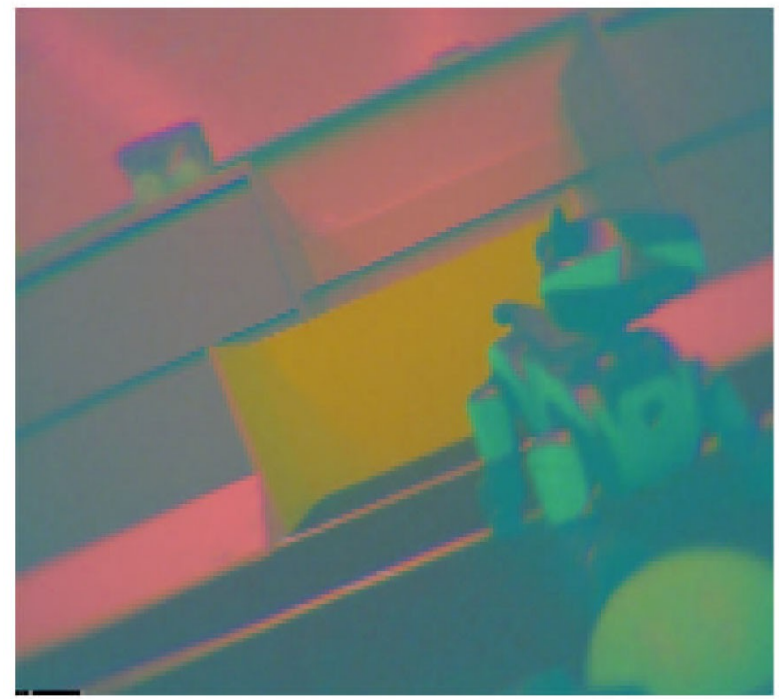


Color Spaces and AIBO

- The AIBO captures images through a 176x144 pixel CMOS camera.
 - Images are received natively in the YUV color space.
- Each pixel has a 3-dimensional value.
- The dimensions of this value are often called channels.
- This value can be represented in many different ways [2]. Two of concern here are:
 - RGB - R=red, G=green, B=blue
 - YUV - Y=brightness, UV=color
- Hopefully, RGB is familiar to everyone. The YUV space is very popular in that it's easy to process and models nicely to *human perception*.
 - YUV color space used by TVs and JPEGs.
 - Notice that YUV is *not* human perception.



RGB vs. YUV [1]





Segmented Vision

- The AIBO perceives vision of the world through a head mounted camera in the YUV color space.
- We've defined our use of vision as discriminating between objects we care and do not care about.
 1. How do we separate objects?
 2. How do we define those objects we care about?
 3. How do we make the AIBO recognize those objects?
- **Segmentation!**
 - Segmentation defines objects by color and intensity
 - A map from raw images to segmented images can be defined
 - The AIBO can then be "trained" to recognize certain objects in a segmented image



Segmentation Problems

- **Segmentation is done by pixel value**
 - Regions of the same or similar color can become indistinguishable.
- **Segmentation of images happens in the YUV space.**
 - Something that appears distinct to us may be very similar to the AIBO
 - When we train vision, be sure to keep in YUV.
 - RGB will work...why is YUV better?
 - How our system is calibrated plays a direct role in what the AIBO sees.
 - Lighting changes will affect recognition.
 - Different light intensity can reflect very different pixel values for the same object.
 - Keep the “target environment” in mind when calibrating your project.
- How can these problems be overcome?



The Process

1. Calibrate (Train) the individual Tekkotsu project according to the lighting and environment suitable for the Behavior's task.
 - Create suitable color mappings through the use of test images and either the TileTrain or VisionTrain java tools.
 - Apply the generated configuration to the Project through tekkotsu.xml and the appropriate startup files.
- Create or identify a DetectionGenerator that is capable of separating the wanted object from noise and generating a *VisionObjectEvent*.
- Register for the appropriate VisionObjectEvent within your Behavior that corresponds to the target object.
 - Behavior can now make decision based on what it "sees".



Quick Example

- Let's quickly trace ball detection using the vision pipeline diagram:
 - <http://www-2.cs.cmu.edu/~tekkotsu/VisionSetup.html>



Calibration

- Arguably the most important step
- These sources are quite helpful
 - <http://www-2.cs.cmu.edu/~tekkotsu/CameraSetup.html>
 - <http://aquarius.ils.albany.edu/robotics/SegmentedVisionIn1>
- Calibration is a two step process
 1. Camera Calibration
 - Applying camera settings to work well in the target environment.
 2. Threshold Mapping
 - Setting up the segmentation part of the vision



Camera Calibration

- **Shutter & Gain.**
 - Anyone familiar with Photography or sound processing should find this fairly intuitive.
 - Shutter speed is how quickly our “lens” opens & closes
 - Slower = Longer Speed = More data per frame
 - shutter_speed slow | mid | fast
 - slower shutter will brighten image, but increases motion blur
 - Gain is how quickly source data “flows” into the camera.
 - Higher = More data per frame
 - gain low | mid | high
 - higher gain will brighten the image, but increases noise
 - We can preview the results of certain settings though TekkotsuMon’s text-input field.
 - !set vision.shutter_speed=slow
 - !set vision.gain=high
 - These are set permanently in ms/config/tekkotsu.cfg
 - Experiment to find optimum settings for your environment



Threshold (Color) Mapping

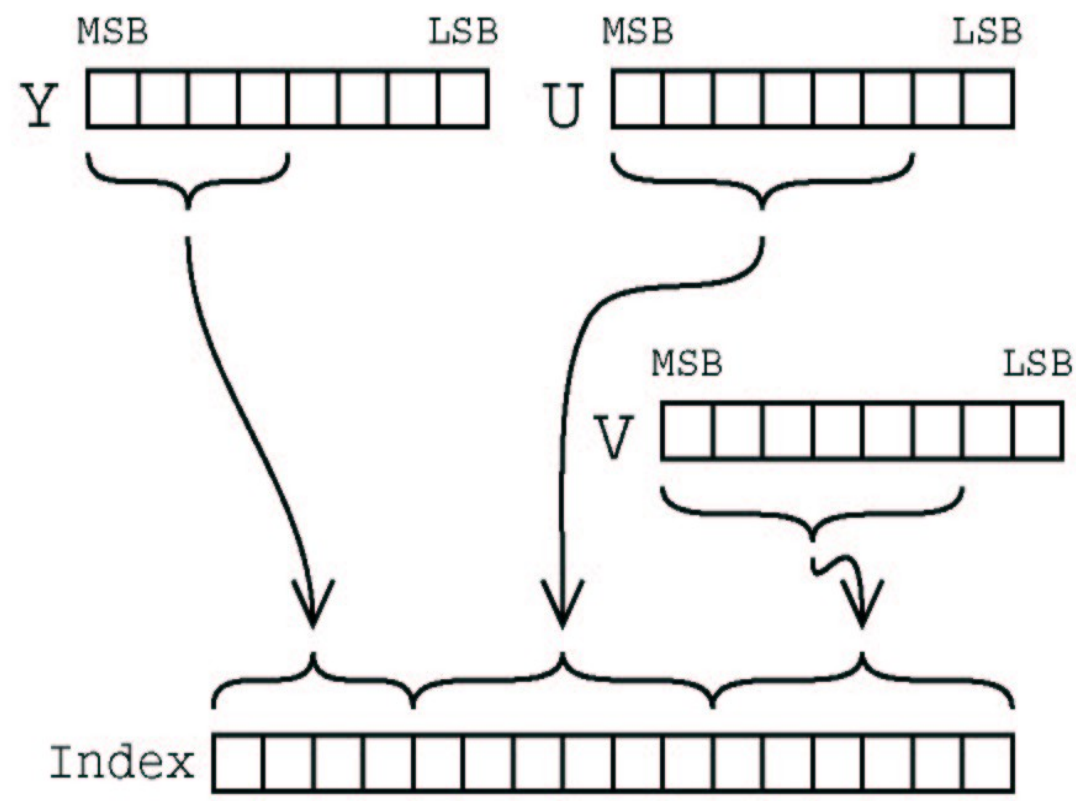
- This is a bit more work than the camera calibration.
- Step-by-step is covered in the reference docs and the in-class demonstration.
- The Idea:
 - Given a YUV image, every possible color in that space can be described in $16 (Y) * 64 (U) * 64 (V) = 65536$ bits (256^3 colors).
 - We want to limit that data to something manageable both in terms of bandwidth & CPU time.
 - The object of color segmentation is to map from raw camera pixels to a member (c) of the class of symbolic colors (C), i.e. to create a function $F[1]$:

$$F(y, u, v) \rightarrow c \in C, |C|=9$$

- This reduces the size of the target domain from 256^3 to 9.



Diagrammatic Look [1]





Creating a calibration

- Make sure that the required readings a few slides back have been read.
 - Steps:
 1. Create good sample images
 - Use TekkotsuMon VisionGUI to create PNG from Aibo's camera
 2. What makes a sample image good?
 3. What makes a good sample set?
 - 2. Create the mapping and color files using TileTrain (recommended) or VisionTrain.
 - 3. Modify the SetupVision behavior and tekkotsu.cfg to properly load the settings on startup.
- [Class Demo]



Detection Generators

- Uses segmented color region information to detect objects.
- Expects its event to come for a RegionGenerator (or compatible subclass)
- Inherits from EventGeneratorBase
 - It defines an event (recall an Event is a Behavior)
- Most parameters are setup in constructor
 - `BallDetectionGenerator(EventBase::EventGeneratorID_t gid, unsigned int sid, unsigned int mysid, unsigned int colorIdx, unsigned int threshmapChan, unsigned int noiseFiltering, float confidence);`
- `processEvent` verifies it's event's origin, looks over it's contents, and decides whether or not to post a "Detection" event.
- [`BallDetectionGenerator` example]



More Detection

- **So, The DetectionGenerator's job should be:**
 1. to sort through the returned regions from the lower levels
 2. filter the noise
 3. and make an informed decision about the region's content
- **The Ball Detection makes simple use of a minimum area and a circumscribed area or a bounding box against a possible circle's area to detect whether or not something is a ball.**
 - What are the benefits / counter-benefits here?
- **Writing detection generators is not always necessary.**
 - If the ball generator identifies what you want, use it.
 - The ball generator can be parameterized by the color channel you want
 - Try experimenting with minimum area and confidence calculation to optimize noise filtering or to identify larger objects.
 - But as an object gets close, how can a large object be separated from a smaller one?



Handling vision in the Behavior

- **Up to this point:**
 - We've calibrated our AIBO to the target environment
 - We've correctly set the camera settings to achieve a good picture
 - We've processed a thorough sample set to achieve a calibration and applied it to the Tekkotsu project.
 - We've created a Detection Generator capable of looking at RLE vision regions returned from the lower level and deciding if they conform to some version of our target object.
 - This generator throws an event our behavior can trap.
- **Now we need to implement this in our behavior**



Handling vision in a behavior

- Make sure to include any used DetectionGenerators headers in the behavior.
- The event posted from the generator is going to be a VisionObjectEvent.
 - But those get posted constantly
 - The Generator specified a type and source id as part of the event to give us some control.
- Registering for the event looks as such:
 - `erouter->addListener(this,EventBase::visObjEGID,ProjectInterface::visPinkBallSID)`
 - Recall that part of applying our calibration to the project was to create a new unique integer ID inside the ProjectInterface namespace
 - That number can be arbitrary, but non conflicting (aim high)



More behavior

- Inside of processEvent...
 - if (event.getGeneratorID()==EventBase::visObjEGID && event.getTypeID()==EventBase::statusETID) { // do stuff }
 - The Detection Generator that posted this event sets the TypeID to statusETID as well as setting the sourceID to whatever particular color (or object or both) was specified in the constructor of the Generator class.
 - Recall the following lines from SetupVision:
 - unsigned int pinkIdx=segcol->getColorIndex("red");
 - if(pinkIdx != -1U) {
 pball = new BallDetectionGenerator(
 EventBase::visRegionEGID,visRegionSID,
 visPinkBallSID,pinkIdx,threshChan,
 noiseFiltering,confidenceThreshold);
 pball->setName("PinkBallDetectionGenerator"); }
 }



VisionObject properties

- Any data that a given behavior has about the object is passed via the VisionObject event.
- Examples:
 - `horiz=static_cast<const VisionObjectEvent*>(&event)->getCenterX();`
 - `vert=static_cast<const VisionObjectEvent*>(&event)->getCenterY();`
- `getCenterX()` & `getCenterY()` are really the only properties given back by default.
 - Returns a float value between -1 and 1 relative to (0,0) at center of AIBO's view. **CHECK THIS**
- This can be extended through extending the VisionObjectEvent class and posting the new event in the Detection Generator.
 - What other information might be useful ?



Helpful Links

- **EasyTrain Tutorial**
 - <http://www.cs.cmu.edu/~dst/Tekkotsu/Tutorial/colc>
- **Spletzer's Lecture**
 - http://www.cse.lehigh.edu/%7Espletzer/cse398_Sl
- **Turner's Tutorial**
 - <http://www.cs.cmu.edu/~tekkotsu/media/SegmentedVisionInTekkotsu.pdf>



Summary

- We've defined vision as throwing out the information you don't want, while keeping the information you do want.
- Tekkotsu is composed of low and high level vision
- High level vision allows us to abstract the features of objects we need to identify, and then compare those features against data passed up from the low level.
- Object identification is accomplished via RLE of the images to create segmented vision.
- Segmented vision has limitations.
- Tekkotsu's vision system needs to be calibrated before putting it to real work.
- Creating a DetectionGenerator for an object is the means by which we can define target features.
- The vision event that's eventually trapped by a behavior is generated inside this detection generator and inherits from VisionObjectEvent.



References

1. CMU Robobits Week 4 lecture “Vision-Low Level”
 - <http://www-2.cs.cmu.edu/~robosoccer/cmrobobits/>
2. CMU Robobits Week 5 lecture “Vision-high level”
 - <http://www-2.cs.cmu.edu/~robosoccer/cmrobobits>

Thanks to:

- Shawn Turner
- Russell Goldstein