OpenCV / AR.Drone

Two-part Presentation

CS 4768 - Robot Learning

Cooper Bills

OpenCV - What is it?

"(**Open** Source **C**omputer **V**ision) is a library of programming functions for real time computer vision"

- http://opencv.willowgarage.com/wiki/

The library has >500 optimized algorithms

Intel was heavily involved from the beginning

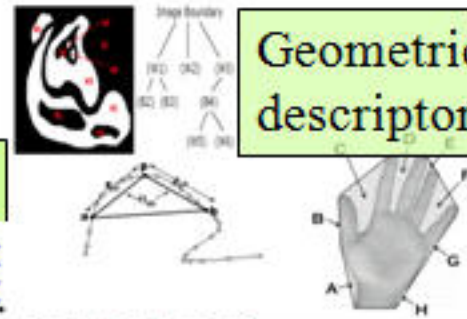
OpenCV Overview: > 500 functions

opencv.willowgarage.com

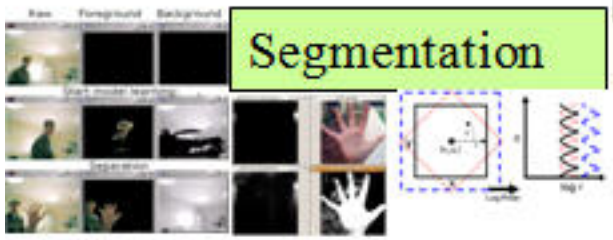
General Image Processing Functions

Image Pyramids

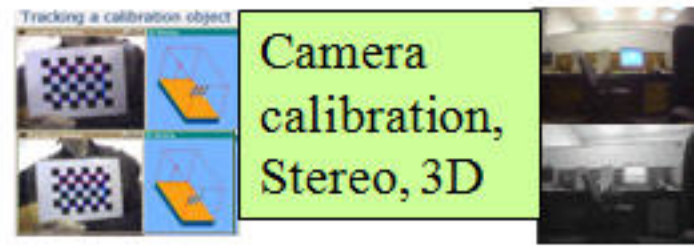
Geometric descriptors



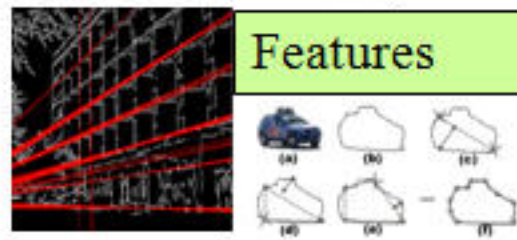
Segmentation



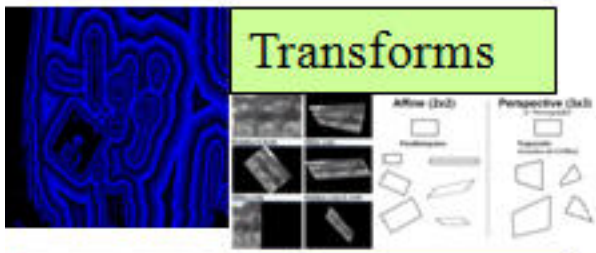
Camera calibration, Stereo, 3D



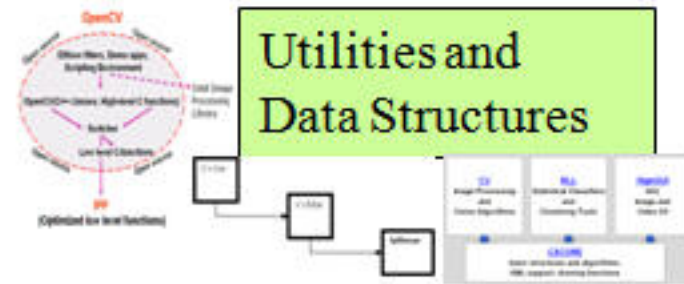
Features



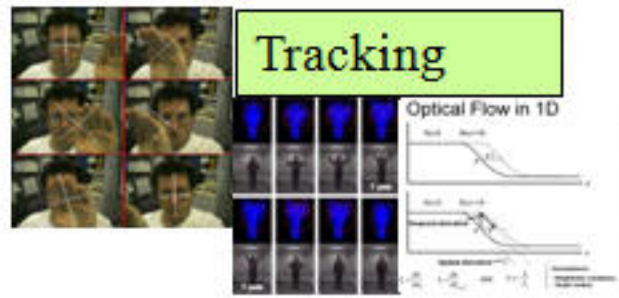
Transforms



Utilities and Data Structures

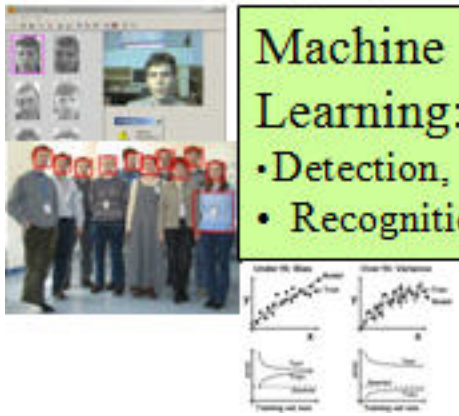


Tracking

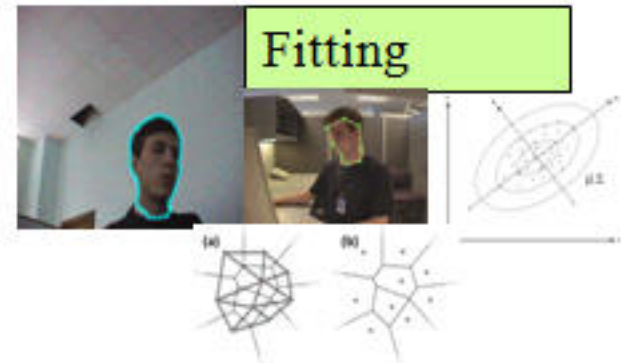


Machine Learning:

- Detection,
- Recognition



Fitting



Matrix Math

OpenCV - Installing

Ubuntu:

'sudo apt-get install libhighgui-dev'

Will install version 2.0 (newest version is 2.2)

For other operating systems or newer versions:

http://opencv.willowgarage.com/wiki/InstallGuide

OpenCV - Hello Webcam!

Now, we'll create a simple OpenCV program!

Goal: Display a video stream (this case a webcam)

Approach: Take an image from the webcam at ~30Hz
and display this image on the screen.

OpenCV - Hello Webcam!

```
/**********************
 * Hello Webcam
 * OpenCV Demo
 * Cooper Bills (csb8@cornell.edu)
 **********************/

#include <stdio.h>
#include <opencv/highgui.h> // highgui.h contains openCV gui elements

int main()
{
  CvCapture* captureDevice; // our webcam handler
  IplImage* currentFrame; // where we'll store the current frame
```

OpenCV - Hello Webcam!

```
captureDevice = cvCaptureFromCAM(0); // get webcam handler.
                                     // 0 is the camera index.  If your
    // computer has more than one
    // capture device, the index
    // picks which camera to use.

  // check to make sure there are no problems
  if(!captureDevice) {
    printf("Error opening webcam. Exiting. \n");
    return -1;
  }

  // Create a window to display our webcam in, The handler for this
  // window is also the name, in this case "myWindow"
  cvNamedWindow("myWindow", CV_WINDOW_AUTOSIZE);

  int key = -1;
```

OpenCV - Hello Webcam!

```
// while there is no key press by the user,
while(key == -1) {
    currentFrame = cvQueryFrame(captureDevice); // get current frame

    // check for errors:
    if(!currentFrame)
        break;

    // Display the current frame in the window we created earlier
    cvShowImage("myWindow", currentFrame);

    // Wait for user input for 30ms, cvWaitKey return the user's
    // keypress, or -1 if time limit is reached. (0 = wait forever)
    key = cvWaitKey(30);
}
```

OpenCV - Hello Webcam!

```
  // Appropreate Cleanup
  cvReleaseCapture(&captureDevice);
  cvReleaseImage(&currentFrame);
  cvDestroyWindow("myWindow");

  return 0;
}
```

< Now Lets Try It! >

OpenCV - The IplImage*

```
typedef struct _IplImage
{
    int  nSize;
    int  ID;
    int  nChannels;
    int  alphaChannel;
    int  depth;
    char colorModel[4];
    char channelSeq[4];
    int  dataOrder;
    int  origin;
    int  align;
    int  width;
    int  height;
    struct _IplROI *roi;
    struct _IplImage *maskROI;
    void  *imageId;
    struct _IplTileInfo *tileInfo;
    int  imageSize;
    char *imageData;
    int  widthStep;
    int  BorderMode[4];
    int  BorderConst[4];
    char *imageDataOrigin;
}
```

OpenCV - Manipulating Image Data

http://opencv.willowgarage.com/wiki/faq#Howtoaccessimagepixels

Easiest Way:

using the macro:

```
CV_IMAGE_ELEM( img, T, y, x*N + c )
```

where:

```
img = image to access data in
T = type of data (uchar, float, etc.)
y = y coordinate
x = x coordinate
N = number of channels
c = channel to access
```

OpenCV - Exercise: Invert Webcam

Given that cvCaputureFromCam() returns a *3-Channel*, *uChar (8-bit) image*, how could we create an inverting effect?

Remember: `CV_IMAGE_ELEM( img, T, y, x*N + c )`

(also note: uchar holds values within 0-255)

OpenCV - Exercise: Invert Webcam

Given that cvCaputureFromCam() returns a *3-Channel*, *uChar (8-bit) image*, how could we create an inverting effect?

Remember: `CV_IMAGE_ELEM( img, T, y, x*N + c )`

(also note: uchar holds values within 0-255)

// Invert the frame... for(int x=0; x<currentFrame->width; x++) for(int y=0; y<currentFrame->height; y++) { ...

OpenCV - Exercise: Invert Webcam

```
// Invert the frame
for(int x = 0; x < currentFrame->width; x++)
  for(int y = 0; y < currentFrame->height; y++) {

    int B = 255 - CV_IMAGE_ELEM( currentFrame, uchar, y, x*3);
    int G = 255 - CV_IMAGE_ELEM( currentFrame, uchar, y, x*3+1);
    int R = 255 - CV_IMAGE_ELEM( currentFrame, uchar, y, x*3+2);

    CV_IMAGE_ELEM( invFrame, uchar, y, x*3) = B;
    CV_IMAGE_ELEM( invFrame, uchar, y, x*3+1) = G;
    CV_IMAGE_ELEM( invFrame, uchar, y, x*3+2) = R;
  }
```

OpenCV - Exercise: Invert Webcam

Demo!

OpenCV - High Level Functions

OpenCV already has many higher level functions!

cvNot() - Bit-wise invert a matrix/image
cvLine() - Draw a line on an image
cvEqualizeHist() - Equalize an image's Histogram
cvSmooth() - Apply Blurring (Gaussian, Linear, etc.)
cvCanny() - Canny Edge Detection
cvSobel() - Apply a Sobel Filter to the Image
cvHoughLines2() - Hough Transform (find lines in an image)
cvHaarDetectObjects() - Apply a Haar Cascade to find objects
cvCalcEigenObjects() - Find Eigenvectors of a set of images

...Just to name a few

OpenCV - Saving and Loading Images

```
IplImage* cvLoadImage( const char* filename,
                       int flags=CV_LOAD_IMAGE_COLOR );

int cvSaveImage( const char* filename, const CvArr* image );
```

So, loading and saving images is easy!

```
IplImage* img = cvLoadImage("mypic.jpg");
//do stuff
cvSaveImage("mypic_modified.jpg", img);
```

OpenCV - Reacap

OpenCV does much of the image processing work for you.

Already implements ways of loading, saving, displaying, and manipulating images.

Provides a strong base to implement your own vision algorithms.

Many resources on Google to help you find functions you need.

I'll post this demo code as a useful starting point.

...Questions?

AR.Drone

AR.Drone - Basic Information

The Parrot AR.Drone is a toy quadrotor we use for research purposes.

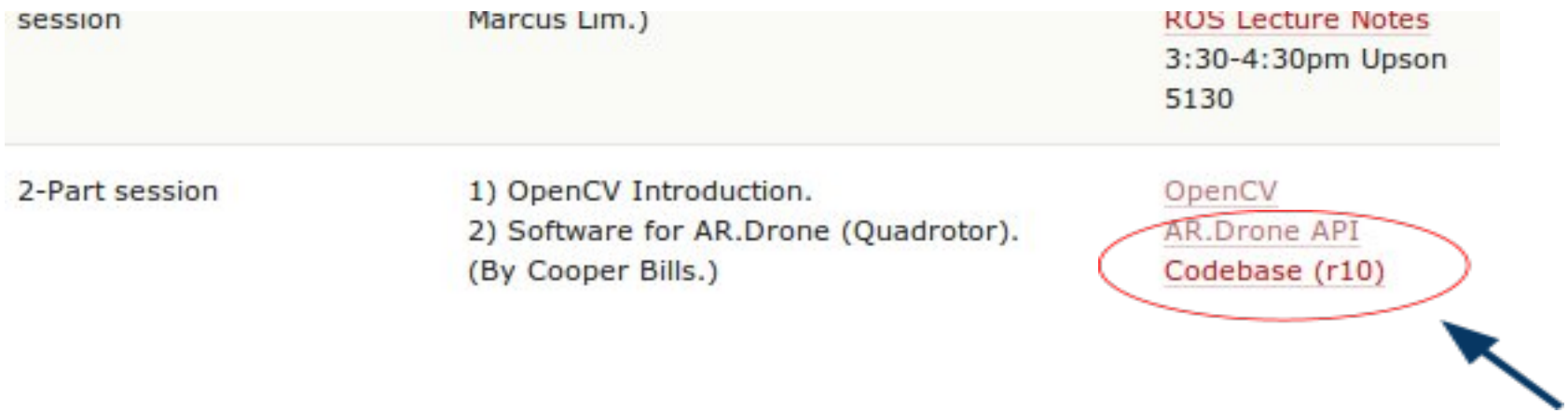
It has built in stabilizing algorithms, and has an API which we utilize.

Has two cameras: one front-facing, and one downward-facing.

AR.Drone - Interfacing

I have created a codebase for you to utilize. This includes the API and documentation to get you up and running.

Note: this codebase is designed to run on Ubunutu/Linux.



AR.Drone - Interfacing

Communicates over Ad-Hoc Wifi.

The codebase allows switching between manual and algorithmic control.

AR.Drone - Controls

We currently use an Xbox 360 controller for manual control:

Note: This image is included with the documentation.

AR.Drone - Controls

Four Degrees of Freedom are available via the API:

Pitch - Forward/Backward movements

Roll - Left/Right movements (Strafing)

Yaw - "Turning"

Gaz - Up/Down movements

AR.Drone - Controls

< Demo >

AR.Drone - Code

Codebase root folder:

application/ - Our code, and where your algorithms go
|--- **Build/** - Where our application gets made and run
|--- **Sources/** - The Code!

ARDrone_API/ - Their code; The AR.Drone API

Documentation/ - More documentation

Quickstart.txt - Basic Documentation

AR.Drone - Code

Codebase/application/Sources folder:

Navdata/ - Code for gathering drone sensor data
Tools/ - A few additional functions
(may worth checking out)
UI/ - Code for major UI components
(gamepad, keyboard, Algorithms etc.)
Video/ - Code for gathering video
xbee/ - Code for the xbee sensors
(ask me later if interested)

ardrone_testing_tool.c - code used by the API
ardrone_testing_tool.h

AR.Drone - Code (UI)

Codebase/application/Sources/UI folder:

gamepad.cpp	*- Code for Gamepad Control*
gamepad.h	
keyboard.cpp	*- Code for Keyboard Control*
keyboard.h	
planner.cpp	***- Code for Algorithmic Control***
planner.hpp	
terminalinput.hpp	*- Code for reading keyboard input (broken)*
ui.c	*- Code used by the API*
ui.h	

AR.Drone - Planner.cpp

This is the file where all of your algorithmic control goes.

It has it's own thread, that loops while the algorithms are active.
This loop is where you write your automated control code.

Basic Idea:

```
        //Current Algorithm Results
  int32_t dpitch, dyaw, droll, dgaz;
        //Final Algorithm Results
  int32_t dpitch_final, dyaw_final,
        droll_final, dgaz_final;
```

An external thread monitors the final values, sending them to the drone whenever they change

AR.Drone - Planner.cpp

< Demo >