Pose Estimation in OpenCV
Pose Estimation of a Known Model

- Assume we have a known object model, with five fiducial targets
- Each target is a “concentric contrasting circle”; it can easily be recognized by thresholding the image and finding connected components

From the 2D-3D point correspondences, we can determine the pose of the model with respect to the camera

• Let’s see how to do this in OpenCV
#include <iostream>

// This includes everything we need
#include <opencv2/opencv.hpp>

int main(int argc, char* argv[]) {
    cv::Mat imageInput = cv::imread("robot.jpg");

    if (imageInput.empty()) {
        printf("Hey! Can't read input file!\n");
        system("PAUSE");
    }
    imshow("input image", imageInput);

    // Wait for xxx ms (0 means wait until a keypress)
    cv::waitKey(0);

    return EXIT_SUCCESS;
}
Useful OpenCV Functions

- **cvtColor** – convert a color (BGR) image to grayscale
  
  ```cpp
  // Convert to gray
  cv::Mat imageInputGray;
  cvtColor(imageInput, imageInputGray, cv::COLOR_BGR2GRAY); // in older versions, use CV_BGR2GRAY
  imshow("Input grayscale image", imageInputGray);
  ```

- **adaptiveThreshold** – do a local adaptive threshold of the image
  
  ```cpp
  // Do adaptive threshold ... this compares each pixel to a local
  // mean of the neighborhood. The result is a binary image, where
  // dark areas of the original image are now white (1's).
  cv::Mat imageThresh;
  adaptiveThreshold(imageInputGray, imageThresh, 255, cv::ADAPTIVE_THRESH_GAUSSIAN_C, cv::THRESH_BINARY_INV, 91, 0);
  imshow("Binary image", imageThresh);
  ```
Useful OpenCV Functions

• **getStructuringElement** – create structuring element
  – Example

  ```cpp
  // Create a disk-shaped structuring element
  cv::Mat structuringElmt = cv::getStructuringElement(cv::MORPH_ELLIPSE,
            cv::Size(3,3));
  ```

• **morphologyEx** – do a morphological operation on the image
  – Example

  ```cpp
  // Apply morphological operations to get rid of small (noise) regions
  cv::Mat imageOpen;
  morphologyEx(imageThresh, imageOpen, cv::MORPH_OPEN, structuringElmt);

  cv::Mat imageClose;
  morphologyEx(imageOpen, imageClose, cv::MORPH_CLOSE, structuringElmt);

  imshow("Binary image after morph", imageClose);
  ```
“Vectors” in C++

- A “vector” class is part of the standard library.
- It is like an array, but its size can change at run time.
- Example:

```cpp
#include <vector>

int main(int argc, char* argv[]) {
    std::vector<int> myvector; // Allocate the empty vector
    int i = 12, j = 34, k = 56; // (need to specify the type)

    myvector.push_back(i); // “push_back” is how you
    myvector.push_back(j); // add elements
    myvector.push_back(k);

    for (unsigned int c = 0; c < myvector.size(); c++)
        printf("%d\n", myvector[c]);
}
```

Prints 12, 34, 56
Useful OpenCV Functions

- **findContours** — find contours on the boundaries of binary regions
  - OpenCV doesn’t have connected component labeling, so we use this

- It can return two types of contours:
  - “Exterior” contours surround white regions
  - “Holes” surround black regions
Useful OpenCV Functions

• **findContours (continued)**
  
  – The “RETR_CCOMP” option organizes contours into a two-level hierarchy, where the top-level boundaries are external boundaries of the components and the second level boundaries are boundaries of the holes

  ![CV_RETR_CCOMP Diagram](image)

  ```cpp
  // Find contours.
  std::vector<std::vector<cv::Point>> contours;
  std::vector<cv::Vec4i> hierarchy;
  cv::findContours(
    imageClose, // input image (is destroyed)
    contours,   // output vector of contours
    hierarchy,  // hierarchical representation
    cv::RETR_CCOMP, // retrieve all contours
    cv::CHAIN_APPROX_NONE); // all pixels of each contours
  ```
Useful OpenCV Functions

• **drawContours**
  
  - Draws contour outlines or filled contours.
  
  - Example

```cpp
cv::Mat imageContours = imageInput.clone();   // copy image

cv::drawContours(imageContours, contours,
  -1,       // contour id; -1 means draw all
  cv::Scalar(0,255,255),    // color
  2,        // thickness
  8);       // line type

imshow("All contours", imageContours);
```
#include <iostream>

// This includes everything we need
#include <opencv2/opencv.hpp>

int main(int argc, char* argv[]) {
    cv::Mat imageInput = cv::imread("robot.jpg");
    if (imageInput.empty()) {
        printf("Hey! Can't read input file!\n");
        system("PAUSE");
    }
    imshow("input image", imageInput);

    // Convert to gray
    cv::Mat imageInputGray;
    cvtColor(imageInput, imageInputGray, cv::COLOR_BGR2GRAY);
    imshow("Input grayscale image", imageInputGray);

    // Do adaptive threshold ... this compares each pixel to a local 
    // mean of the neighborhood. The result is a binary image, where 
    // dark areas of the original image are now white (1's).
    cv::Mat imageThresh;
    adaptiveThreshold(imageInputGray,
        // imageThresh, // output thresholded image
        255, // output value where condition met
        cv::ADAPTIVE_THRESH_GAUSSIAN_C, // local neighborhood
        cv::THRESH_BINARY_INV, // threshold_type - invert
        91, // BlockSize (any large number)
        0); // a constant to subtract from mean
    imshow("Binary image", imageThresh);
}
/// Apply morphological operations to get rid of small (noise) regions
/// cv::Mat structuringElmt = cv::getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(3,3));
/// cv::Mat imageOpen;
/// morphologyEx(imageThresh, imageOpen, cv::MORPH_OPEN, structuringElmt);
/// cv::Mat imageClose;
/// morphologyEx(imageOpen, imageClose, cv::MORPH_CLOSE, structuringElmt);
/// imshow("Binary image after morph", imageClose);

/// Now find contours.
/// std::vector<std::vector<cv::Point>> contours;
/// std::vector<cv::Vec4i> hierarchy;
/// cv::findContours(
///     imageClose,    // input image (is destroyed)
///     contours,      // output vector of contours
///     hierarchy,     // hierarchical representation
///     cv::RETR_CCOMP, // retrieve all contours
///     cv::CHAIN_APPROX_NONE); // all pixels of each contours
///
/// Draw all contours.
/// cv::Mat imageContours = imageInput.clone(); // copy image
/// cv::drawContours(imageContours, contours,
///                   -1, // contour id; -1 means draw all
///                   cv::Scalar(0,255,255), // color
///                   2, // thickness
///                   8); // line type
/// imshow("All contours", imageContours);

/// Wait for xxx ms (0 means wait until a keypress)
cv::waitKey(0);

return EXIT_SUCCESS;
}
Contour Within Contours

• We are looking for an exterior contour surrounding a hole

```cpp
// Find only those contours with a child inside.
std::vector<std::vector<int>> contours;
contours.push_back(hierarchy);
for (unsigned int i1 = 0; i1 < contours.size(); i1++) {
    int i2 = hierarchy[i1][2];
    if (i2 < 0) continue;  // See if it has child inside
    cv::drawContours(imageChildren, contours, i1,  // contour id; -1 means draw all
                     cv::Scalar(0, 255, 255),  // color
                     2,  // thickness
                     8);  // line type
}
```
Finding CCCs

• The OpenCV function “moments” computes values of

\[ m_{ji} = \sum_{x,y} (\text{array}(x, y) \cdot x^j \cdot y^i) \quad i, j = 0..3 \]

  – The centroid is given by

\[
\bar{x} = \frac{m_{10}}{m_{00}}, \quad \bar{y} = \frac{m_{01}}{m_{00}}
\]

• Check the "circularity" ratio of the outer region, which is the ratio of area to perimeter squared:

\[ R = 4 \pi A / P^2 \]

  – R is 1 for a circle, and \( \pi / 4 \) for a square
// Find CCCs.
cv::Mat imageOutput = imageInput.clone();
for (unsigned int i1 = 0; i1 < contours.size(); i1++) {
    int i2 = hierarchy[i1][2];
    if (i2 < 0) continue;  // See if it has a child inside

    // Compute centroids of inner and outer regions
    cv::Moments mu1 = cv::moments(contours[i1]);
    cv::Point2d x1(mu1.m10 / mu1.m00, mu1.m01 / mu1.m00);
    cv::Moments mu2 = cv::moments(contours[i2]);
    cv::Point2d x2(mu2.m10 / mu2.m00, mu2.m01 / mu2.m00);

    // Check if centroids coincide
    const double DPIXEL = 3.0;  // max distance between centroids
    if (norm(x1 - x2) > DPIXEL) continue;

    // Check the "circularity" ratio of the outer region, which is
    // the ratio of area to perimeter squared:  \( R = \frac{4\pi A}{P^2} \).
    // \( R \) is 1 for a circle, and \( \pi/4 \) for a square.
    double P1 = arcLength(contours[i1], true);
    double A1 = contourArea(contours[i1]);
    if (4 * 3.1415 * A1 / (P1 * P1) < 3.1415 / 4) {
        // Let's say that we want our region to be at least as round as a square.
        continue;
    }

    // Looks like a valid CCC. Draw a circle around it.
    cv::circle(imageOutput,
                x1,  // center
                3,   // radius
                cv::Scalar(0, 0, 255),  // color
                -1);  // negative thickness=filled
}
imshow("Detected CCCs", imageOutput);
Solving for Pose

• The OpenCV function “solvePnP” finds the pose of an object from 3D-2D point correspondences.
  – “PnP” stands for “Perspective n-Point” problem.

• Inputs:
  – A “vector” of 3D object points
  – A “vector” of corresponding 2D image points
  – Intrinsic camera parameters

• Outputs:
  – Rotation vector (axis of rotation, multiplied by angle)
  – Translation vector

This is the C++ vector (a dynamic array)

Camera matrix K

See “axis-angle” rotation convention, in Lecture 4a