ME 4451 Lab 5 Handout
2D Pose Detection and Tracking

Anirban Mazumdar, Bryan Blaise, Nikhil Nandikanti
(1 Session Lab)

Spring 2024

1 Objectives

1. Learn the basics of template based 2D Object Pose detection
2. Extend the detection to 2D Object Pose Tracking

2 Prelab Assignment

- Read Lab 5 Handout.
- Complete function $feducialCentroidDetection.m$.
- Familiarize yourself with the remaining tasks by outlining steps/functions needed in your code.
- Be able to describe in your own words the physical significance and the size (i.e. 3x3 array, 3x1 vector) of the following values:
  - p_0
  - p_{01}
  - p_2
  - p_{1230}
  - p_{123}
- Print out an image of the template image in template_printout.PNG to use it for live tracking.

3 Object Pose Detection

Knowing the object pose (i.e., position and orientation) is important in manipulation tasks. It could be used in closed-loop control in robotic tasks. In most applications, vision based pose detection is used. In this lab, we cover the basics of the 2D pose estimation and tracking. In practice this can be done with QR code, AR code templates.

4 Pose Estimation Basics

Figure 1, illustrates the sight from above of a robot with a simple template attached to its top surface. The goal is to obtain the robot's pose (position + orientation) at any given moment by detecting the displacement of the template using image processing. The pose is measured with reference to the global reference frame. In this case, we're treating the original location of the robot/template as the global origin of this system.

Here, we're demonstrating the robot performing two different motions, leading to the template being detected at 2 different locations of the camera's field of view. This results in two different poses/transformations between the global reference frame/origin and the reference frame attached to the robot.

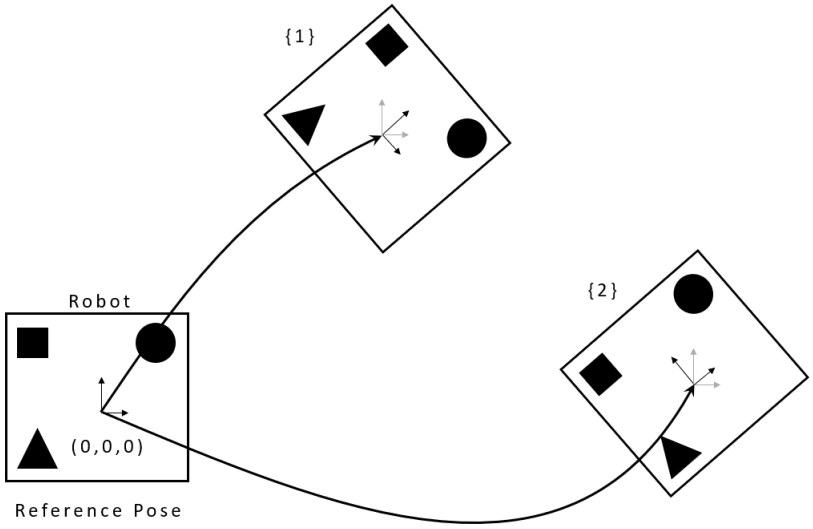

Figure 1: 2D Robot Pose Tracking Scenario. The global origin/reference frame is the location of robot at the program's start. Pose 1 and Pose 2 are 2 different locations of the robot, with two separate translation vectors and rotation matrices relaying their pose to the origin

4.1 Retrieving Template Centroids

In the previous component of the lab, you made yourself familiar to detect object shapes. Here, we extend that method to retrieve the centroid of distinctive shapes in the template(i.e, triangle, square, circle. NOTE: in that order). You can use `regionsprops`() function from Image Processing Toolbox. The skeleton for the code is given to you in

`feducialCentroidDetection.m`. Please read the comments in the code to for the better understanding of the input and return types.
Show it to your TA for a checkout!

4.2 Template Matching

Once you complete the section 4.1, you have the location of the robot and centroid locations of each shape inside the template. Knowing the template centroid location, you can generate a translation vector (x and y coordinates) of the robot at the current timestep with respect to the origin. And knowing the location and direction of the shapes with respect to the template centroid can be used to determine the rotation of the robot. To determine the orientation of the robot, we match the templates (relative to the origin). Since the centroid locations were determined using image processing, there will be some inaccuracies. Thus, we use least-squares estimation to generate an estimation of the rotation matrix that relates the global reference frame to the frame local to the robot. In general,

$$\mathbf{p}_i = \mathbf{p_0} + \mathbf{R} \times \mathbf{p}_{i0} \tag{1}$$

Here $\mathbf{p}_i$ is the centroid location of $i'th$ shape; either the triangle (1), the square (2), or the circle (3) in the current image. $\mathbf{p}$ is the centroid of the template in the current image. $\mathbf{p}_{i0}$ is the location of the template shapes in the original/reference template. R is the rotation matrix representing the change in orientation between the base0.PNG and the template in the current image

$$\mathbf{R} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

To estimate $\mathbf{R}$, we use the least squares solution that you have used in algebra. Except now, we perform the calculation on all 3 shape locations at once to improve the estimation. We do this by creating a matrix of all 3 shape vectors p_{123} that has the location of all 3 shape centroids concatenated side by side

$$\mathbf{R}_\theta{}^1 = (\mathbf{p}_{123} - \mathbf{p}_0)\ \mathbf{p}_{1230}^{-1} \tag{3}$$

Finally the 2D robot pose can be concatenated as $\{\mathbf{p}_0, \theta\}$

NOTE: In code, use `pinv` for the inverse

[1] θ can be extracted from `rotm2eul()` in MATLAB

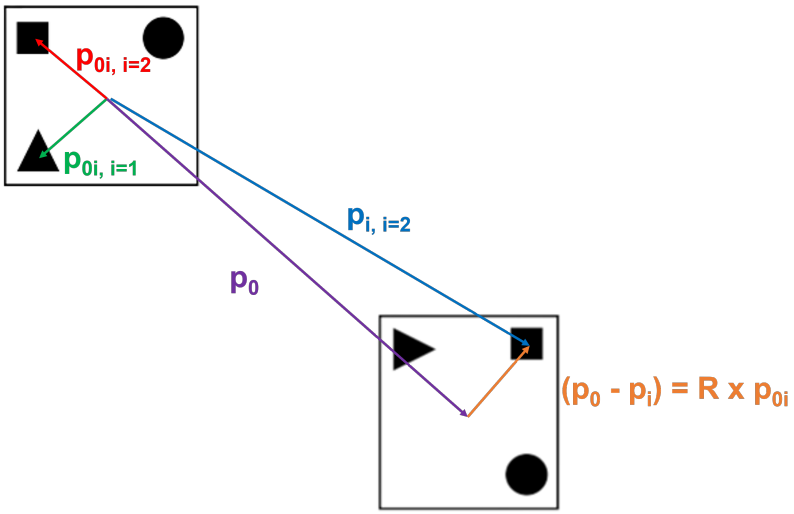

Figure 2: Template matching.

Perform template matching for the static images included in the Lab 5 folder. Treat the center of the template in base0.PNG as the origin/reference frame and be able to detect the translation/rotation of each other image in the folder with respect to the origin. Note, in image processing, the y axis faces down. Which means pixel coordinates get lower as you get higher in the image.

5 Tracking of the Pose

Using your webcam, capture images of the robot/template as you move it around your camera's field of view. Do this by holding an image of the template (on a piece of paper or on a phone/tablet). Use the robot's location in the first snapshot as the global reference frame. You can save this image to your workspace to repeatedly use the same origin. You will demonstrate your ability to track the robot by outputting the pose of the robot (x, y, θ) command window or overlayed onto the figure screen. Additionally, plot the boundary around the shapes, each in a different color. First perform this with still webcam images to make sure your program works, then do this with a video feed for full points. *pose_tracking_main.m* has the skeleton code for you to complete
Show it to your TA for a checkout!

5.1 Extra Credit

The template matching method used in the lab should only work when the template is approximately the same size as it was in the original image. Develop a scale-invariant version of this method that allows the template to be tracked despite changes in the robot's size. This is equivalent to the robot getting further away from the camera. Demonstrate

this to the TA by moving your template far away and very close to the camera such that it substantially changes size from the size in the first snapshot.