1. **Collision Checking.** Please implement a python function isColliding(rect1, rect2). The function returns True if these two rectangles are colliding. Otherwise, return False. rect1 and rect2 are 2-dimensional rectangles in the format of ((centerX, centerY), (width, height), angle). You may use the libraries numpy and cv2.

2. **Optimization**. In this problem you will estimate the transformation between two cameras from corresponding image points. You may only use the python libraries `pickle, numpy, scipy, cv2`, and any built-in libraries and functions.

   A calibration object with 49 points was displayed in 10 different poses to two stationary cameras with a fixed transformation between them. The pickle file `calibration_data.pp` contains a python dictionary with three keys containing data from this experiment. The first key, `objectPoints`, contains a numpy array containing the 3D positions (in millimeters) of the object points given in an arbitrary object frame. The second key `leftCameraData` holds another python dictionary with two keys, `cameraMatrix`, the intrinsic projective camera matrix in units such that object points in millimeters will be taken to image points in pixels, and `imagePoints`, a list of 10 numpy arrays of the two-dimensional image points in pixel units. The third and final key, `rightCameraData`, contains the corresponding data for the right camera. Note that `imagePoints` for the two cameras are already ordered correspondingly. For example, `leftCameraData[imagePoints][3][25]` corresponds to `rightCameraData[imagePoints][3][25]`.

   Suppose we have an estimate T of the homogeneous transformation matrix that takes points from the right camera frame to the left camera frame. Using only `objectPoints` and `leftCameraData`, for each image we can compute the 3D positions of the object points expressed in the left camera frame. For the ith image, denote these 3D positions by $X_1^i, \ldots, X_{49}^i \in \mathbb{R}^3$ for $i = 1, \ldots, 10$. Alternatively, using only `objectPoints`, `rightCameraData`, and the estimate T, for the ith image we can again compute the 3D positions of the object points expressed in the left camera frame, which we denote by $Y_1^i, \ldots, Y_{49}^i \in \mathbb{R}^3$. Note that for a given image index i, the object points should be identical whether we compute them the first way or the second way, i.e., $X_j^i = Y_j^i$ for $j = 1, \ldots, 49$. If our estimate T is accurate, then the root-mean-square error

$$J = \sqrt{\frac{1}{(10)(49)} \sum_{i=1}^{10} \sum_{j=1}^{49} \|X_j^i - Y_j^i\|^2}$$

   will be small.

   (a) Implement a python function that takes in a vector of 6 elements (the first 3 for rotation and the last 3 for translation) that parameterizes the transformation matrix T and returns the root-mean-square error J defined above using the data in `calibration_data.pp`. You may use any parameterization of your choice for the 3 rotation angles.

   (b) Use the function `scipy.optimize.least_squares` to find an estimate T of the homogeneous transformation matrix that minimizes the error J. You may want to modify the function you wrote above before you pass it to the `least_squares` function. This function requires an initial guess, which you may produce any way you wish. Feel free to experiment with the optional arguments of `least_squares` to improve your estimate. In addition to your code, report your final estimate T as a 4-by-4 matrix with translation in millimeters, and report your final value of the error J in millimeters.