Problem Statement

ROUND 2 - Robonaut 2 Tool Manipulation Contest

Prize Distribution

The best 5 performers in this 14-day contest according to system test results will receive the following prizes:

1. place: $4000
2. place: $2750
3. place: $1750
4. place: $1000
5. place: $500

Bonus Opportunity

The winner of this challenge will win an all-expenses paid trip to the 2016 Topcoder Open Finals in Washington DC November 18-21, 2016. Topcoder will coordinate and pay for the travel details as well as assist with a visa appointment (if needed). At the TCO16 finals, the winner will assist Topcoder in reporting on the TCO16 Marathon Competition finals on the Topcoder blog. In the event the winner of the match already has a TCO16 ticket, then the bonus ticket goes to the next highest ranked competitor in this contest, and so-on.

Requirements to Win a Prize

Achieve a score in the top 5, according to system test results. Your system test score needs to be greater than 300000 in order to be eligible for a prize. See the scoring section below.

Within 7 days from the announcement of the challenge winners, submit a complete report at least 2 pages long outlining your final algorithm, explaining the logic behind and steps to its approach, and describing how to install any required libraries and run it. The required content appears in the report section below.

If you place in the top 5 but fail to do any of the above, then you will not receive a prize, and it will be awarded to the contestant with the next best performance who did all of the above.

Background

This match will act as Round 2 to our R2 challenge held earlier this year.

Robonaut 2 ("R2"), a humanoid robot that operates both on Earth and on the International Space Station, commonly used tools. For example, it manages inventory using an RFID reader and fastens bolts with a drill. In order to use a tool, R2 relies on an algorithm to determine a 3D representation of the tool. The algorithm works with the robot's control system and allows R2 to create a plan for grasping objects and completing its tasks.

There exist several algorithms that could be used to determine a 3D representation of the tool. However, the robot employs an older, less capable set of vision sensors, due to its space heritage and having been exposed to high levels of environmental radiation over time. Many existing algorithms assume that the vision data being used is of relatively high resolution, detail, and quality, and such algorithms are not effective when used with the grade of vision data available to R2. As a result, the R2 team needs you to create vision algorithms for determining the 3D representation of different tools that will be effective with noisy, stereo vision data.

Data Description

The following training package can be downloaded here. The package contains the following:

- A string-based mesh file containing a 3D wire-frame model of each tool. The mesh file is in ply format. More information about ply format can be found here
- 88 stereo image pairs of the tools. 176 images in total.
- For each stereo pair the 3D representation of the object in each stereo pair as a 3-element translation vector and 4-element rotational quaternion.

An additional tool included in provisional and system testing, and a few sample images (low resolution) are available here.

A visualization tool is available and can be downloaded here.

Note that you are encouraged to use the 3D model in your solution as the center point of the object is determined by the 3D model.

The 3D position at (0,0,0) is defined as the mid point between the focal points of the left and right cameras. The cameras are looking in the Z-axis direction with the X-axis going horizontal and the Y-axis vertical. The units used for position is in millimetres (mm).

An example stereo image pair rendered with the visualization tool can be seen below. The green points represents the vertices of the 3D model transformed to the ground truth location. The red points represents an example of the output of an algorithm.

Implementation

Your task is to implement trainingModel, trainingPair, doneTraining and testingPair methods, whose signatures are detailed below.

`int[] leftImage` and `int[] rightImage` contains the unsigned 24 bit image data. The data of each pixel is a single number calculated as 2^16 * Red + 2^8 * Green + Blue, where Red, Green and Blue are 8-bit RGB components of this pixel. The size of the image is 1600 by 1200 pixels. Let x be the column and y be the row of a pixel (top-left = 0,0), then the pixel value can be found at index [x+y*1600] of the image arrays.

The first 3 elements of `double[] groundTruth` contain a 3-element translation vector, and the next 4 contain a 4-element Quaternion, which together give the 3D representation of the object in each stereo image pair. Some objects have two ground truths due to object symmetry, those objects will have a `groundTruth` with length 14 (First ground truth position in the first 7 elements and the second ground truth in the last 7 elements).

The `trainingModel` method will be called first and will be fed the string-based ply formatted mesh file containing a 3D model of the tool for that test. The more your algorithm relies on the model files and the fewer training images it uses, the higher your score. See the "Testing and Scoring" section below. Next, your `trainingPair` training method will be called multiple times. You can use this method to train your algorithm on the pairs of stereo images of the tools. If you return 1 from `trainingModel`, your algorithm with not receive any image pairs. Similarly, if your `trainingPair` method returns 1, then no more image data will be passed to your algorithm, and the testing phase will begin. This method call defines the tool that will be used throughout the remainder of the test run.

Once all training images have been supplied, or your `trainingModel` or `trainingPair` method has returned 1 to end the reception of training data, `doneTraining` will be called. This will signal that your solution has concluded receiving training data, and--if desired-- take any action necessary to prepare to receive test data.

Finally, your `testingPair` method will be called for each testing image in the test. The array you return should contain exactly 7 values. Each element in your return should contain the following information:

- `Xe` : estimated X-coordinate of the translation vector
- `Ye` : estimated Y-coordinate of the translation vector
- `Ze` : estimated Z-coordinate of the translation vector
- `qre` : estimated R-element of the Quaternion
- `qie` : estimated I-element of the Quaternion
- `qje` : estimated J-element of the Quaternion
- `qke` : estimated K-element of the Quaternion

Library Functions

The source code of the visualization tool that you can download here contains a class called `Transform`. The class contains several useful methods that you can freely use.

- `double[] transform3Dto2D(double x, double y, double z)`. The `transform3Dto2D` method takes as input a (x,y,z) 3D position and returns an array that contains the 2D pixel positions for the left and right images. Let `R` be the returned array. The pixel coordinate of the given 3D point on the left image will be (R[0], R[1]) and on the right image at (R[2], R[3]). The code is commented and explain each step of the process.
- `double[] rotate(double x, double y, double z, double qr, double qi, double qj, double qk)`. The `rotate` method rotates a given (x,y,z) 3D position with the given Quaternion (qr, qi, qj, qk). The method returns a 3-element array that contains the rotated point in 3D space.

Testing and Scoring

There are 4 example tests, 2 provisional tests and 3 system tests.

The breakdown of example, provisional and system tests can be seen in the table below. The tools used for provisional and system tests may or may not overlap with the example tools, this information is kept hidden.

It is known that the model of the softbox is not 100% correct and will not be used for provisional or system testing. Some of the sides in the model are swapped. However, the softbox remains part of the example tests such that you can use for training purposes.

Test	Tool	Training Pairs	Testing Pairs	Total Pairs
Example 1	Drill	6	14	20
Example 2	EVA Handrail	6	10	16
Example 3	RFID Reader	7	23	30
Example 4	Softbox	7	15	22
Provisional 1	HIDDEN	2	8	10
Provisional 2	HIDDEN	3	20	23
System 1	HIDDEN	3	7	10
System 2	HIDDEN	4	20	24
System 3	HIDDEN	6	12	18

Your algorithm's performance will be quantified by average distance scoring as follows (Please have a look at the source code of the visualizer for the exact implementation details of the scoring code):

- `Xo(i), Yo(i), Zo(i)` : Ground truth (X,Y,Z) coordinates of the i'th model 3D point.
- `Xe(i), Ye(i), Ze(i)` : Estimated (X,Y,Z) coordinates of the i'th model 3D point. Calculated by transforming the 3D model with the translation vector and Quaternion provided by your algorithm.
- `T(i) = sqrt((Xo(i) - Xe(i))^2 + (Yo(i) - Ye(i))^2 + (Zo(i) - Ze(i))^2)` : The positional error for the i'th point.
- `AverageT = Sum of T(i) over all i divided by the number of model points` : Average positional error over all model points.
- `D = Object diameter` : Calculated as the largest distance between any two points in the model.
- `AD = 1 / (1 + (AverageT / (0.1*D)))` : Average distance score, desired to be within 10 percent of the object diameter.
- `ScorePair = 1000000 * AD / (1 + 0.01 * TrainingImagesUsed)` : Score for an image pair.

Your overall score for a single test case is the sum of all ScorePair scores. You can see these scores for example test cases when you make example test submissions. If your solution fails to produce a proper return value, your score for this test case will be 0.

The overall score on a set of test cases is the mean (average) of all ScorePair scores. The match standings displays overall scores on provisional tests for all competitors who have made at least 1 full test submission. The winners are competitors with the highest overall scores on system tests.

Special rules and conditions

- This match is not rated.
- The allowed programming languages are C++, Java, C#, Python and VB.

External data sources

Many publicly available data sets addresses the problem of 6D pose estimation. External data sources may be used as long as it does not violate our acceptable license agreements. The following links might be useful:

- Research by Rigas Kouskouridas. http://www.iis.ee.ic.ac.uk/rkouskou/Research.html
- International Workshop on Recovering 6D Object Pose. ICCV 2015. http://www.iis.ee.ic.ac.uk/ComputerVision/3DPose-2015.html
- The top 2 submissions of round 1 can be downloaded here and may be used as a starting point for your submission.

Image processing hints

This section provides a few hints on standard image processing techniques that can help in solving the problem, these techniques are not required. Now that the penalty of using training images have been significantly reduced, feature matching techniques may become much more useful. Texture features can be extracted from training images and used to find a good match on the testing images. Using the 2D locations of the 3D model points in the training images instead of detecting features with a feature detector might also be useful. Some feature descriptors worth investigating: Histogram of Orientated Gradients (HOG), Local Binary Patterns (LBP), Speeded up robust features (SURF), and many more... If one calculates the normal vector of each triangle in the model, it is possible to determine when a 3D point is facing away from the camera (behind, and can not be seen), which provides a hint on how to determine when to extract features from a model point.

Differences between Round 1 and Round 2

The main differences between Round 1 and Round 2 follows:

- The model of the drill has been updated with a more detailed model which fits better on the images.
- The penalty of using training images has been reduced significantly.
- The scoring function has been updated to average distance scoring.
- The source code of the top submissions of Round 1 is available and can be used as a starting point.
- Win a trip to TCO16 !!

Report

Your report must be at least 2 pages long, contain at least the following sections, and use the section names below.

Contact Information

- First Name
- Last Name
- Topcoder Handle
- Email Address
- Final Code Submission File Name

Approach Used

Please describe your algorithm so that we can understand it even before seeing your code. Use line references to refer to specific portions of your code. This section must contain at least the following:

- Approaches Considered
- Approach Ultimately Chosen
- Steps to Approach Ultimately Chosen, Including References to Specific Lines of Code
- Advantages and Disadvantages of the Approach Chosen
- Comments on Libraries
- Special Guidance Given to Algorithm Based on Training
- Potential Algorithm Improvements

Definition

Class:	RobonautVisionR2
Method:	doneTraining
Parameters:	
Returns:	int
Method signature:	int doneTraining()

Method:	testingPair
Parameters:	int[], int[]
Returns:	double[]
Method signature:	double[] testingPair(int[] leftImage, int[] rightImage)

Method:	trainingModel
Parameters:	String[]
Returns:	int
Method signature:	int trainingModel(String[] plyData)

Method:	trainingPair
Parameters:	int[], int[], double[]
Returns:	int
Method signature:	int trainingPair(int[] leftImage, int[] rightImage, double[] groundTruth)

(be sure your methods are public)

Notes

- The match forum is located here. Please check it regularly because some important clarifications and/or updates may be posted there. You can click "Watch Forum" if you would like to receive automatic notifications about all posted messages to your email.
- Time limit is 10 minutes per training method called, and 90 seconds per testing method call. Total execution all training and testing pairs for a test run is limited to 60 minutes.
- The memory limit is 2048MB.
- There is no explicit code size limit. The implicit source code size limit is around 1 MB (it is not advisable to submit codes of size close to that or larger).
- The compilation time limit is 60 seconds. You can find information about compilers that we use, compilation options and processing server specifications here.

Examples

0)

```
SEED=1
```

1)

```
SEED=2
```

2)

```
SEED=3
```

3)

```
SEED=4
```
