Propeller Hunt 1

My solution of this Marathon Match is based on *random forest* method. The objects are detected using Canny's edge detection filter. The linking of propellers is trivial – using the radius to split the results into several baskets.

Image Processing

With every image, the following procedure is performed (both during training and testing period):

- Canny's edge detection is applied with parameters (`tLow`, `tTop`, `SIGMA`) selected by observing the results for several different values. The result of the edge detection is a vector of 8-connected regions of edge points, called "edge shapes". Each shape stores its list of edge points from which several features are extracted later.
- The 4-connected regions of non-edge point's which do not contain more than `SIZELIMIT` pixels (i. e., small-enough areas surrounded by edge points) are stored as "region shapes". (Region shapes are stored in the same structure as edge shapes and similar features are later extracted.)
- The set of edge shapes and region shapes is reduced according to these rules:
  - Edge shapes with at least `SIZELIMIT` points and edge shapes with at most `SIZELIMITDOWN` points are discarded.
  - The properties `gx` (average of x-coordinates), `gy` (average of y-coordinates), `slope`, and `linearity` for each edge shape and region shape are calculated (the line fit using total least squares is calculated, `slope` is the slope of this line and `linearity` is the ratio of long axis to short axis, i.e., ratio of eigenvalues of the matrix used to calculate total least squares).
  - Edge shapes with `linearity >= LINLIMIT` are discarded.
  - Region shapes with `linearity >= LINLIMITR` are discarded.
  - Each region shape which is too close (euclidean distance of (`gx`, `gy`) less than 10) to some of the remaining edge shapes is discarded.
- In case of training, each remaining shape is tested against `imageGroundTruth` if it actually is a propeller. The features are assigned to the shapes.

Final Answer

At the beginning of the first call of `testingData()`, the random forest is built over the shapes from training period. In `getAnswer()` call, the shapes from the testing period are evaluated on the random forest. Each tree assigns the ratio of the number of propellers in the set where the shapes falls. The results of all the trees are averaged to get the score of the shape.

The testing shapes are then sorted in descending order according to the score. The image rank of each testing shape is calculated as the order of the shape in the sub-list of the shapes from the same image. All the shapes are then sorted again with image-rank as the primary key (ascending) and random forest score as the secondary key (descending).

The first 10000 items from the list are returned as the answer. Before returning, the shapes are classified according to the radius (distance from Saturn) – the range of all radii is divided into N equidistant baskets where N equals to the number of testing images (this is probably not the best choice but I forgot to change it to something else). `ObjectID` of each shape is set to the number of the basket it falls in.

Features

To each shape, the following features are assigned, using the notation from the preceding text (together 9 features, 0-based numbering):

0. shape type – 0 in case of edge shape, 1 in case of region shape;
1. number of pixels in the shape;
2. shape linearity;
3. shape slope;
4. distance from the image border in pixels;
5. orbital radius with respect to Saturn;
6. longitudinal coordinate with respect to Saturn [*not used in final submission*];
7. number of shapes detected in the same image;
8. deviation of the slope of the shape from the slope of the orbital trajectory at given position [*not used in final submission*].

Random Forests Setup

The possible parameters for the random forests are:

Parameter description	**Value in final submission**
Number of trees	Up to 500 (time measurement was used to interrupt the forest creation when approaching the deadline, but probably the interruption did not occur)
Number of random features used to split nodes	3
Stopping criteria	`MINNODE=1, MAXLEVEL=50`
Cost function used to split nodes	$\sqrt[3]{x(1-x)}$

`MINNODE=1` means that the tree will grow up to the nodes of size 1, which means there is no stopping criteria based on node size. `MAXLEVEL` is the maximum depth of the tree. I did not check if this level was ever reached, so maybe this stopping criteria is redundant. I did not optimize these parameters.

Overview of implementation

Lines of code	**Description**
1 to 34	Includes and preprocessor definition of the cost function $G(x)$ used to split nodes
36 to 87	Definition of constants and parameters; the values for these parameters were selected by several trials and observing the results, no special science was used for optimizing them
89 to 103	Time measurement method
105 to 132	General methods like random generator, splitting the text by comma
134 to 433	Translation of the provided classes SimpleMatrix and Transformer from Java to C++
436 to 514	Definition of classes used for storing the shapes and calculating their properties
515 to 528	Method generating a random sample of features
529 to 575	Definition of the `class Item`, used to store items upon which random forest is build

	and assign features
576 to 655	Ugly implementation of comparing methods, used to sort a sample at a specific node by a specific feature, to easily find the best splitting
657 to 749	Definition of the `class Sample`, used to store a sample of items at a specific node when building a tree
750 to 817	Definition of the `class Node`; each tree of the forest is a collection of these nodes; each node stores the id's of the left and right node below it, feature used to split this node, value of that feature used to split this node, level of this node and the number of items for each cluster among the items in this node
818 to 1080	Definition of the `class Tree`; including recursive method `clusterAtNode` used for calculating the answer of a tested item on a particular tree and the recursive method `divideNode` used to build the tree. When splitting a node, the sample is sorted by every tested feature and split by every possible intermediate value. The best splitting is stored and performed at the end. The class contains also some methods for other type of random forest, which were not used in this match.
1081 to 1147	Method for building one tree; i.e., it selects random sample of candidates, creates a tree with one node and call the recursive method `divideNode` described in the previous cell. (And alternative methods for other types of random forest not used in this match.)
1148 to 1176	Methods used in edge detection
1177 to 1324	Edge shape detection method
1325 to 1393	Region shape calculation method
1394 to 1418	Method gluing the image processing commands together in one place; used both in training and testing
1420 to 1593	Unused methods; remainders of the first approach
1596 to 1610	Some global variables declaration
1612 to 1894	Definition of the `class PropellerDetector`: initialization of edge detection filters and random forest clusters `[lines 1618 - 1647]`; processing training image and comparing with ground truth `[lines 1649 - 1669]`; random forest creation `[lines 1766 - 1792]`; processing testing image `[lines 1793 - 1800]`; unnecessary lines from first approach `[lines 1805 - 1814]`; evaluating test shapes on the forest `[lines 1840 - 1852]`; sorting the result `[lines 1854 - 1861]`; determination of ObjectId `[lines 1862 - 1873]`.

Notes

- At the very end of the first phase I found out that better parameters for Canny's edge detection would probably lead to score increasing. But I had only one trial left and did not want to risk. You can check my first submission from the second phase (parameters on lines 40 – 42). It would shift me to the 3rd position in provisional rank of the first phase.
- I understand that my approach is quite week (comparing with the score of the best two submissions). The reason is that I did not use the provided RingSubtractor code at all. It was in Java and I did not have time enough to translate in into C++ (to be able to use my random forest and edge detection code from the previous matches).
- I tried to run also the provided Java code, but due some reason, the RingSubtractor got stuck on the third image from the example test.