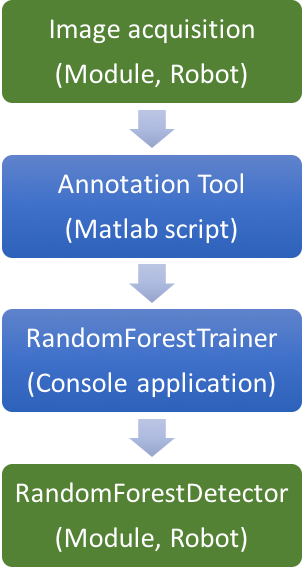
**Documentation for Random Forest Toolset**

Random forest is a supervised ML classifier and needs therefore an annotated training set. Using random forest on images patches requires to acquire several images on the robot, annotate the objects within the images, create the patches for the training set, train the random forest on the image patches and transfer the generated random trees to the robot where the random forest can be used for classifying patches online.

For a faster image acquisition, there exists a module for the robots called “ImageAcquisition” that allows to acquire images automatically. The images are transferred from the robot to a PC for the offline training process. The annotation process and creation of the training set is done with the “Annotation Tool” in matlab. The image patches are then fed to the “RandomForestTrainer” (console application) that trains and tests random trees and provides some statistics for evaluating the results. The “RandomForestDetector” module loads the random trees for classifying.



Images (bmp)

Random trees (xml)

Image patches (bmp)

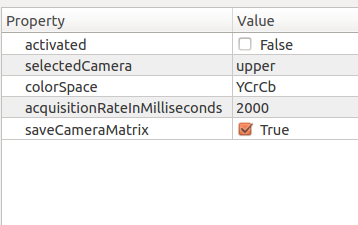
**Image Acquisition Module**

Purpose: Convenient way to acquire images in SimRobot automatically.

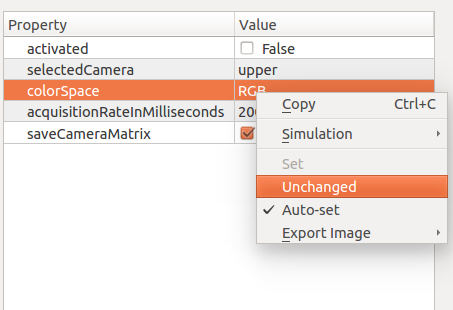
Can get loaded by typing in the console of SimRobot: “vd representation:ImageAcquisition”

Properties:

* Activated True/False
* SelectedCamera upper/lower
* colorSpace YCrCb/RGB
* acquisitionRateInMilliseconds 1…MAX\_UINT
* saveCameraMatrix True/False



The image acquisition gets activated by clicking on the checkbox of the “activated” property. Changes may only get applied if you right click on one of the properties and click on “Unchanged” (strange behaviour, it is what it is…).



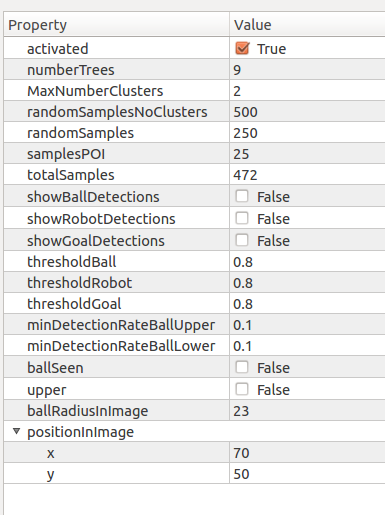
The acquired images are saved as bmp on the robot in “~/Images/”. If “saveCameraMatrix” has been checked for each image the camera matrix is saved in a xml file in the “~/Images/xml/” directory.

To transfer the images/xmls to your PC, secure copy (scp) can get used with the input argument “-r” for recursively copy on the entire Image directory. Don’t forget to delete the images on the robot after the transfer. The entire Image directory can get removed.

**RandomForestDetector Module**

Purpose: Module for classifying image patches with the random forest and providing ball position by clustering and verifying the results.

The representation can get loaded by typing in the console of SimRobot: “vd representation:RandomForestDetector”



RandomForestDetector is activated by default. Changes in the representation will not get saved after closing SimRobot! The default values can be set in the config file: “~/Config/Locations/Default/RandomForestDetectorProvider.cfg”

Properties:

* activated True/False
* numberTrees size of the random forest
* maxNumberClusters max. number of clusters for seeking the ball
* randomSamplesNoClusters number of random samples/patches if no cluster centroids are found in the last iteration.
* randomSamples number of random samples/patches if cluster centroids are found in the last iteration.
* samplesPOI random samples around point of interests (cluster centroids, last ball center)
* totalSamples unchangeable, provides number of total samples
* show[Ball/Robot/Goal]Detections enables/disables colouring classified patches of corresponding class in the image
* threshold[Ball/Robot/Goal] classification threshold. Samples with lower probabilities will get assigned to the background class
* minDetectionRateBall[Upper/Low.] Percentage of classified ball samples vs. all samples around cluster region needed for ball detection.

Note: To see the patches and other drawings in SimRobot you need to add the following lines to “~/Config/Scenes/ScriptRemoteRobot.con”:

vid rawUpper module:RandomForestDetector:upper

vid raw module:RandomForestDetector:lower

**Annotation Tool**

Purpose: Tool for annotating objects in acquired images and generating patches for the training set.

The tool consists of three matlab files. The AnnotationTool.m file contains three code sections:

* Initializing
* Start or resume annotating
* Create set of patches for training

The initializing section **MUST** be run at the beginning and every time settings in the section are getting changed.

To start the annotation process or to resume, the second section can get executed.

The third section will create the patches for the training set.

The initializing section contains several settings:

* **imageZoom:** Zoom of the images during annotation. Different display resolution may need a different zoom setting for easy annotation.
* **convertFromYCrCb:** If the images are in the YCrCb format (default) they are converted to RGB if set to true during annotation. Note: The patches will not be converted.
* **pathDataSet:** Relative path to the acquired images. They should be located in a subdirectory of the output path.
* **pathOutput:** Path where the xml files, and patches are saved (in subdirectories).
* **annotationFile:** Name of the xml file that contains the annotation of the images. The file will be saved in the xml subdirectory of the output path.
* **classColors:** color settings for the bounding boxes during annotation (does not need to be changed)
* **classLabels:** names of the classes. Starts from index 0 up to 9.
* **defaultClass:** default index for annotation. Index 1 corresponds to the second element in the classLabel vector.
* **startWhereLeft:** If set to true, the annotation process will resume at the last image after closing the window.
* **trainingSetForClass:** Vector which denotes for which classes a training set should be created.
* **patchSize:** Width and height of the patches in pixel
* overlap: Number of pixel outside the annotation box which will be considered for creating patches
* **numPatches:** Number of patches that are extracted randomly of each annotation

**Start annotating:**

If you run this section the first image for annotation gets loaded. Objects within the image can get annotated by drawing bounding boxes around the object with the cursor.

If there is only one object to annotate, the next image can get loaded by pressing space. The annotated object will get assigned to the default class set in the initialization section.

If there are multiple objects within the image to annotate, the drawn bounding boxes are getting assigned to the corresponding class by pressing the class number on the keyboard (0…9). Please note that the negative/background class is assigned to class 0. After assigning the class, a next bounding box can be drawn. If you want to finish the current image and continue with the next one, press first escape to close the drawing mode and press then space to load the next image.

The annotations of the current image are saved after clicking space. If you want to pause the process you can close the window. An error will be printed in the console: Figure closes. To continue, make sure “startWhereLeft” is set to true and the settings are saved. You can run the section for annotation again to continue.

If you want to delete an annotation in an image, close the drawing mode by pressing escape and press backspace to delete the boxes.

**Create Set of Patches for Training:**

Running this section will create the patches and xml files for the training set automatically and save them in the corresponding folders. The xml files contain paths to the patches and additional information. The xml files are needed for the random forest trainer.

**Random Forest Trainer**

After the training set for the random forest has been created, it can be fed to the random forest trainer. The RandomForestTrainer is a console application and accepts several input arguments (Some of them are required). If an input argument requires several strings, they are separated with the colon “:” character.

Arguments:

-h Displays usage instructions.

-i Input (required): At least two from the annotation tool created xml files separated by “:”.

-o Output(required): Paths for saving the trees as xml separated by “:”. Number of output trees is implicitly set by number of output files.

-d Maximum depth of output trees (default: 10).

-l Minimum number of samples required to be at a leaf node (default: 10).

-s Minimum number of samples required to split an internal node (default: 20).

-r Number of random tests (randomly generated split functions) per node (default:1000).

-p Maximum number of patches/samples for each class for training (default: lowest number of patches of a class)

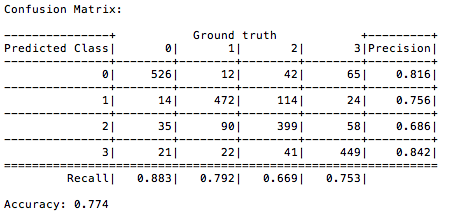
-v verbose mode (without additional string). Provides additional output.

Example:

./RandomForestTrainer –d 5 –s 20 –l 10 –r 1000 –p 5000 –i ball.xml:background.xml –o tree0.xml:tree1.xml -v

The training of the random trees can take some time. After training, a confusion matrix is provided in the console by testing the random forest with the test set.

Example:



Note: All the paths to the image patches in the xml files are seen relative to the working directory of the console! If you run the application from a wrong directory you will get an error.

The generated random trees can now be copied in the Config folder. The paths to the trees must be stored in “~/Config/Locations/Default/RandomForestDetectorProvider.cfg”.