This document describes how to use the object_recognition stack provided by team homer@UniKoblenz.

Required packages:
- *object_recognition* stack with the packages *or_libs*, *or_msgs*, *or_nodes*
- *robbie_architecture* (contains some core libraries from our framework)
- *String, VectorGraphics* (two other libraries from our framework)

The package *or_nodes* provides two ros nodes:
- *obj_learn* for object learning and
- *obj_rec* for object recognition

The package provides two launchfiles:
- *obj_learn.launch*
- *obj_rec.launch*

Each of these files starts the corresponding node and loads some parameters onto the ROS parameter server by reading the file */config/params.yaml* contained in this package.

So far, the file *params.yaml* contains 3 parameters:
- */OrNodes/sConfigFile*: Path to the legacy configuration file relative to the package path (default value is /config/custom.xml).
- */OrNodes/sProfile*: Profile inside the configuration file specified in */OrNodes/sConfigFile* to be loaded on startup (default value is *drinks*).
- /OrNodes/sInputImageTopic: Image input topic for object recognition (default value is */camera/rgb/image_color*, i.e. the default *openni* RGB topic of the Kinect sensor).

Some hints regarding the legacy configuration files:
The *config* folder contains the two files *default.xml* and *custom.xml*. The file *default.xml* is automatically loaded on startup. It contains all parameters necessary for object learning and object recognition, encapsulated in the profile *default*. Normally, you don't need to change values in this file. To specify custom values, please use the file *custom.xml*. Currently, the profile *drinks* is defined in *custom.xml* that overrides the default value of *ObjectRecognition/sLoadObject*, namely the object files that should be loaded on startup (only the loaded objects can be recognized by the system). When loading the configuration files, a *merged.xml* is generated by the config loader, that contains all available profiles.

Interfacing the obj_learn node

The obj_learn node subscribes to the following topics (the messages are added in brackets):
- /or/learn_commands (or_msgs::OrLearnCommand)
- the input image topic specified by the parameter */OrNodes/sInputImageTopic* (sensor_msgs::Image), default value: /camera/rgb/image_color

The following topics are advertised (messages in brackets):
- /or/obj_learn_primary (or_msgs::OrImage)
- /or/debug_image (or_msgs::OrImage)
- /or/learning_status (or_msgs::OrLearningStatus)

To learn an object one has to create an interface with the following functionality (an example screenshot is shown below):

1. Select image source (right now this is always the value specified in by the parameter */OrNodes/sInputImageTopic* in the config/params.yaml file)
2. Grab background image (this image will be subtracted from the object image (foreground)); this should publish the message *or_msgs::OrLearnCommand* on topic */or/learn_commands* with the *command* field set to *ORLearningModule::GrabBackgroundImage*. The next incoming image on the image source topic will be processed as background image. The node's response is an image on the topic */or/debug_image,* containing a color and grayscale background image. Further, a color image is published on */or/obj_learn_primary*.
3. Grab foreground image; this should publish the message *or_msgs::OrLearnCommand* on topic */or/learn_commands* with the *command* field set to *ORLearningModule::GrabForegroundImage*. The next incoming image on the image source topic will be processed as foreground image. The node's response is an image on the topic */or/debug_image*, containing a color and grayscale background image. Further, a color image is published on */or/obj_learn_primary*. This color image is superimposed with the outline of the object obtained from background subtraction.
4. Now different thresholds for the object outline can be adjusted. Each change will result in an image published on */or/obj_learn_primary* with the superimposed outline with new threshold values. The thresholds should be published on */or/learn_commands,* the *command* field of the *or_msgs::OrLearnCommand* should be *ORLearningModule::SetDifferenceThreshold*, *ORLearningModule::SetOpenRadius*, and *ORLearningModule::SetBorderSize,* respectively. The threshold value itself should be put into the *int_value* field of the message.
5. To add the obtained object view to the object file, the image has to be saved. To do this a *or_msgs::OrLearnCommand* with the *command* field *ORLearningModule::SaveImage* should be published. Optionally, the *string_value* field can be assigned an image name. The node publishes an *or_msgs::OrLearningStatus* on */or/learning_status* containing all image names saved so far. An image can be removed again by an *or_msgs::OrLearnCommand* with the command *ORLearningModule::DeleteImage*. The *int_value* field has to contain the index of the image that has to be deleted. The index is chosen according to the index of the image in the *or_msgs::OrLearningStatus* message.
6. Now different views of the same object should be added by repeating steps 3 to 5. In our experience obtaining about 12 to 15 views is enough to recognize an object reliably.
7. After obtaining enough object views, the object file has to be saved. This is done by publishing *or_msgs::OrLearnCommand* with the command *ORLearningModule::SetObjectType*, where the type "object" has to be provided in the *string_value* field (in our framework a second type "face" exists that can be learned with the same message). Then a second *or_msgs::OrLearnCommand* with the command *ORLearningModule::SaveObject* should be published, where the *string_value* field contains

the file name.

Other commands:

- instead of grabbing background and foreground images, these images can be loaded from disk with the *or_msgs::OrLearnCommand* message and the commands *ORLearningModule::LoadBackgroundImage* and *ORLearningModule::LoadForegroundImage*, respectively.
- Some thresholds can result in various segments for one object. The user can choose between handling only the largest segment and all segments by sending an *or_msgs::OrLearnCommand* message with the command *ORLearningModule::SetIsolateLargestSegment*. Note the in the current state the *string_value* field has to be set to the string "true" or "false" for this command.
- *or_msgs::OrLearnCommand* and the command *ORLearningModule::LoadObject* loads an object file. The file can be modified by adding or removing some of the images.

An example from our framework (right now not compatible with ROS, but ROS compatibility is planned for the future) is shown on the next page. The step by step learning in this example is as follows:

1. Lower right corner: Select image source (000 Top Camera)
2. Click "Grab background"
3. Click "Grab foreground" (before clicking, put the object in front of the camera)
4. Adjust thresholds (using the three sliders, usually we keep the defaults)
5. Enter image name and click "Add Image". The image appears in the image list (on the right side). The image list is updated by the *or_msgs::OrLearningStatus* message.
6. By clicking an image in the list it will be displayed and can be removed by clicking "Remove Image". By clicking "Reset" all images are removed.
7. Repeat steps 3 to 5 (or 6) to obtain various object views.
8. Enter the object file name and select the object's type, then cklick "Save".

Other commands:

- Click "Load background" and "Load foreground" to load instead of grab input images
- Select checkbox "Single segment" true or false (in our case always true)
- Clicking "Load Object" loads an object file for inspection and modification.

The big white spaces on the screenshot are used for image display (left: Object Learning primary, top right: Object Learning Secondary (not used anymore)).

9 Jan 22:05 Viktor Seib

| Robot Control | Map | System | Robotic Arm | 3D View | Images | Object Recognition | Object Learning | Test Tab | Gesture Recognition | Gesture Learning |

400 Object Learning (Primary)

401 Object Learning (Secondary)

Name: _____ Object ▵ 💾 Save

#	Name

Remove Image   Reset   Load Object

Image name: _____   Add Image

Background deletion threshold   25

Mask open radius   15

Additional border   5

000 Top Camera   Grab background   Load background

☑ Single segment   Grab foreground   Load foreground

🔴 Robot State: Waiting for start

⛔ Emergency Stop   Full Screen (F11)

Interfacing the obj_rec node

The obj_rec node subscribes to the following topics (the messages are added in brackets):
- /or/extract (or_msgs::ExtractKeyPoints)
- /or/match_result (or_msgs::OrMatchResult)
- /or/commands (or_msgs::OrCommand)
- the input image topic specified by the parameter */OrNodes/sInputImageTopic* (sensor_msgs::Image), default value: /camera/rgb/image_color

The following topics are advertised (messages in brackets):
- /or/extract (or_msgs::ExtractKeyPoints)
- /or/match_result (or_msgs::OrMatchResult)
- /or/debug_image (or_msgs::OrImage)
- /or/obj_names (or_msgs::OrObjectNames)

To recognize objects one has to create an interface with the following functionality (an example screenshot is shown below):

1. Select image source (right now this is always the value specified in by the parameter */OrNodes/sInputImageTopic* in the *config/params.yaml* file)
2. Load object files for recognition (note that you can specify objects that are automatically loaded on startup). To manually load an object, an *or_msgs::OrCommand* message with the command field set to *ORControlModule::LoadObject* needs to be published on the topic */or/commands*. The *string_value* field of the message has to contain the full path and file name that has to be loaded.
3. To grab an image for recognition, publish an *or_msgs::OrCommand* message on the */or/commands* with the command field set to *ORControlModule::GrabSingleImage*. The next incoming image on the image source topic will be processed as input image for object recognition. The node's response is a message on the topic */or/match_result*, containing the input image with superimposed bounding boxes of the recognized objects, the found key points and key point matches to the loaded object files.
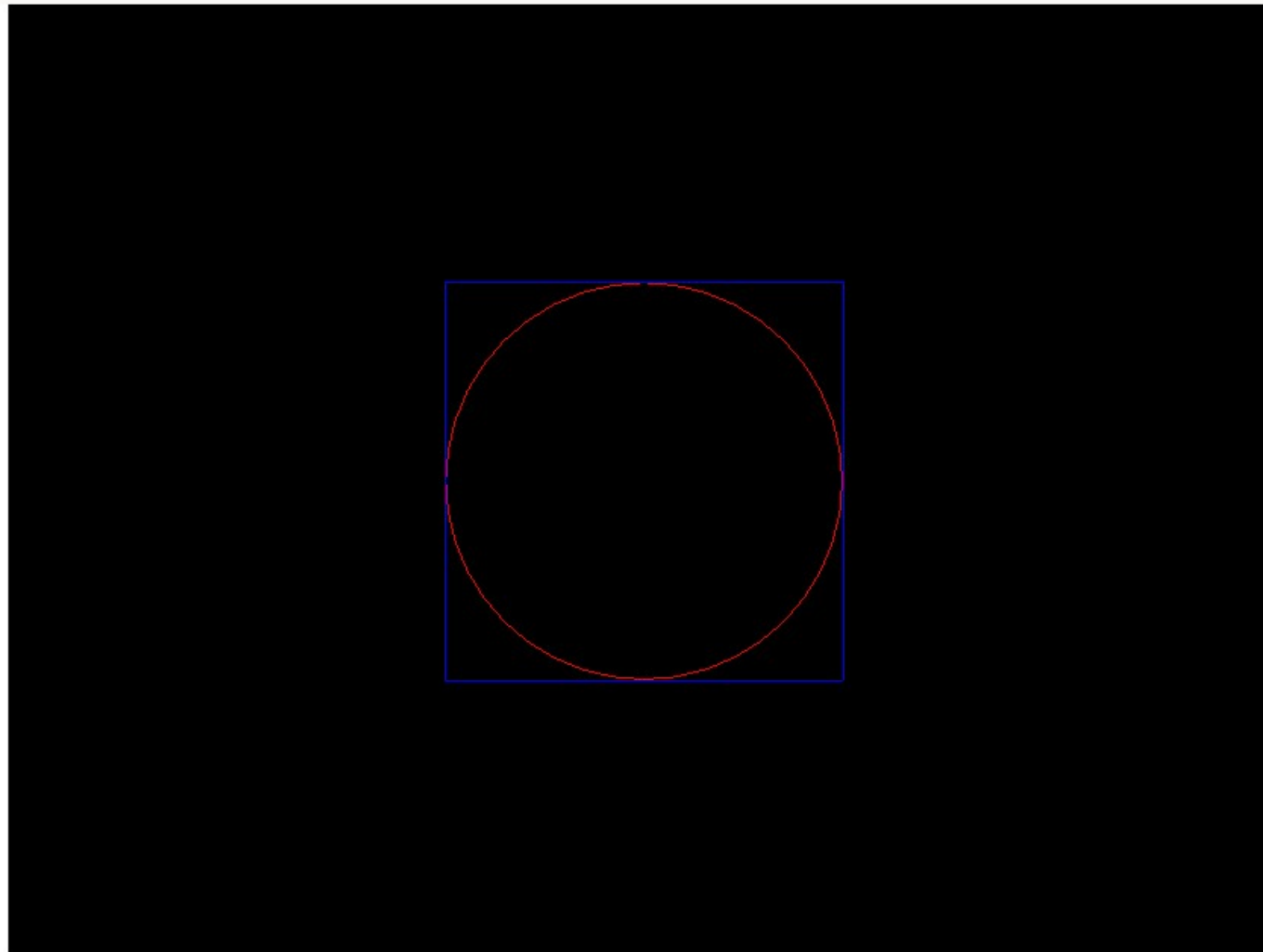
Other commands:
- instead of grabbing an image, an image can be loaded from disk with the *or_msgs::OrCommand* message and the command *ORControlModule::LoadSingleImage*
- instead of grabbing a single image, continuous recognition on the input image topic can be started with the *or_msgs::OrCommand* message and the command *ORControlModule::StartRecognitionLoop*. Usually images will arrive faster than they are processed. The maximum number of images in the queue is determined by the *ObjectRecognition/iMaxImagesInPipeline* parameter in the *default.xml* configuration file. An *or_msgs::OrCommand* message with the command *ORControlModule::StopRecognitionLoop* will stop the recognition loop (images already in the queue will be processed).
- Loaded object files can be unloaded with the *or_msgs::OrCommand* message and the command *ORControlModule::UnloadObject*. The *string_value* field of the message has to containe the object's name.

An example from our framework (right now not compatible with ROS, but ROS compatibility is planned for the future) is shown on the next page. The step by step recognition in this example is as follows:

1. Lower left corner: Select image source (000 Top Camera)
2. Load object files (lower right corner, button "Load")
3. Grab an image for recognition (lower left, button "Grab Image")
4. The result appears on the black area. For debugging purposes, different visualisation options are provided on the right part of the interface (visualizing different information contained in the resulting *or_msgs::OrMatchResult* message).

Other commands:

– The button "Load Image" allows to load an image for recognition
– The buttons "Start Recognition Loop" and "Stop Recognition Loop" start and stop continuous recognition on the input image topic, respectively.
– Clicking "Delete" in the lower right corner deletes selected object files from the object file list