University of Genoa

Software Architecture project

# Deep Learning for Object Recognition

*Garello Luca, Guelfi Serena*

May 9, 2018

# 1 Objective

The objective of this assignment is to integrate methods for object recognition and representation in a table-top scenario with a geometric objects representation using Deep Learning. The project focuses on the integration of those two approaches and the representation of the generated beliefs.

# 2 Accomplishment

- Creation of a ROS node for Tensorflow
- Change of the reference frame for Tensorflow's coordinates
- Creation of a node to merge data from Pit and Tensorflow
- Creation of a ROS node to integrate slower information provided by Pit
- Manipulation of the ontology to save information
- Creation of messages to share information between nodes

# 3 Software architecture

The architecture has been built with a modular structure, in this way it is easy to access, offering the possibility of future developments and customizations.

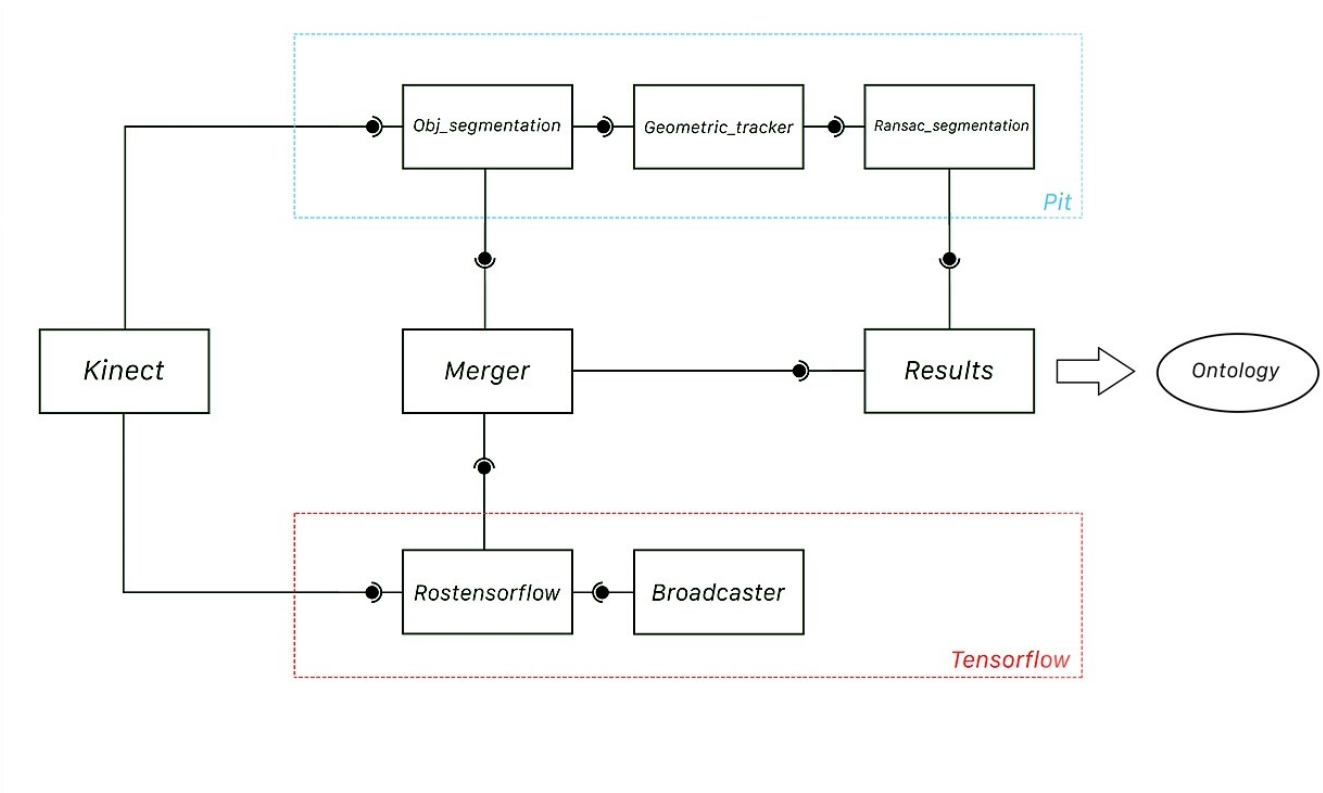


Figure 1: architecture

The architecture is divided into two main structures:

- Pit
- Tensorflow

These two structures are merged together by the third branch in order to build a single software for object detection composed by them.

## 3.1 Ros-tensorflow node

TensorFlow is an open-source software library for data-flow programming across a range of tasks. It Is necessary to convert It into a ROS node in order to communicate with the Baxter and the other software. The node is based on the original code of the software and has been transformed into a publisher-subscriber node in order to receive rgb information from the Kinect and return information about the detected objects. It subscribes to $/cameraB/rgb/imagerectcolor$ topic receiving an rgb image. The neural network processes the image that is returned with bounding boxes around the recognized objects with a label describing the corresponding name. The internal function that draws the bounding boxes has been substituted with a new one that takes the four coordinates of the corners of the bounding boxes and computes its center estimating the mid value and returning it. The dataset is provided for the offline computation in order not to add delay to the node. It Is otherwise possible to modify the code in order to download the dataset online.

The coordinates of the recognized objects are expressed in pixels by Tensorflow while Pitt returns them in meters, for this reason it is necessary to develop a conversion from pixels to meters. This conversion is provided by Tensorflow node simply multiplying the pixel coordinates by a coefficient equal to:

$$\frac{DimensioninMeters}{CameraResolution} = Conversionfactor \tag{1}$$

The resolution of the image is automatically detected.

```
xi=np.size(image_np,0) -> 480
yi=np.size(image_np,1) -> 640
```

The dimension in meters of the image has been manually measured (our case was 0.85 x 1.15). The conversion factor is obtained dividing the two values, in this case is 0.00178. After the conversion the coordinates are multiplied by the transformation matrix placed in the top left of the table (see next subsection). All the information are written at the end on a message 'TensorOutput' and published on the $/TensorOutput$ topic.

## 3.2 Change of reference frame

Tensorflow returns coordinates related to a frame positioned in the top left corner of the acquired image. Pit refers coordinates to the WORLD frame positioned in the body of the Baxter robot.

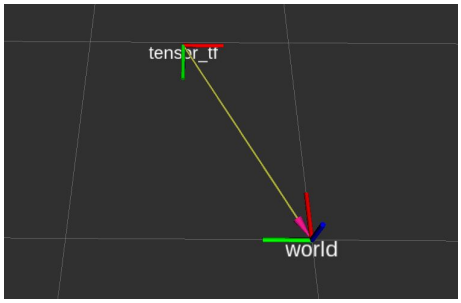


Figure 2: Frame 'Tensor' with respect to 'World' frame

The transformation requires some not trivial configuration steps: initially the frame has been created using 'AR track alvar'. Some markers have been positioned on the table that are detected by the software as a new frame and returns the transformation matrix between the new frame and

the world frame. Markers are positioned looking at the image on the Kinect using RVIZ. However after some tests this method wasn't used because it was difficult for the software to detect a frame in the corner of the image so it wasn't accurate enough. For this reason the frame has been created starting from the 'world' frame and rotating and translating it in order to obtain the frame used by Tensorflow. This operation has been done supposing that the camera cannot be moved and that the new frame is at the same z-coordinate of the WORLD frame. The following steps have been followed for the creation of the frame:

- translation of the frame to the corner of the image

```
$rosrun tf static_transform_publisher 0.97 0.5 0 0 0 0 1  world link1 100
```

- rotation of the frame of -90 degrees around z

```
$rosrun tf static_transform_publisher 0 0 0 1 0 0 0  link1 link2 100
```

- rotation around x axis of 180 degrees

```
$rosrun tf static_transform_publisher 0 0 0 0 0 -0.707 0.707  link2 tensor_tf 100
```

- read the transformation from terminal

```
$rosrun tf tf_echo /world /tensor_tf
```

After the creation of the frame a broadcaster node has been created in order to send the transformation matrix to the Rostensorflow node. The Rostensorflow node works as listener for the frame and uses the transformation matrix to change the coordinates of points. The transformation matrix is manipulated using the tf library.

The transformation of points is performed applying the theoretical rule of change of coordinate of a point :

$$P^a = T_b^a * P^b \tag{2}$$

In this way it is possible to obtain the coordinates of the point by multiplying the transformation matrix by the coordinates of the first frame.

## 3.3 Merger node

The merger node compares information coming from two different softwares that return different information about the same objects at different rates. Pitt returns information at two different rates: The first one is comparable with tensor rate and keeps information related to the position of the object coming from the tracker. The second one is slower because it keeps information related to the shape of the object processed by the Ransac Segmentation which introduces a considerable delay. This information will be successively added by another node in such a way to not delay the flux of information. The Merger node compares the information coming from Pit and Tensorflow in order to match the recognized objects and their information. The comparison is computed for X and Y coordinates of the objects, the two pieces of information are merged every time that their difference is below a certain threshold defined by the user. After some tests we found that the recognition based on the point cloud (Pit) is more reliable than the one provided by Tensorflow, in many cases Pit is able to recognize more objects with a more accurate geometric positioning. For this reason if the information coming from Pit and Tensorflow match We preserve the spatial coordinates given from Pit.

The node subscribes to:

- /TensorOutput topic that receives the TensorOutput message from the Tensorflow node about objects detected by Tensorflow. It contains information related to the 2d position of the object and its label.
- */geometrictracker/trackedCluster* topic that receives a 'ClastersOutput' message from the geometric-tracker node that contains fastest information by pit. This information is related to the 3d position of the objects and its ID.

After the comparison, information is written on Result message and published on the */result* topic

### 3.4 Results node

The results node is the node that merges information from merger node and the slower information about the shape coming from Pit. This kind of information is slower because the RANSAC segmentation compares the point cloud with four primitive shapes having different dimensions and chooses which one of them best fits the object. It subscribes to:

- */results* topic that receives the Result message from the merger node about faster information from pit and Tensorflow
- */ransacsegmentation/trackedShapes* topic that receives a 'TrackedClusters' message from the Ransac-segmentation node that contains slowest information by pit.

Pit contains a tracker in its architecture so each object is characterized by an irrefutable ID number and in this way it is possible to compare information of every object referring to the ID number.

All the information is saved on an ontology that groups the knowledge about all the objects on the table. To interact with the ontology the python library 'armor-py-api' has been used. In order to properly manipulate the ontology new functions have been added to the library.

The code keeps memory of information about all the objects detected and saves it in the ontology only if data are changed. This implementation has been done in order to avoid the use of queries that add delays. Because the library function 'replace()' actually uses queries, it is advisable to reduce its use as much as possible. For this reason a threshold has been set on the difference between the new value and the old one in order to recognize only significant changes of the parameters. To keep in memory the information a class 'Object' has been created , It is written in the file 'objects.py' that contains all the properties that an object can have and that the ontology needs in order to understand the shape.

Information related to the coordinates is substituted with the estimated coordinates returned by the segmentation because is supposed to be more accurate coming from the centroid of the chosen shape.

Also coefficients returned by pit have been added, they are represented by a vector of numbers that assumes a different significance for each shape. All this information is reported in the pit report file.

Information about detected objects is saved on the ontology every time the node receives a message from Pit.

All the nodes have a rospy.spin() in the architecture without the rospy.rate() because the aim is to have the maximum speed for the entire architecture.

## 4 Limitations of the system

The system has some important limitations that are underlined in the previous explanation of the software but are summed up in this section.

- The camera is supposed to be still.
- Tensorflow does a two-dimensional analysis of pictures, if It can't assign a label to an object it will be ignored.
- Pit does a three-dimensional analysis of pictures, It is more accurate than Tensorflow .
- Errors in recognition are due to Pit, the positioning of the frame and the computation of tensor coordinates
- Objects detectable are very few because the two softwares are able to detect different kinds of objects that do not match.
- -> Only the situation in which Pit has detected objects without considering Tensorflow has been considered, because in this way it is possible to assign an id to the object. This because Pit, differently from Tensorflow, contains a tracker in its architecture that assigns an irrefutable id to each detected object.
- the software saves on the ontology all the objects detected during the execution of the program; it is possible to modify the code in order to delete objects that are no more detected.

# 5 How to run

To run this program are required:

- Tensorflow (follow the tutorial, downloading models:
  https://pythonprogramming.net/introduction-use-tensorflow-object-detection-api-tutorial/)
- Ros Kinetic (see http://wiki.ros.org)
- OpenCV (see http://opencv.org/orhttps://github.com/opencv/opencv)
- cv-bridge, and camera driver (for example, cv-camera)

  ```
  $sudo apt-get install ros-indigo-cv-bridge ros-indigo-cv-camera
  ```

- rosjava (see: http://wiki.ros.org/rosjava)
- armor, amor and amor-msgs
  see https://github.com/EmaroLab/multi_ontology_reference
  see https://github.com/EmaroLab/armor
  see https://github.com/EmaroLab/armor_msgs
- armor-py-api (see: https://github.com/EmaroLab/armor_py_api)
- pitt-object table segmentation
  see https://github.com/EmaroLab/pitt_object_table_segmentation
  see https://github.com/EmaroLab/pitt_geometric_tracking
  see https://github.com/EmaroLab/pitt_msgs
- download neural network and put it in the folder

It is necessary to copy the folder of the models $object-detection$ contained in Tensorflow into the folder $/tensorflow-node/src$.
All the downloaded components have to be -> in the same folder in which the $object-recognition$ folder will be saved.

These are the steps to launch the program:

- launch Baxter environment
- launch Tensorflow launcher

  ```
  $roslaunch launch/launch_tensor.launch
  ```

- launch pit launcher

  ```
  $roslaunch launch/table_segmentation.launch
  ```

Performances can be increased disabling the output on screen from Pit node.
A launcher file is provided for the entire architecture.

```
$roslaunch launch/launch_project.launch
```

# 6 Future development

Here are described some possible future developments for this architecture:

- Training for Tensorflow to recognize objects known to pit and vice versa
- Add the graphic viewer for pit and also for Tensorflow. In this project have been removed in order to make the software faster.
- Improve positioning of the frame and study a more accurate way for the coordinates transformation.
- Think about the possibility of using other software for object recognition in order to improve this architecture.

# 7 Bibliography


- Primitive shapes segmentation from points cloud
  https://github.com/EmaroLab/primitive_identification_tracking_tagging/blob/master/Doc/PITTreport.pdf
- commands for armor
  https://github.com/EmaroLab/armor/blob/master/commands.md
- A Software Architecture for Object Perception and Semantic Representation
  https://pdfs.semanticscholar.org/883f/69658b76c4d30d233ca2e8e63263b05495c6.pdf