Online Semantic Segmentation and Manipulation of Objects in Task Intelligence for Service Robots

Adrian Llopart[1], Ole Ravn[1], Nils A. Andersen[1] and Jong-Hwan Kim[2], *Fellow, IEEE*

***Abstract*— Task Intelligence is the capacity of a robot to learn, reason and execute specific behaviours based on its environment. In this paper, the Task Intelligence problem formulated by the Robot Intelligence and Technology Laboratory at KAIST is researched further: specifically the proposed contribution is a brand new perceptual pipeline in which the recognition, detection, segmentation and grasping of objects is achieved assuming no prior knowledge of the environments arrangement nor the objects appearance. A Convolutional Neural Net (CNN) is used to detect, recognize and semantically label those objects that need to be interacted with. 3D point clouds, corresponding to the objects model, are extracted after several segmentation procedures and registered over time. Dimensional and positional information of the object is acquired. Additional grasping pose data is calculated. All of the collected knowledge is parsed so that the Task Intelligence system is able to deal with previously unknown objects in dynamic environments. This system is formed by an Episodic Memory (Deep-ART), an action sequence generator (FF-planner) and a trajectory warping module for pre-learnt behaviours. The proposed approach has been tested using the *Webots* simulator.**

Index Terms - Task Intelligence, semantic segmentation, CNN, Episodic Memory, Deep-ART, humanoid robot

I. INTRODUCTION

The major difference between industrial and service/domestic robotics is the ability of the latter to correctly determine which tasks are necessary to complete a certain goal and in what sequence, depending on the situational context. To accomplish this, robots must be able to memorize task sequences and retrieve them when required. However, when a task sequence is needed, but has not been previously learned, a deterministic search method is employed to infer the best actions or behaviours to achieve its goal, thus creating a new sequence altogether. The results are then stored in memory for later use. Finally, these actions must also be adaptive, meaning that they must be able to be altered depending on the object's position and robot's state.

Previous papers that deal with Task Intelligence rely on the robot having all the information of the objects location and orientation since the very start ([1], [2], [3]); otherwise, a CNN [4] is employed to find the objects positions. Nevertheless, the general dimensions of the object and the grasping poses are still manually predefined. This CNN is trained over the images of the specific objects dealt with in the experiments so it cannot be generalized to other objects with different forms, colours and sizes.

This paper presents a novel perceptual pipeline that allows a robot to update online the known, or lack of, data of the objects found in its environment. The information is stored in memory and is used by the modules in the Task Intelligence system to achieve a certain goal. This way, the robot can deal with unknown or highly dynamic environments instead of being confined to a predetermined static scenario.

The paper is organized as follows: Section II describes the different elements pre-required for the Task Intelligence system. The proposed perception pipeline is shown in Section III. Section IV details the experimental setup and results. The concluding remarks and future work are found in Section V and VI, respectively.

II. TASK INTELLIGENCE SYSTEMS

In this section, a summary of the modules that form the Task Intelligence system is described. These are the Episodic Memory (Deep-ART), the task generator (FF-planner) and a trajectory warping algorithm.

A. Episodic Memory: Deep-ART

The Episodic Memory models the long-term memory of humans: a robot can memorize task sequences based on experiences which are later retrieved depending on partial input cues. The unsupervised neural network Deep-ART, proposed by *Park et. al.* [1] will be used (Fig. 1). It is based on Fusion-ART [5] and is able to learn and retrieve episodes without errors, as opposed to other approaches ([6], [7]).

The episodes are encoded by categorizing events or task sequences defined as a verb clause $\{v$ (action), o_1 (object), γ (geometrical relation), o_2 (object)$\}$.

A.1. Event encoding

The attribute layer (${}^nF_1^k$, *n* and *k* are the number of inputs and attributes) of Deep-ART contains detailed features of the input which are sent to the input field (F_2) through the input channels (${}^nI^k$). ${}^1F_1^1$ receives always an action *v*. For $n \neq 1$, ${}^nF_1^1$ takes a preposition and ${}^nF_1^k$ takes an object class o_n.

TABLE I

DEEP-ART ENCODING OF SUBTASK 6 IN TABLE II

POUR	*Wine_bottle*	in	*Wine_glass*
v	o_1	γ	o_2
${}^1F_1^1$	${}^2F_1^2$	${}^3F_1^1$	${}^3F_1^2$

*This work was not supported by any organization

[1] The authors are with the Department of Electrical Engineering, Technical University of Denmark, Elektrovej 326, 2800 Lyngby, Denmark. (adllo, or, naa)@elektro.dtu.dk

[2]The authors are with the School of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejon 305-701, South Korea (johkim)@rit.kaist.ac.kr

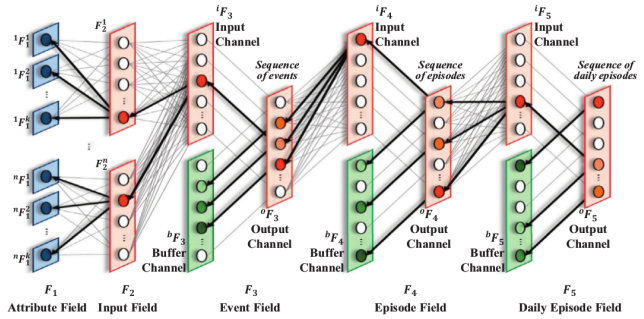

Fig. 1. Deep-ART architecture

An encoding example is shown in Table I.
The input field (F_2) is used to encode the events themselves and works similarly as to aforementioned encoding procedure. The activation of the F_2 nodes is achieved by the choice function:

$$ {}^n_jT = \sum_{l=1}^{k} {}^n\gamma^l \frac{| {}^nx^l \wedge {}^n_jw^l |}{{}^n\alpha^k + | {}^n_jw^l |} \tag{1} $$

where ${}^nx^k = [{}^nI^k, 1 - {}^nI^k]$ is the activity vector, ${}^n_jT$ is the activation value of the *j-th* node, ${}^n\gamma^l$ is a contribution parameter, ${}^n\alpha^k$ is a choice value and ${}^n_jw^l$ is a weight vector associated with the category *j*. The resonance value for the J-th node (the node with the highest activation value) is checked to be greater than a certain vigilance parameter (${}^n\rho^k$):

$$ {}^n_Jm^k = \frac{| {}^nx^k \wedge {}^n_Jw^k |}{| {}^nx^k |} \geq {}^n\rho^k; \text{where } J = \arg\max({}_jT) \tag{2} $$

If this condition holds true, the weights are updated with the rule:

$$ {}^n_Jw^{k(new)} = (1 - {}^n\beta^k) {}^n_Jw^{k(old)} + {}^n\beta^k({}^nx^k \wedge {}^n_Jw^{k(old)}) \tag{3} $$

where ${}^n\beta^k$ is a chosen learning rate. If the condition does not hold true, resonance is checked for the next largest activation value node. Eventually, the events will be categorized in the input channels (${}^iF_3$) of the event field F_3.

A.2. Episode encoding

As mentioned above, the input channel ${}^iF_3$ categorizes events. Now, the buffer channel ${}^bF_3$ will save the values from an output channel vector ${}^oF_3$. Thus the episodes can be encoded such that:

$$ {}^bx_n = {}^ow\,{}^oy_{n-1} \tag{4} $$

$$ {}^oy_n = {}^iw\,{}^ix_n + {}^bw\,{}^bx_n = {}^iw \sum_{k=0}^{n-1} ({}^bw\,{}^ow)^k\,{}^ix_{n-k} \tag{5} $$

where ${}^ix_n$, ${}^bx_n$ and ${}^oy_n$ are the input, buffer and output vectors, respectively; and ${}^iw$, ${}^bw$ and ${}^ow$ their corresponding weights. Thus the output vectors memorize the event sequences which serve as inputs to the episode layer F_4. These episodes are then categorized in the input channel ${}^iF_4$. The encoding procedure can be repeated to obtain daily episodes F_5.

A.3. Episode retrieval

To retrieve episodes based on partial input cues, Deep-ART reverses the episode encoding process, as described in [1]. Thus, Deep-ART can recall past episodes (hence, task sequences) from similar current situations, just like a human would do.

B. Action sequence generator: FF-planner

Deep-ART can only retrieve episodes in prelearnt scenarios; if the environment changes, Deep-ART will fail to generate correct task sequences. Thus, the fast-forward planner (FF-planner) [8] is used to solve unlearnt tasks based on similar pre-learnt ones. The generated task sequences are transfered and stored back by Deep-ART as an episode. The FF-planner uses enforced hill climbing (EHC) as a search method and A* algorithm if EHC fails. It parses PDDL language [9] to represent a task planning problem as a tuple $\{A,\ s_0,\ s_g\}$, where *A* is a set of actions defined as preconditions and effects between objects and locations, s_0 is the starting state and s_g the goal state. Thus, the FF-planner will generate a new task sequence to achieve s_g based on combinations of pre-learnt behaviours.

C. Trajectory warping for new behaviours

The task sequences provided by Deep-Art and the FF-planner require certain behaviours that have been pre-trained, either by human demonstration or by moving the robots arm passively by an expert [10]. These actions depend on specific start and end states which will surely not match with those in real scenarios because object and end effectors position are variable (i.e. pouring, throwing, shaking...)
As proposed in [3], the end effector pose in a trajectory is represented as a twist vector $\xi = [\omega_x, \omega_y, \omega_z, v_x, v_y, v_z]^T \in \mathbb{R}^6$, where ω is an angular vector and v a linear change. Thus the homogeneous transformation matrix between start ($_s$) and goal (^{g}) points will be:

$$ H^g_s = exp(\hat{\xi}^g_s) = exp\left(\begin{bmatrix} 0 & -\omega_z & \omega_y & v_x \\ \omega_z & 0 & -\omega_x & v_y \\ -\omega_y & \omega_x & 0 & v_z \\ 0 & 0 & 0 & 0 \end{bmatrix}^g_s \right) \tag{6} $$

The warping variation (H_{diff}) is obtained knowing the end-effectors start and end poses of the demonstrated trajectory and those of the actual environment ($H^s_o, H^g_o, \hat{H}^s_o, \hat{H}^g_o$, respectively), related to the origin frame ($_o$). Then, the warped twist vector $\hat{\xi}_{diff}$ is obtained:

$$ H_{diff} = (H^s_o - H^g_o)^{-1} \cdot (\hat{H}^s_o - \hat{H}^g_o) \tag{7} $$

$$ \hat{\xi}_{diff} = log(H_{diff}) \tag{8} $$

Finally, the end effector poses of the demonstrated trajectory are warped. H^k_s is the end-effectors pose at a *k-th* point in the demonstrated trajectory, *N* is the total number of these points and $\hat{H}^k_s$ are the end-effectors poses in the warped trajectory:

$$ \hat{H}^k_s = \hat{H}^s_o H^k_s exp(k \frac{\hat{\xi}_{diff}}{N}) \tag{9} $$

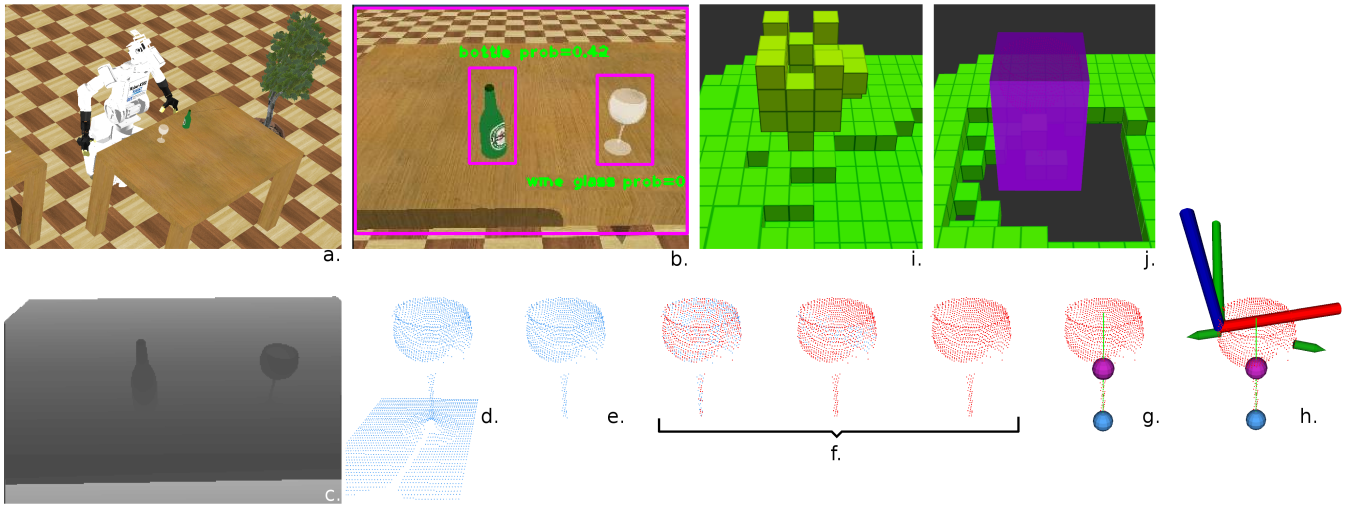

Fig. 2. Perception pipeline for unknown objects: **a.** Simulation on *Webots* **b.** CNN results **c.** Rectified Depth image **d.** Point cloud (wineglass) **e.** Result of segmentation **f.** Registration (red) of new point clouds (blue) **g.** Data extraction: centroid (purple sphere), surface contact (blue sphere), principal direction (green vector) and overall size of point cloud **h.** Possible grasping poses (green arrows) and selection of best one (large axis) based on cosine similarity with closest hand pose **i.** Generated octomap **j.** Collision object based on extracted data that remove collision checking between robot and octomap.

To follow these warped trajectory points whilst, simultaneously, avoiding collisions, the *OMPL MoveIt* implementation of the sampling based motion planning algorithm Quick-RRT* [11] is applied.

III. Perception pipeline

The main contribution of this paper is adding an online object detection and segmentation system to the Task Intelligence problem. That is, adding the necessary perceptive means for the robot to understand a dynamic environment, memorize concepts and act accordingly.

Prior to this, the robot must evaluate whether the action or task that is to be carried out relies on the knowledge of a certain object. If so, the knowledge could be there already, from previous events or from manual human input. Nonetheless, if said object's information has not yet been found and memorized, the following section describes the perceptive procedure to do so. All in all, the procedure follows the pipeline presented by *Llopart et. al.* [12], with the difference that now only one object is being segmented allowing for faster processing times and better performance.

A. Detection

If no information of the object, included in the requested behavior, has not yet been stored in memory, it is necessary to find it first. Initially, if Deep-ART retrieves a pre-learnt position where the object is usually found, the robot will navigate to that location (i.e if the robot is requested to grab a knife, these are commonly found on the kitchen table). Once the position is reached, the robot will *look around* its environment until it finds the object using an RGB-D sensor mounted on the head (Fig. 2a.). The robot will turn its head 30° both pitch and yaw-wise with a 3° step. It will evaluate if the object has been detected at every step.

Object detection, recognition and semantic labelling is done using a Convolutional Neural Network (CNN) through which Regions of Interest (ROI) are extracted in real time around a labeled class (Fig. 2b.). The YOLO Real-Time Object Detection System ([13]) is employed since it has been pre-trained on the COCO dataset to detect up to 80 different classes, which include, amongst others, common household items and kitchenware. Then, the obtained bounding box is converted into a binary mask which is used to crop the obtained rectified depth images (Fig. 2c.). By converting this data to a point cloud, only the points that are within the found ROI will be accepted (Fig. 2d.). These will represent mainly the desired object plus some background and noise. The centroid of the point cloud is then found (which represents a vague estimate of the objects position) and stored in memory (inside an *object's class*) with the object's semantic label found by the CNN. This way, the information can be later referenced if the object has to be found again; this positional data can be updated online if a more precise value is found or if the object is moved.

B. Segmentation and registration

Once an estimated position of the object is found, and the robot is correctly facing it, the segmentation process takes place. Partial point clouds representing the object are determined continuously but they include noise and background information which needs to be removed to obtain an accurate model of the object.

Initially, all point clouds are filtered to remove encoding errors (NaN) and noise, thus an outlier removal filter (with a radius of 10 cm and a minimum point threshold of 4) is applied. Additionally, a pass through depth filter is also employed to remove those points closer than 20 cm to the sensor (they are probably noise too) and those further than 5 meters since they lack precision.

Up to 7 segmentation methods are applied based on the semantic label previously found by the CNN, as described

in [12], to remove unwanted data (Fig. 2e.). This means that depending on the object detected, one or another segmentation method is selected. These remove those points that do not represent the object (such as tables or walls). Moreover, some of these methods derive additional information about the object. For instance, cylindric segmentation gives an estimate of the radius and height of an object (used for bottles or cups), whilst spherical segmentation only provides the radius (used for apples or oranges). These values are then also stored inside the *object's class*.

A more reliable model of the object can be built by registering multiple object's point clouds over a small amount of time (Fig. 2f.). This procedure is accomplished by applying the Iterative Closest Point (ICP) between two sets of point clouds, and, if it converges, merge the transform of one point cloud with the other. The ICP algorithm minimizes the point coordinates between two sets of point clouds to find the best data transform between them; this allows to take the continuously generated point clouds and add them up together, if they are close enough already, resulting in the best possible model of the object. Plenty of information can be then extracted from these models.

C. Extracting data

To be able to interact with and manipulate objects, some data must be obtained from the objects models (Fig. 2g.). This knowledge is updated in the *object's class* that is being stored in memory.

A better centroid estimation of the object is found; hence, the object's initial estimated position is updated. The normalized covariance matrix and the eigenvectors that represent the principal axes of the point cloud are derived to find the object's orientation. After transforming the point cloud to the new reference frame, the maximum and minimum point values in each direction are obtained. This provides significant insight into the objects dimensions and allows the extraction of values such as maximum height, point of contact with surface (i.e in the cases where the objects are on top of tables) and, most importantly, serves to delimit the point clouds dimensions.

D. Collision object

The dimensionality information is also stored in memory because, with it, a *collision object* can be created (Fig. 2j.). These are built using a rectangular prism, cylinders or spheres, based on the labeled semantics of the object. This means that those objects that have been segmented using cylindrical or spherical methods will require a cylindrical or spherical *collision object*, respectively; whilst the rest will be modeled as a rectangular prism for simplicity.

The reason for constructing such *collision objects* is that it enables a robot running the *MoveIt* libraries from ROS to discern between objects and a generated octomap (Fig. 2i.). The octomap is a 3D occupancy grid mapping method that models the robots surroundings as occupied, free or unknown space, which helps avoid collisions when navigating or moving around. However, objects will always be included inside the occupied space which will deny the robot any possible interaction with them. Therefore, by adding *collision objects* to the octomap, the collision checking between robot and objects is disabled, allowing said interaction.

E. Grasp pose selection

Selecting a correct grasping pose is a very relevant step prior to any sort of interaction between robot and object. The *agile_grasp* [14] package requires only a pointcloud (geometry of the objects model) and the robotic grippers dimensions to be able to sample grasp hypothesis which are then validated, if the grasping points are antipodal, through a Support Vector Machine (SVM). This procedure results in an array of grasping poses (represented as green arrows in Fig. 2h.) from which the best one must be selected.

The best grasping pose within the extracted array of poses has to be selected. This is attained by finding the current position of both arms end effectors and deriving the Euclidean distance from each of them to the objects centroid (calculated before). This metric helps know which hand is closest to most grasp poses, making the path planning easiest.

The cosine similarity measure ($cos(\theta)$) is calculated for all the elements inside the grasping poses array to select the best pose candidate. This value measures the similarity between two non-zero vectors by calculating the cosine of the angle between them. Therefore, it evaluates orientation and not magnitude: having a cosine similarity of 1 means both vectors have the same orientation, if the result is 0 the vectors are orthogonal, and if the cosine similarity is -1 the vectors are diametrically opposed.

$$cos(\theta) = \frac{A \cdot B}{\|A\|_2 \|B\|_2} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{10}$$

Where A_i and B_i are components of a vector *A* and vector *B*, respectively. Vector *A* describes a vector parallel to the ground and the robot (thus following the X reference axis) and its direction is determined by the hand closest to the object, which was selected before: if the right arm was selected, vector *A* has a value of *(-1, 0, 0)*, if the left arm is selected, the vector takes the value *(1, 0, 0)*. Vector *B* is the orientation of each grasp pose in the array. The grasp pose with the highest cosine similarity (larger than 0 and closest to 1) is the chosen one.

However, sometimes, due possible malformations of the grasping poses in which their direction might be opposite from the closest arm, all cosine similarity results will be smaller than 0. In this special case, a new grasping pose must be derived. To achieve this, the cosine similarity procedure is repeated but using the reversed grasp orientation vectors and the centroid of the object. All in all, a final correct grasp pose is selected based on the closest arm to the object, the objects geometry and the hands dimensions.

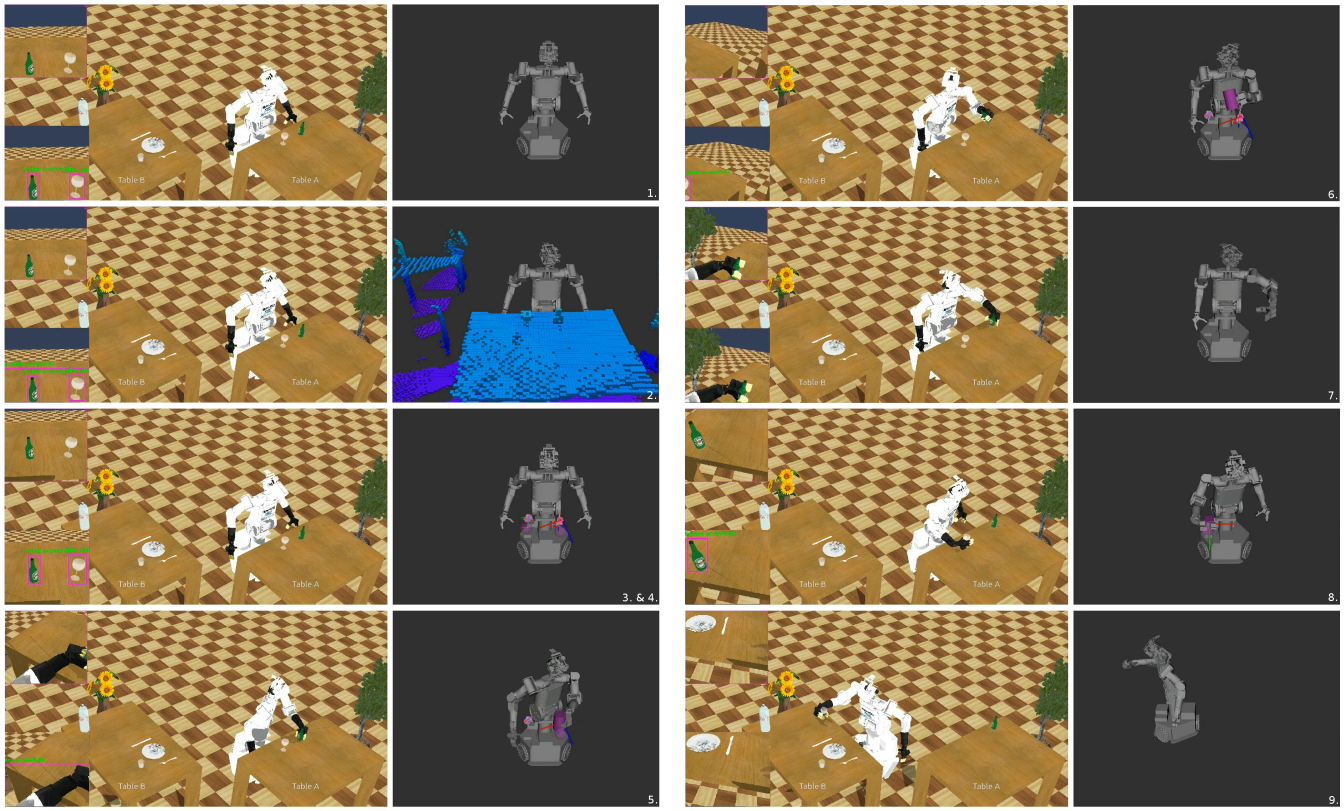

Fig. 3. The images depict the behaviours retrieved by Deep-ART for the *Bring wine to Table_B* Task. Their actions and order are shown in Table II. **Left.** *WeBots* simulator illustrating the current view of the robot (upper image) and the results provided by the CNN (lower image) **Right.** RViz visualizer showing object segmentation and data extraction.

IV. Experiments

The proposed perceptive pipeline for Task Intelligence system has been tested on a simulated environment to evaluate its performance and effectiveness. The humanoid robot *Mybot* was modelled and used in this simulation.

A. Mybot Humanoid Robot

The Robot Intelligence and Technology Laboratory (RIT) at KAIST has developed a wheel-based humanoid robot, *Mybot*. It uses an Odroid XU board with Ubuntu 16.04 and ROS Kinetic Kame. The robot incorporates a total of 26 Degrees of Freedom (DOF): 8 DOF for each arm and 3 per hand, with 2 additional DOF in the waist (tilt and pan). The mobile base is omni-directional for easy navigation and contains a 2D laser rangefinder. The head, containing an Xtion ASUS sensor used to obtain rectied RGB and Depth images of the surroundings, has 2 DOF (pitch and yaw).

B. Experimental Setup

All tests have been run inside the *Webots* simulator; in the future, the real robot will be used. A desktop computer with a GTX 1080 GPU card and running Ubuntu 16.04 and ROS Kinetic Kame is employed for such simulations. A scenario with two tables is reproduced, where each table has a series of objects on it. The Deep-ART has pre-learnt a series of episodes using PDDL (the ones from [3]) and an additional one which includes the task sequence of pouring wine from a bottle onto a glass. The exact locations, geometry and grasps of the objects are initially unknown since finding those values is the main objective of the proposed approach.

C. Experimental Results and Discussions

When the robot is asked to "bring a cup of wine to table B", Deep-ART inferred a task sequence from the partial cue. Specific actions/behaviours were also retrieved by the FF-planner because they had not been learnt beforehand.
Initially, the robot would move to Table A where the wine bottle and glass were supposed to be on. To travel there, an octomap of the environment must be generated which is updated online. This allows collision avoidance for all other actions of the robot in the future. Once the robot is in front of Table A, it must look around to find precisely where the required objects are. The procedure has been described in Section III and results in the detection of the specified objects and their correct labeling, the precise estimate of the object's position on top of the table, their centroid, geometry and dimensions, and the best grasping pose available based on the closest hand to them.
In this case, the wine bottle was grasped and, knowing the position of the ring/top neck of the bottle, its grasping pose and the centroid of the wine glass, a previously learnt *pouring* behaviour is warped, as seen in Section II-C, to match these initial and final states. It is worth mentioning that if the

TABLE II

TASK: BRING CUP OF WINE TO TABLE_B

1	MOVE to *Table_A*
2	CREATE *octomap*
3	FIND *Wine_glass*
4	FIND *Wine_bottle*
5	GRASP *Wine_bottle*
6	POUR *Wine_bottle* in *Wine_glass*
7	RELEASE *Wine_bottle* on Table_A
8	GRASP *Wine_glass*
9	MOVE to Table_B
10	RELEASE *Wine_glass* on Table_B

grasping of an object with one hand or its motion planning fail, the same pipeline takes place but using the other hand to be able to try all possible cases before the robot gives up. Once the pouring action ends, the wine bottle is released again on top of the table and the (now full) wineglass is grasped and taken to Table B where it is released, ending the whole episode. This procedure is described in Table II adn seen in Fig. 3, where the locations and objects are written in *italic* and the behaviours in *capitals*.

V. CONCLUSION

In this paper, an additional module for the Task Intelligence problem is proposed. The previous system works in constrained environments where most of the object's information was pre-defined. This system is formed by three modules: the Episodic Memory (Deep-ART) retrieves a task sequence after a certain partial input cue. The action sequence generator (FF-planner) allows the creation of new task sequences to solve unlearnt cases. Finally, the trajectory warping module is used to perform specific behaviours for each task based on the robot's and object's current states.

The presented perceptual pipeline builds on and enhances the preceding system, allowing the robot to understand and interact with a dynamic environment whilst updating online all information on the surrounding objects. Thus, this more generalized approach requires considerably less prior knowledge of the environment and objects than previous approaches but equips the robot with the ability to learn about its surroundings dynamically, memorize the new data and use it to its advantage. This is achieved using a CNN to detect and recognize objects, various segmentation methods, a selection of best grasping poses and a XMl parser to be able to share or update the newly registered data with the other Task Intelligence modules.

The results of the simulated experiment show a robot that, given a partial instruction, retrieves the necessary task sequence from previously learnt episodes (memories). It uses the newly presented perceptual pipeline to comprehend and extract data of its environment. It then follows the retrieved task sequence, modifying its behaviours based on the objects data, until it finally accomplishes its goal.

VI. FUTURE WORK

The proposed pipeline works as intended in the presented test case. However, further scenarios must be developed, not only in simulation, but also in the real world. Having a wider range of demonstrated trajectories (actions) and a larger number of recognizable classes by the CNN, would allow for more complex and interesting task sequences. Novel neural networks, like Mask-RCNN (proposed by *He et. al.* [15]) result in the pixel-wise semantic segmentation of objects. The usage of networks like these, even at the cost of a lower processing speeds, allows for the elimination of the segmentation step in the 3D point cloud domain (as described in Section III-B). Finally, adding a multi-viewpoint perspective ([16] and [17]) when registering the objects point clouds would entail more reliable objects models, thus more accurate extracted data and better grasping poses.

REFERENCES

[1] G.-M. Park, Y.-H. Yoo, D.-H. Kim, and J.-H. Kim, "Deep art neural model for biologically inspired episodic memory and its application to task performance of robots," *IEEE transactions on cybernetics*, vol. 48, no. 6, pp. 1786–1799, 2018.

[2] I.-B. Jeong, W.-R. Ko, G.-M. Park, D.-H. Kim, Y.-H. Yoo, and J.-H. Kim, "Task intelligence of robots: Neural model-based mechanism of thought and online motion planning," *IEEE Transactions on Emerging Topics in Computational Intelligence*, pp. 41–50, 2017.

[3] D.-H. Kim, G.-M. Park, Y.-H. Yoo, S.-J. Ryu, I.-B. Jeong, and J.-H. Kim, "Realization of task intelligence for service robots in an unstructured environment," *Annual Reviews in Control*, 2017.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[5] A.-H. Tan, G. A. Carpenter, and S. Grossberg, "Intelligence through interaction: Towards a unified theory for learning," in *International Symposium on Neural Networks*. Springer, 2007, pp. 1094–1103.

[6] W. Wang, B. Subagdja, A.-H. Tan, and J. A. Starzyk, "Neural modeling of episodic memory: Encoding, retrieval, and forgetting," *IEEE transactions on neural networks and learning systems*, vol. 23, no. 10, pp. 1574–1586, 2012.

[7] G. M. Park, Y. H. Yoo, and J. H. Kim, "Rem-art: Reward-based electromagnetic adaptive resonance theory," in *Proceedings International Conference on Artificial Intelligence (ICAI)*, 2015, pp. 805–811.

[8] J. Hoffmann and B. Nebel, "The ff planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research*, vol. 14, pp. 253–302, 2001.

[9] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 639–646.

[10] A. Jain, S. Sharma, T. Joachims, and A. Saxena, "Learning preferences for manipulation tasks from online coactive feedback," *The International Journal of Robotics Research*, vol. 34, no. 10, pp. 1296–1313, 2015.

[11] I.-B. Jeong, S.-J. Lee, and J.-H. Kim, "Rrt*-quick: A motion planning algorithm with faster convergence rate," in *Robot Intelligence Technology and Applications 3*. Springer, 2015, pp. 67–76.

[12] A. Llopart, O. Ravn, N. A. Andersen, and J.-H. Kim, "Generalized framework for the parallel semantic segmentation of multiple objects and posterior manipulation," in *Robotics and Biomimetics (ROBIO), 2017 IEEE International Conference on*. IEEE, 2017, pp. 561–568.

[13] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[14] A. ten Pas and R. Platt, "Using geometry to detect grasp poses in 3d point clouds," in *Robotics Research*. Springer, 2018, pp. 307–324.

[15] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE, 2017, pp. 2980–2988.

[16] J. McCormac, A. Handa, A. Davison, and S. Leutenegger, "Semanticfusion: Dense 3d semantic mapping with convolutional neural networks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 4628–4635.

[17] C. Li, H. Xiao, K. Tateno, F. Tombari, N. Navab, and G. D. Hager, "Incremental scene understanding on dense slam," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 574–581.