Knowledge representation of human centred automated manufacturing processes for Industry 4.0 and beyond

handed in
PRACTICAL COURSE

B.Sc. Sherif Shousha

born on the 17.07.1994
living in:
Agnesstrasse 31
80798 Munich
Tel.: +49 - 15773776335

Human-centered Assistive Robotics
Technical University of Munich

Univ.-Prof. Dr.-Ing. Dongheui Lee

Supervisor: M. Eng. Matteo Pantano

TECHNISCHE UNIVERSITÄT MÜNCHEN
Human-centered Assistive Robotics
UNIV.-PROF. DR.-ING. DONGHEUI LEE

February 12, 2021

PRACTICAL COURSE
for
Shousha Sherif, Mat.-Nr. 03704289

Knowledge representation of human centred automated manufacturing processes for Industry 4.0 and beyond

Problem description:

Manufacturing processes compromise several steps for the transformation of raw materials into finished products for the market (value chain). In case of new products such steps require meticulous engineering for fine-tuning machines and robot programs to satisfy functional requirements, specifications and directives. Due to the processes' complexity, usually thoughtful documentations is the de-facto approach for ensuring a smooth integration. However, in current times, where push to digitization is stronger and contacts have to be limited, digital collaboration becomes an essential asset for companies.

The a-priori defined 4th Industrial revolution forecasted these possibilities and proposed architectures for documentation through the so-called connected world [1]. However, clear applications of such concepts are still futuristic and more confined to research activities without a clear manufacturing industrial usage [2]. Therefore, in this advanced seminar you will research and propose methodologies for the digital documentation. This will compromise diving into knowledge representation strategies for human centred automated manufacturing processes.

Tasks:

- Managing and discuss solutions of the research group
- Implement own methods for knowledge representation
- Testing of the represented methods on some simulated robots

Bibliography:

[1] H. Kagermann, W. Wahlster and J. Helbig, "Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Securing the future of German manufacturing industry", Industrie 4.0 Working Group - Forschungsunion, Berlin, BE, Germany, Apr. 2013.

[2] Tenorth, Moritz, and Michael Beetz, "KnowRob: A knowledge processing infrastructure for cognition-enabled robots", The International Journal of Robotics Research, Vol. 32, No. 5, 2013, pp 566-590.

Supervisor: M. Eng. Matteo Pantano

(D. Lee)
Univ.-Professor

Contents

List of Figures

Acronyms

FCL	Fast Collision Library.
FOON	Functional Object-Oriented Network.
GUI	Graphical User Interface.
OMPL	Open Motion Planning Library.
ROS	Robot Operating System.
SRDF	Semantic Robot Description Format.
URDF	Universal Robot Description Format.

1 Introduction

1.1 Motivation and Problem Statement

The economic landscape is changing and is influenced by efficient automation, manufacturing technologies, and data exchange systems. A high degree of digitalized and networked processes in industrial operations enables a strong desire for more interfaces in development and production. Manufacturing processes involve various stages in the processing of raw materials into finished products for the marketplace. However, an increase need for a more efficient autonomous and customizable production has led to a new industrial revolution, the "industry 4.0", which represents the link between industrial production and information technology [1] [2].

Autonomous service robots are being used in industry 4.0 to understand the vaguely described tasks, such as "sort different products" or "put products in boxes" and different kind of object manipulations. To understand and execute complex commands such as "sort different products" they need to understand the missing pieces of information that are not spelled out explicitly in these vague instructions [1] [3]. For instance, the robot has to determine which items should be taken and where they shall be placed to complete: "put products in boxes".

In this project, we decided to implement a robot arm manipulation task, using a knowledge representation concept. The knowledge representation concept used in this internship is called the functional object-oriented network (FOON) [4]. The graphical model FOON is used to model the connectivity of the functional-related objects and their motions in manipulation tasks. Using a FOON, robots can interpret a task's goal, look for the correct objects at the wanted states on which to operate, and finally generate a sequence of manipulation motions[5][6].

In this internship, we analyze a beer factory use case. The beer factory produces different kinds of beer and different beer boxes for each sort of beer. In this case, the robot arm should be able to understand the commands given. For example, with the command "sort augustiner", the robot starts looking in the surrounding environment for all relevant objects to "augustiner" and their state, then it uses the positions of these relevant objects to plan a sequence of motions to complete the task and change the state of each relevant object after the motion, such as "empty box" to "full box" and "unsorted beer" to "sorted beer". We used the industrial robot arm "Franka Emika Panda" [7] to achieve this tasks in a simulated environment. Moreover, we used the "moveit ROS" [8] [9] open source for manipulating the arm, planning the motion, and building the objects in the 3D visualization tool "RViz ROS" [10]. The ROS 1 (Melodic Morenia) is utilized to create the behavior of the robot arm. The code is available on Gitlab and Git repository [1] [2].

[1] Gitab: https://gitlab.lrz.de/ge98law/knowledge-graph-for-industry-4.0
[2] GitHub: https://github.com/SherifShousha/MoveIt_FOON-for-industry-4.0

2 State of art

In this chapter, we will present the basics of the FOON, which represents manipulation tasks to connect interactive objects with their functional motions. We will describe the main functionality of the FOON, and then we will present the main parts of the FOON, the nodes and the functional units. After that, we will discuss the state of the art of the Robot Operating system (ROS) and its communication structure. Finally, we will present the MoveIt, its interfaces with ROS, and how to use the MoveIt setup assistant.

2.1 FOON

The functional object-oriented network (FOON) is a graph-based network used for encoding knowledge about manipulation tasks by encrypting the flow of actions coming one after another. Thus, the FOON used by a robot for solving manipulation problems given a target goal. Originally, FOON was implemented for cooking and kitchen domain, and the FOON can be extended to work in other domains and environments like industrial ones.

2.1.1 Nodes of FOON

The FOON is a bipartite network including two kinds of nodes, the motion and the object state nodes. The interaction of the manipulation motion of input objects will result in a change in the state of the input objects into the output object state. Each input object state node is connected to the output object state nodes through the motion node. This pattern only allows the object state nodes to be connected to motion nodes and the motion nodes to be connected to object nodes, which thus creates the bipartite network. As shown in Figure 2.1, the two input object state nodes are connected to output object state nodes through the motion node, since in reality the change in the state of an object is caused by the manipulation that is represented from the motion node. Furthermore, no object state node is connected to another object state node directly, since the object state can not be changed without manipulation.

In FOON, each object node is unique regarding its name, state, and attributes. For instance, in a manipulation task, the object state node represents the object in a specific state when the robot manipulates the object. Then, the object state will be changed after manipulation. The manipulation is created from the motion node, which contains the type and information about the manipulation. For example, in real life, if we have an apple and a knife, we can cut the apple into pieces. The apple and the knife are presented as two input object state nodes, the apple object state node has the state “whole” and the knife object state node has the state “clean”. The cutting motion node will change the state of the apple from “whole” to “cut” and the state of the knife from “clean” to “dirty”. The apple with the changed state “cut” represents the first output object state node, and the knife with the changed state “dirty” represents the second output object state node, as shown in Figure 2.1.

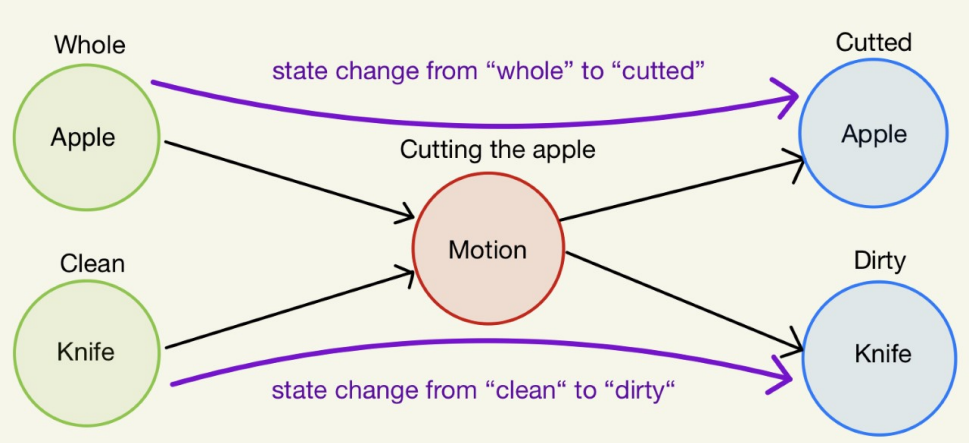

Figure 2.1: A basic functional unit with two input nodes (green nodes) and two output nodes (blue nodes) connected by an intermediary single motion node (red node). The purple arrows refer to the state change through the motion node.

2.1.2 Functional Unit

The functional unit is the minimum learning unit in a FOON. The functional unit describes a single atomic action that is part of an activity. It is important to note that an activity could be viewed as a series of actions. For example, in the activity of serving a beer, one functional unit would be picking up the bottle of beer from the table, another functional unit would be opening the bottle, and the final functional unit would be serving the beer.
Every functional unit has three components: input object state nodes, output object state nodes, and a motion node that represents the action that causes a change in the input objects' states. A functional unit represents the transformation of object states before and after a manipulation motion occurs, this is described by input object state nodes (object state before manipulation) and output object state nodes (object state after manipulation).

2.2 ROS 1 (Melodic Morenia)

ROS (Robot operating system) is a flexible open-source framework for writing robot software. In practice, ROS is essentially a large collection of libraries, tools, and protocols that aim to simplify creating complex and robust robot behavior across many robotic platforms. Its most important advantage lies in its powerful generalized Transmission Control Protocol/Internet Protocol (TCP/IP) communication framework based on publish/subscribe pattern allowing to easily and efficiently handle the variety of information transiting in a robot such as sensors data, video, and control orders.

Computations in ROS are performed through a network of concurrent processes called nodes. Within this network – known as the ROS computation graph – nodes can communicate with each other synchronously or asynchronously using a set of dedicated routines. Nodes are pieces of software that can be written in C++ or Python. They can take care of a small subset of tasks, for example, reading a sensor or controlling a servo, those nodes connect through a publish/subscribe protocol. That means that if a node has a piece of information to share, it will share this information using a topic, and if another node is interested in that information, it

subscribes to that topic and reads the information. The main elements of the ROS computation graph are nodes, master node, topics, services, actions, bags, parameter server, and messages.

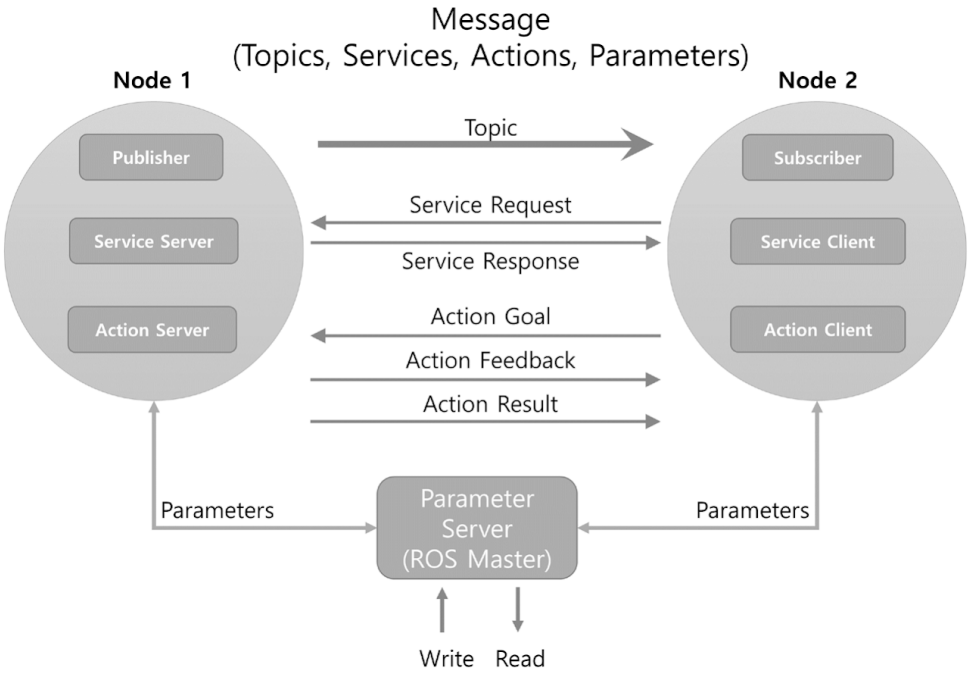

Figure 2.2: ROS formal overview, showing the different message communication and data exchange methods between two nodes.

2.3 MoveIt

MoveIt is a state of the art software for manipulation, combining the most advanced progress in motion planning, 3D perception, control, kinematics, and navigation. MoveIt provides an easy-to-use platform for developing robotics applications, evaluating new robot designs, and building integrated robotic products for Manufacturing processes. MoveIt is a widely used open-source software for manipulation and has been successfully integrated with 65 robots. It is written entirely in C++ but also includes Python bindings for higher-level scripting. The MoveIt software structure allows the use of MoveIt with unlimited robotic frameworks by creating a formal separation between core functionality and robotic framework dependent aspects such as communication between components.

MoveIt uses by default the core ROS build and messaging systems. To be able to swap components easily, moveIt uses plugins for most of its functionality as follows:

- Motion planning: The interface is through a ROS action or service. The motion planners for "move_group" are configured using the MoveIt Setup Assistant. MoveIt unites directly with Open Motion Planning Library(OMPL) and uses the motion planners from that library as its primary/default set of planners. OMPL is an open-source motion planning library that essentially implements randomized motion planners.

- Collision detection: MoveIt is currently using the Fast Collision Library (FCL) [11].

- Planning scene: It represents the environment surrounding the robot and also saves the state of the robot. It is sustained by the planning scene monitor inside the "move_group" node.

MoveIt setup assistant is a Graphical User Interface (GUI) that guides new users through the initial configuration requirements of using a custom robot with the motion planning framework. It accomplishes the objective of immediacy for the user by automatically generating the many configuration files necessary for the initial operation of MoveIt. These configurations include a self-collision matrix, planning group definitions, robot poses, end effector semantics, virtual joints list, and passive joints list.

One of the robots that can be generated with the MoveIt setup Assistant to be used in RViz is the FRANKA EMIKA PANDA robot by installing the robot arm from the URDF file.
The robot is designed to act like a human arm with high sensitivity. It includes two parts, the arm part and gripper part. The arm part includes seven joints, which are used to move the robot in all directions. The gripper part includes two joints, which are used to pick and place the objects in the environment.

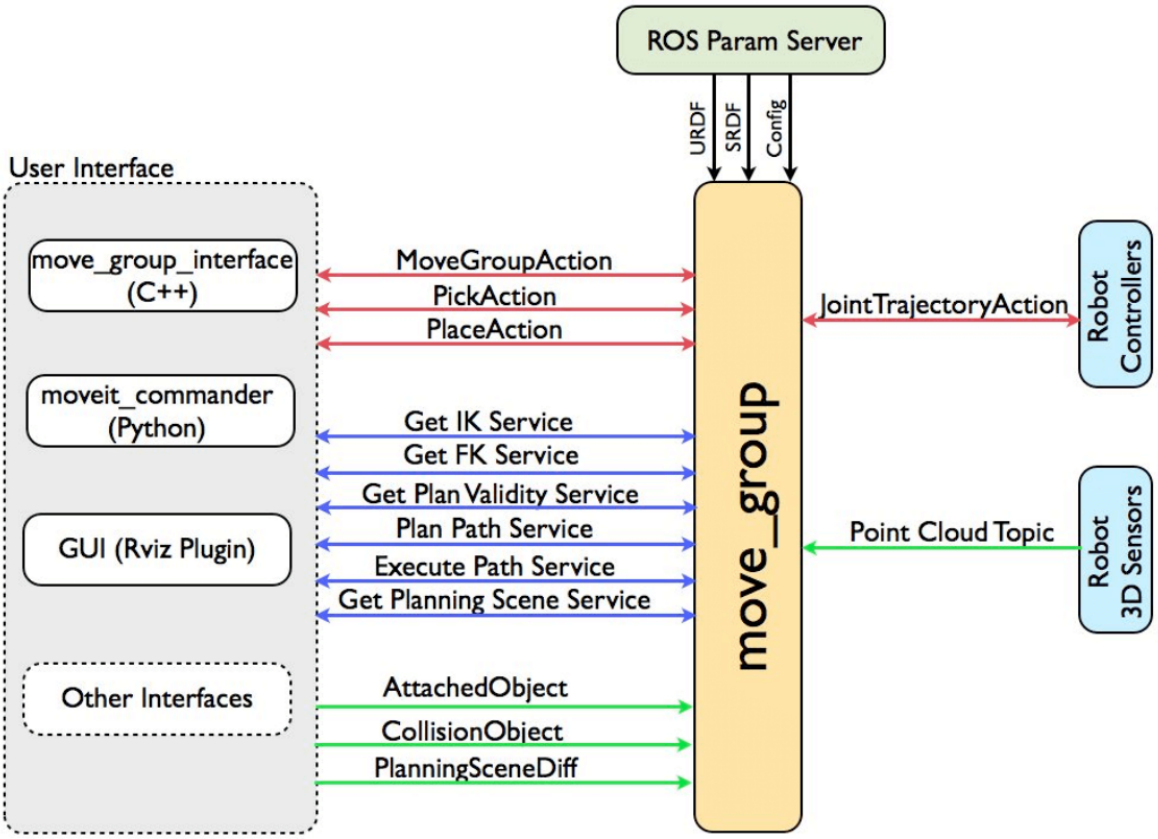

Figure 2.3: MoveIt architecture.

3 Implementation

The goal of the implementation was to make the robot acting human-like by using the FOON graph to extract the needed object to achieve a specific goal. When the robot received a command, it had to know the input object and its state and it had to plan the motion sequence, which led to the change in the object state to complete the task successfully.

In our implementation, we created an environment that includes two kinds of beer, "augustiner" beer and "rothaus" beer. Each beer has its own beer box. The environment begins with the beer bottles located on a table and the boxes located on another table. The robot should use a input command like "sort beer" to build a sequence of actions using the FOON to sort every kind of beer in its own box. Furthermore, the robot should be able to sort any kind of beer to its box from the input command. For instance, when the robot gets the command "sort augustiner", the robot should be able to recognize the needed objects and look for the augustiner beer bottle and the augustiner box. The robot should then get that the state of the beer bottle is "unsorted" and the state of the box is "empty". After that, the robot generates manipulation motion sequences from the FOON to put the beer in its box. The output from this sorting task is the beer bottle with the state "sorted" and the box with the state "full".

The robot should gather all existing beer bottles from the same kind in its own box before changing the state of the box from "empty" to "full". The robot arms use "pick" and "place" functions with different input information unique to every object in the processes, which means that all motion nodes in the FOON are consisted of these two manipulation functions, but the sequence of this motion, the targeted positions, and orientation are not the same within every manipulation problem.

To achieve that, rospy and its build system "catkin" environment were needed. The implementation started in two parallel parts. The MoveIt part and FOON part. The MoveIt part included the robot arm structure, the joints number, and the joints movement limit. Moreover, the MoveIt implementation included the movement planning and needed move function like pick and place behavior, which should be human-like. The FOON part can be referred to as the "robot brain", since this part included all existing objects in the environment, and planning and creating the sequence of action to achieve the desired goal.

3.1 FOON

The FOON should ideally be able to train automatically from observing human activities. However, due to the complexity of the object, state, and motion recognition, we decided to give the FOON the objects and motion information. The objects information list included:

- The object ID.

- All possible object's states.
- The position of the object in the environment.
- The neighbor objects or, in other words, the objects' ID, which are interacting with the current object.
- Information that is needed to build the object in the MoveIt RViz.

And the motion list included:

- The motion's ID.
- The motion's type.
- MoveIt needed information such as the targeted object's ID, position, pitch, yaw, roll, start supported surface, and end supported surface.

We then implemented the needed classes to build the FOON. We started with the superclass "Thing", which included the initial required information about the object and motion, such as the ID and type and interacting objects. Furthermore, we implemented two subclasses, the "object" class and the "motion" class, which inherits the superclass "Thing" and extends the functionality of the inherited methods.
Finally, we created the "functionalUnit" class, which is used to build the atomic unit of the FOON. This class collected the information of the input objects and the relevant motion sequence, and the new output object state. That means the "functionalUnit" class usually includes four or more object and motion classes.

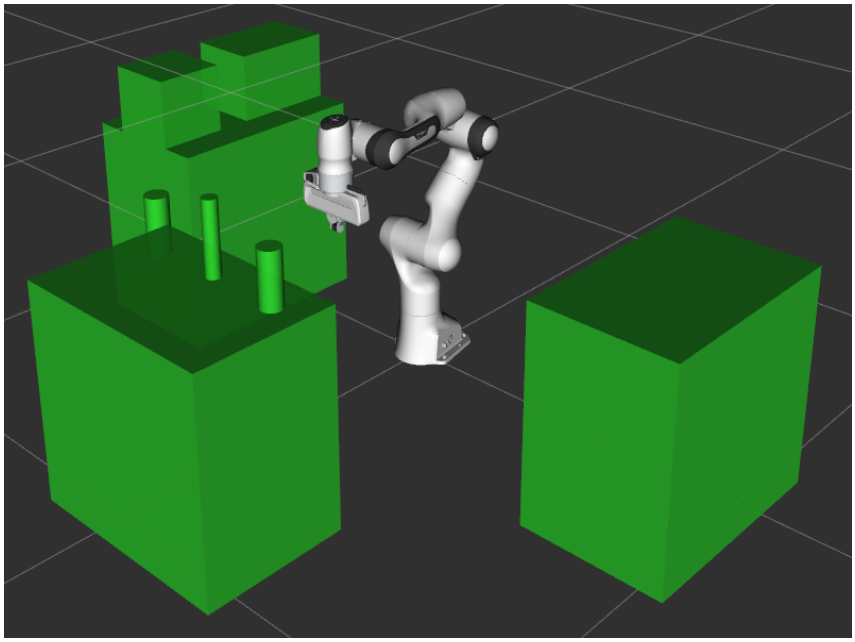

Figure 3.1: MoveIt RViz after build the targeted objects.

We used the object and motion lists and FOON classes in file "FOON_structure" to build the targeted objects and motions. First, we build all the objects in the list as object classes, and then the object classes are sent from the FOON to the MoveIt to construct these objects in the environment, as shown in Figure 3.1.

Then the built objects in the FOON are filtered by their type. For instance, the augustiner beer bottles and augustiner box will be filtered as the type of "augustiner". These filtered objects are sent to the "buildFunctionalUnit" function to create the inputs objects state nodes in their own functional unit.

Simultaneously, the motions are observed from the motions list and create a sequence from these motions to achieve the task goal. For example, the motion pick augustiner bottles, place augustiner bottles, pick augustiner box, and place augustiner box are created as "sort augustiner" motion. After that, this created motion is sent to the "buildFunctionalUnit" function to generate the motion node in its own functional unit.

The FOON is created from one or more functional units. The input command reports the FOON, which functional units should be used to accomplish the current task. Like in "sort beer" command case, the FOON merges the two functional units "augustiner" and "rothaus" to sort all the beer bottles in the environment. This example can be extended to include hundreds or thousands of functional units based on the complexity of the goal task.
For more resolution of the FOON work, we used Networkx [12] to visualize the current FOON graph and synchronize the Graph with the robot arm. The graph shows in which part of the FOON the robot is working in every step. An example for the "sort augustiner" command is shown in Figures 3.2 and 3.3. The Graph 3.3 describes the current step of the robot in the FOON with a blue edge. The NodeTree includes all functional units (command "sort beer") is shown in Figure 3.4.

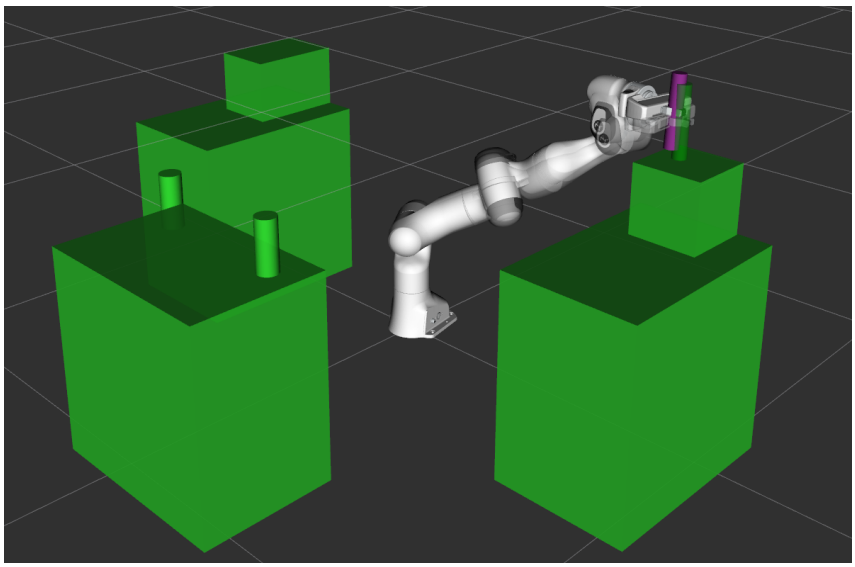

Figure 3.2: The robot is moving the augustiner bottle.

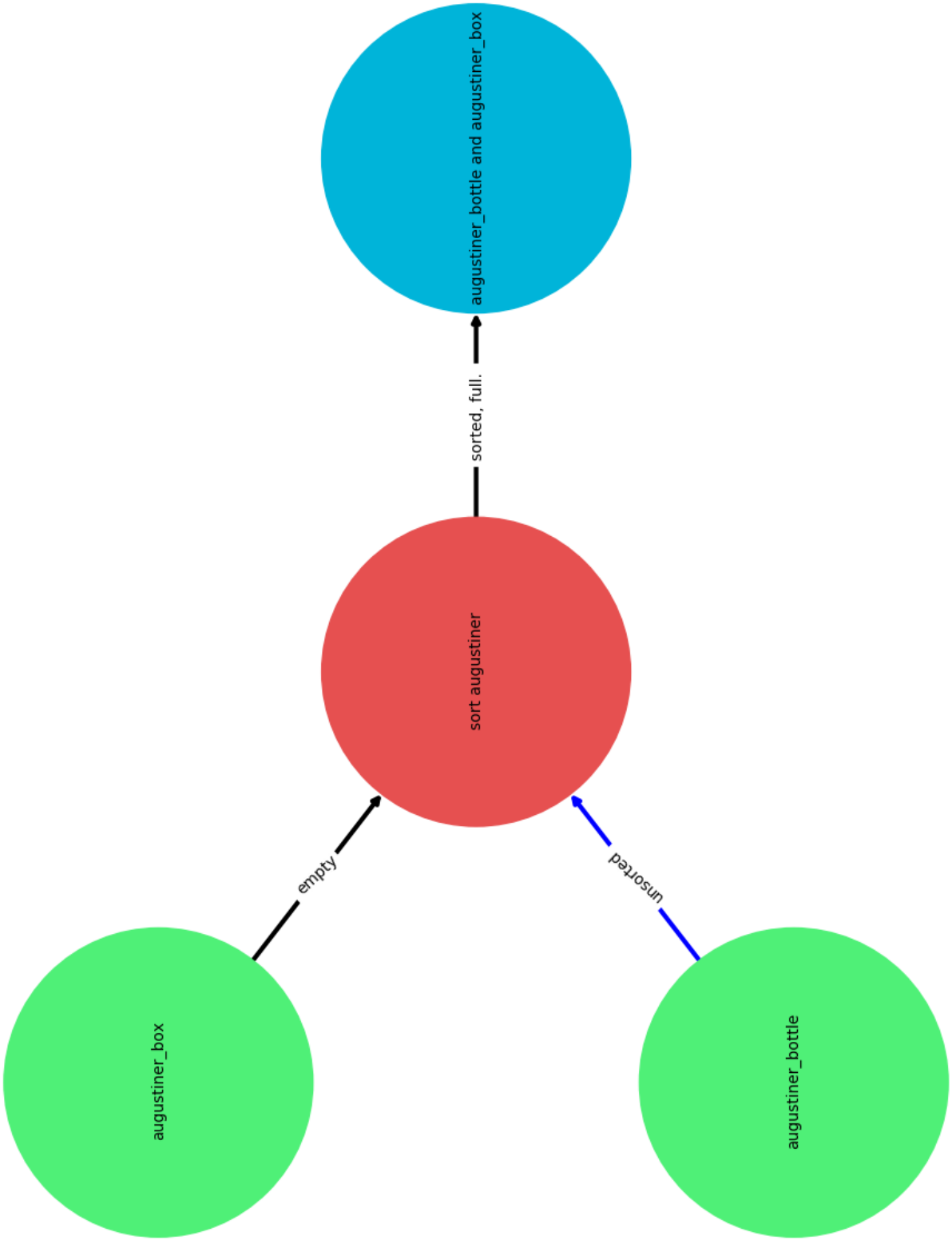

Figure 3.3: The augustiner bottle edge is colored with blue.

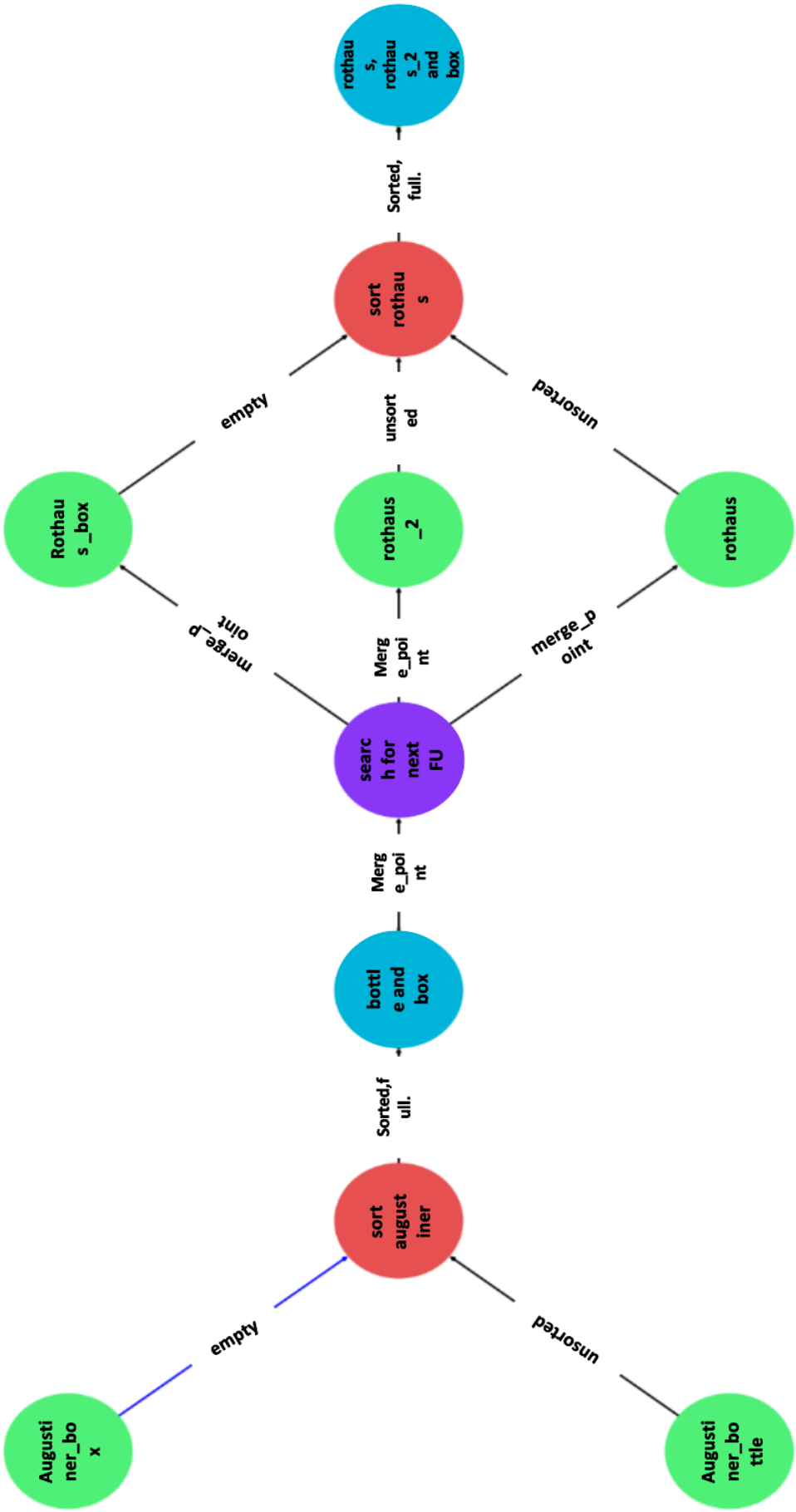

Figure 3.4: The TreeNode includes all functional units.

3.2 MoveIt

We started this part by using the “MoveIt Setup Assistant” graphical interface to generate a Semantic Robot Description Format (SRDF) for the Panda robot arm. We first needed to install the Franka description package, which includes the Universal Robot Description Format (URDF) of the Panda robot. After that, we used the Panda URDF to generate the self-collision matrix, add the virtual joints, add planning groups (panda arm and panda hand) and create the end effectors in the robot hand. Finally, we got the configuration files, which we use to run and control the robot in RViz.

Since we needed to write the implementation only in python and most of the MoveIt documentation were only written in C++, we had to convert all required parts from C++ to python.

At the beginning of implementation, the moveIt_commander“ and ”rospy_node“ had to be initialized, then we used the MoveIt commander to instantiate the robot commander and move group commander. These two commanders allowed us to change the joints state and choose which move group should run in every task.

We used the Planning scene interface, which allowed us to set and update the surrounding environment. To create the required objects in our simulation, we used the FOON to set the objects in the planning scene interfaces. Then we started the initial movement planning by implementing a function that sets the robot arm to a slightly better configuration. This start position enables the robot to move in all directions, most simply. The robot moves to this start position every time after finishing any motion to plan for the next movement.
The two essential movements for our tasks were the pick and place functions, including the open gripper and close gripper functions. The pick and place functions were implemented by using the Grasp class from MoveIt. This Grasp class needs the position of the target object and the position of the end place. Therefore, the Grasp class requires the quaternion to track and apply the 3D rotation. The quaternion has four components (x, y, z, w), which are used to represent an object’s attitude. To calculate the quaternion units, we used the function ” quaternion_from_euler“ to convert the Euler angels to their corresponding unit quaternion. Because of that, we had to calculate the Euler angles (roll, pitch, and yaw) for every movement to pick or place an object. The Euler angles are presented as shown in Figure 3.5.

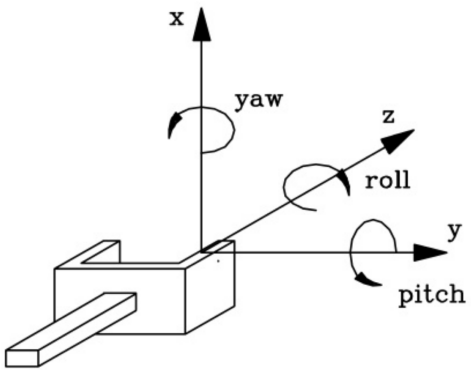

Figure 3.5: Robot hand with yaw, pitch and roll rotations.

To complete the pick and place functions, we had to implement the gripper functions used to open and close the hand joints during the pick and place processes. It is important to know that all required information is given from the FOON. The FOON will control which function is needed in every step and provide the function the targeted object information to accomplish the task.

To get a better understanding of the code structure and functionality, we illustrated the workflow in Figure 3.6 and an example from our experiment in Figure 3.7.

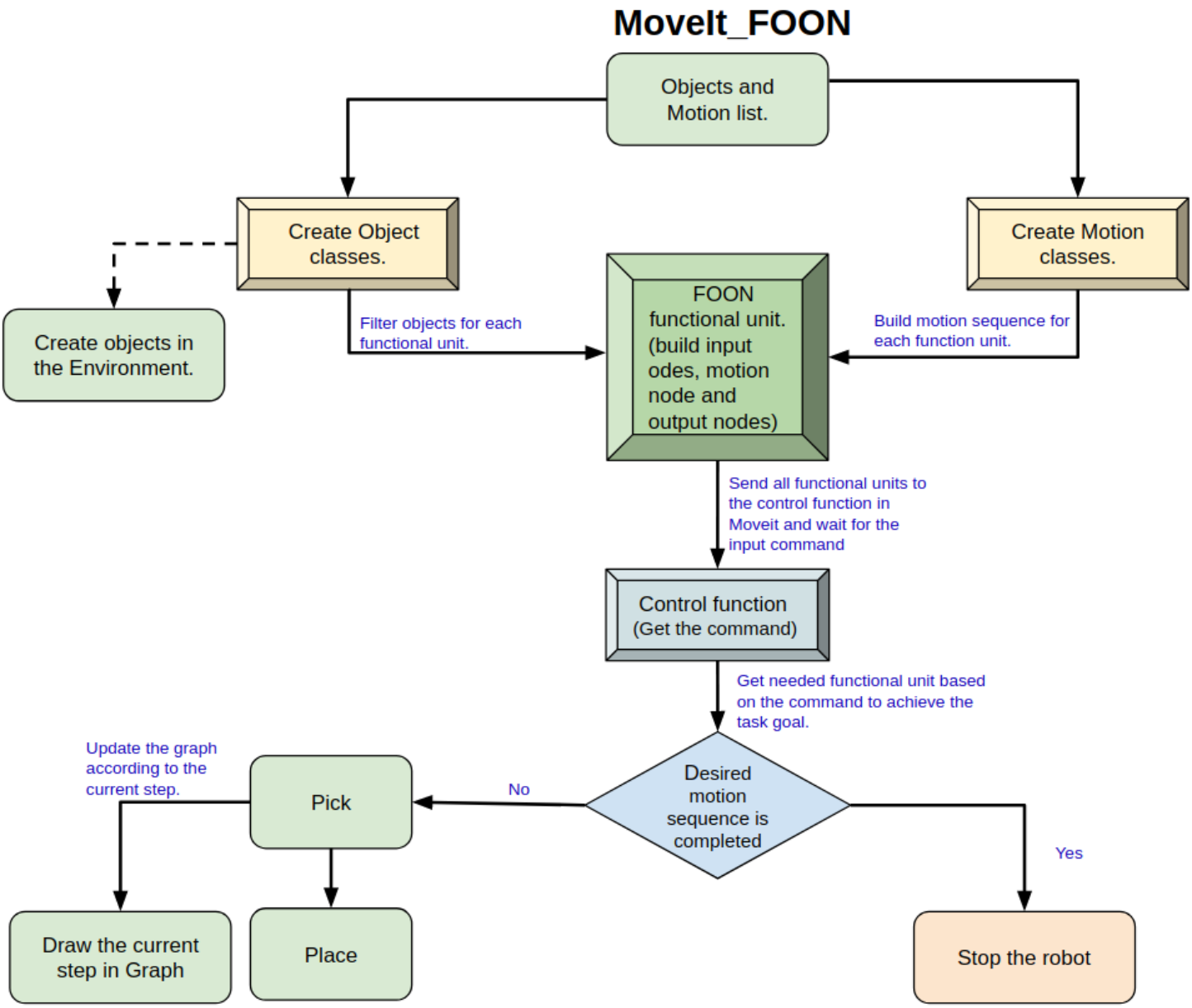

Figure 3.6: Code structure.

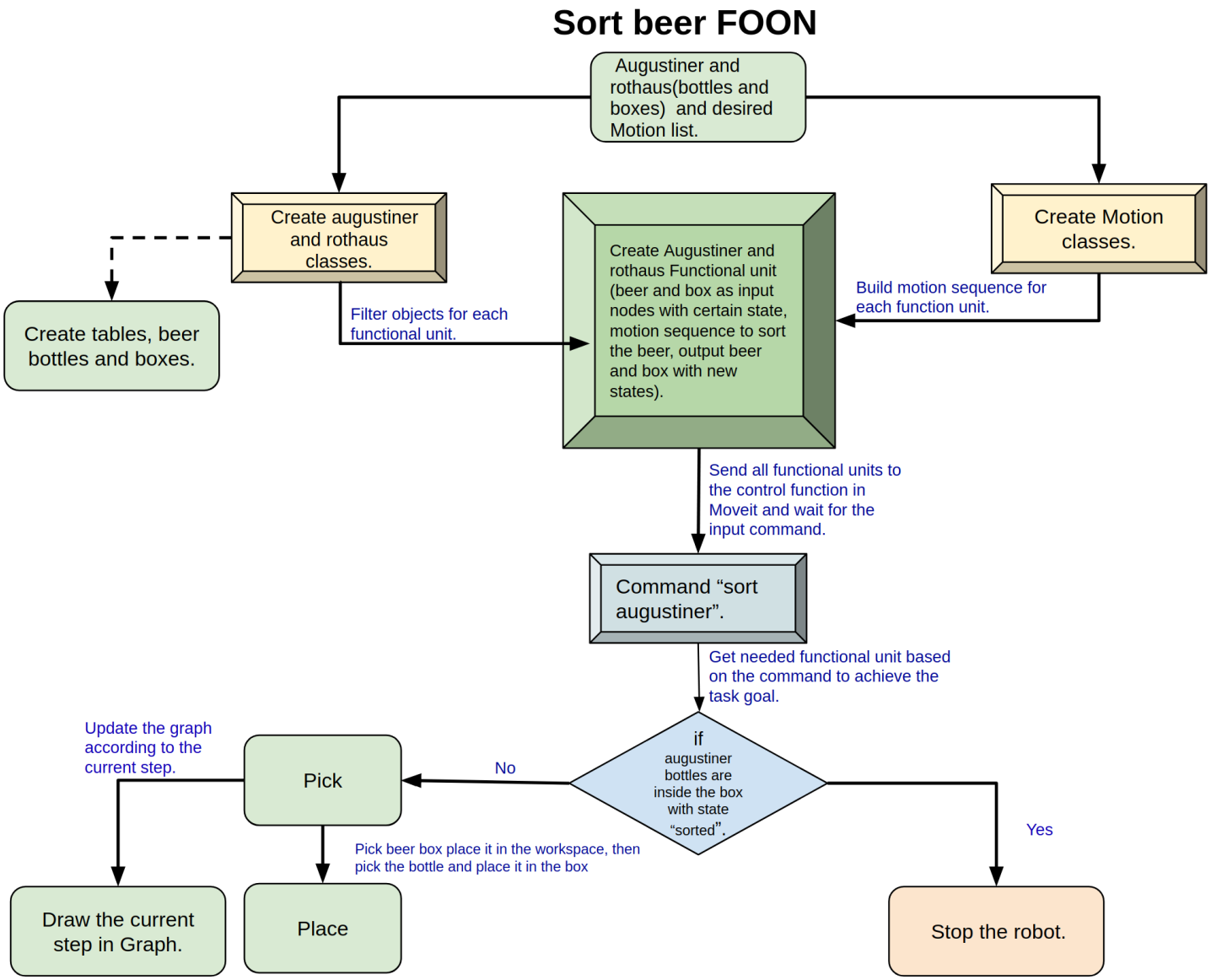

Figure 3.7: Example from the experiment.

4 Conclusion

The project aimed to present the functional object-oriented network (FOON) and expand its use in industry 4.0. We used the FOON to represent manipulation tasks to connect interactive objects with their functional motions. The FOON has a great value for learning manipulation tasks and understanding human activities. The fundamental unit of the FOON is the functional unit, which has the same logical structure as the human brain. The functional unit structure includes three main parts: the input objects with their current state, the intended manipulation tasks which achieve the target goal, and the output objects with their new state caused by manipulation.

We developed an approach to construct functional units using a list of different objects. The information of these objects was extracted by a human user to avoid the complexity of the object, state, and motion recognition. The object list was used to build the functional unit with the help of a filtering algorithm using the object's category and its possible states. The created functional units were merged and saved into the FOON. These functional units, which include manipulation knowledge, can be retrieved from the FOON by a given manipulation goal using a searching algorithm.
In FOON, the motion nodes are represented as a combination of motion harmonics. The motion nodes can be utilized in different environments with different objects by changing the values of the input parameters.

In our experiment, we proposed two sorts of beer bottles and their own boxes. These objects build two functional units using the filtering algorithm and merge them together to save them in the FOON. The FOON uses the input command to search for the desired functional unit and sort the targeted beer type in its box. The FOON can also sort all the beer bottles in its environment, dependent on the input command. Our motion nodes consist of pick and place functions, which we used in both functional units with different parameter values. We calculated the Euler angles for every object manually to optimize the robot arm movement. We used the robot arm "Franka Emika Panda" to achieve the manipulation tasks.

In the future, we plan to improve the network structure to work dynamically. The network should compare the exciting objects in the FOON and the new objects and be able to extrapolate the category of the new objects and create their possible functional units dynamically. Furthermore, we aspire to implement a vision recognition system to find the objects' information directly from the surrounding environment and learn the human behavior using the movement recognition algorithms.

Bibliography

[1] K. Balasingham. Industry 4.0: Securing the Future for German Manufacturing Companies. *School of Management and Governance Business Administration*, 11(2):15, 2016. URL: http://essay.utwente.nl/70665/1/Balasingham_BA_MA.pdf.

[2] M. Tenorth and M. Beetz. KnowRob: A knowledge processing infrastructure for cognition-enabled robots. *International Journal of Robotics Research*, 32(5):566–590, 2013. doi:10.1177/0278364913481635.

[3] A. G. Gonzalez, M. V. Alves, G. S. Viana, L. K. Carvalho, and J. C. Basilio. Supervisory Control-Based Navigation Architecture: A New Framework for Autonomous Robots in Industry 4.0 Environments. *IEEE Transactions on Industrial Informatics*, 14(4):1732–1743, 2018. doi:10.1109/TII.2017.2788079.

[4] D. Paulius, Y. Huang, R. Milton, W. D. Buchanan, J. Sam, and Y. Sun. Functional object-oriented network for manipulation learning. *IEEE International Conference on Intelligent Robots and Systems*, 2016-Novem:2655–2662, 2016. arXiv:1902.01537, doi:10.1109/IROS.2016.7759413.

[5] D. Paulius, A. B. Jelodar, and Y. Sun. Functional Object-Oriented Network: Construction Expansion. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 5935–5941, 2018. arXiv:1807.02189, doi:10.1109/ICRA.2018.8460200.

[6] D. Paulius, K. S. Pei Dong, and Y. Sun. Functional object-oriented network: Considering robot's capability in human-robot collaboration. *arXiv*, 2019. arXiv:1905.00502.

[7] C. Jaehne. Franka Emika Panda, 2019. (accessed 20/12/2020). URL: https://github.com/frankaemika/franka_ros.

[8] D. Coleman, I. Sucan, S. Chitta, and N. Correll. Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study. pages 1–14, 2014. URL: http://arxiv.org/abs/1404.3785, arXiv:1404.3785.

[9] I. A. Chitta and Sachin. MoveIt, 2019. (accessed 01/12/2020). URL: https://moveit.ros.org/.

[10] W. Woodall. RViz ROS, 2018. (accessed 20/11/2020). URL: http://wiki.ros.org/rviz.

[11] J. Pan, S. Chitta, and D. Manocha. FCL: A general purpose library for collision and proximity queries. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 3859–3866, 2012. doi:10.1109/ICRA.2012.6225337.

[12] A. Hagberg, D. Schult, and P. Swart. NetworkX Reference (Python). *Python package*, page 464, 2011.