Program Structure of Robot Data Collection Framework Using ROS and Gazebo

1. Directory Structure

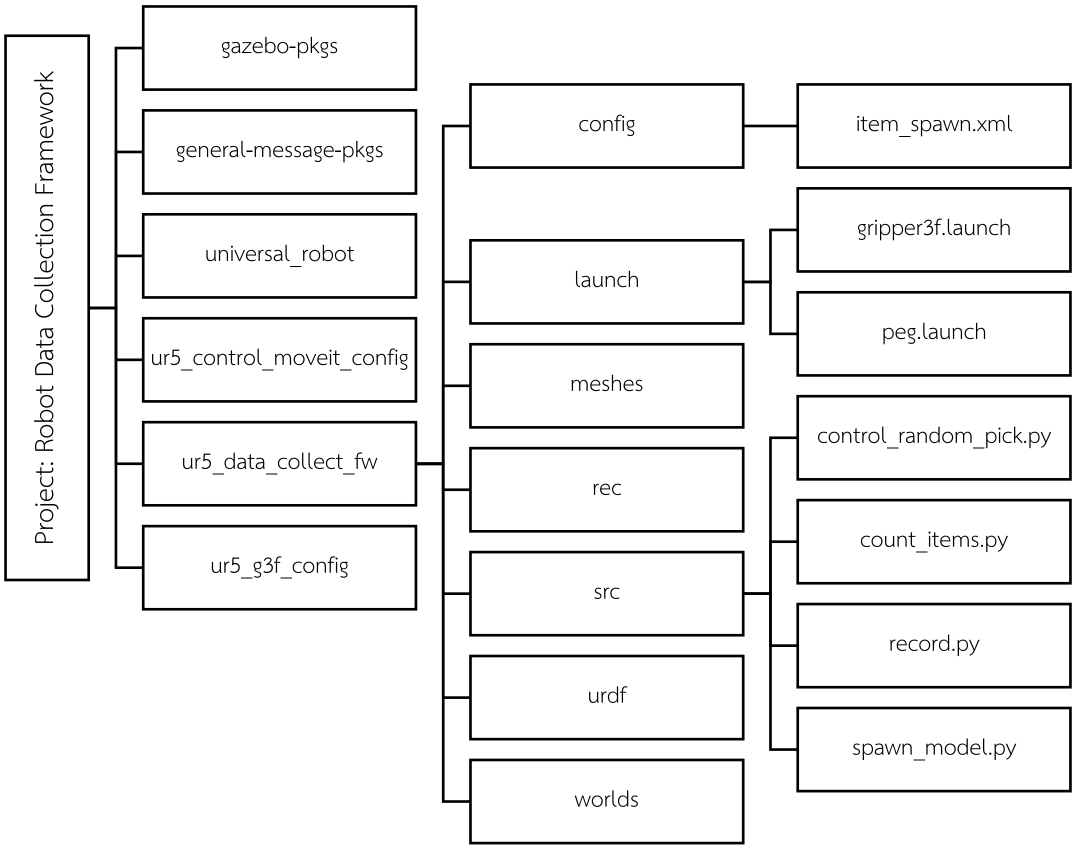


Figure 1. Directory structure for significant files and folder in project: robot data collection framework

From the figure above, there are 6 ROS packages in the system. The first two, gazebo-pkgs and general-message-pkgs, are for gripper plugin, which allows gripper to grasp and pick something up in Gazebo. Universal_robot is the package which contains UR5 files, including meshes and its basic controllers. The two package which end with “config” is generated from “MoveIt setup assistant” and is used to generate controller that allow MoveIt to control the robot. Last, ur5_data_collect_fw is the package which will be mainly focus as many important files are kept here; for example, launch file of data collection framework in folder launch, e.g., gripper3f.launch allows user to simulate task to move gripper to pick up objects in the system and place them in the bin, and peg.launch allow user to use multiple controllers (MoveIt and compliance) to put peg in hole. Furthermore, there are python files to control, record, and spawn items in folder src, and URDF files of robot combining with end-effector and other objects in the system.

2. Node Structure

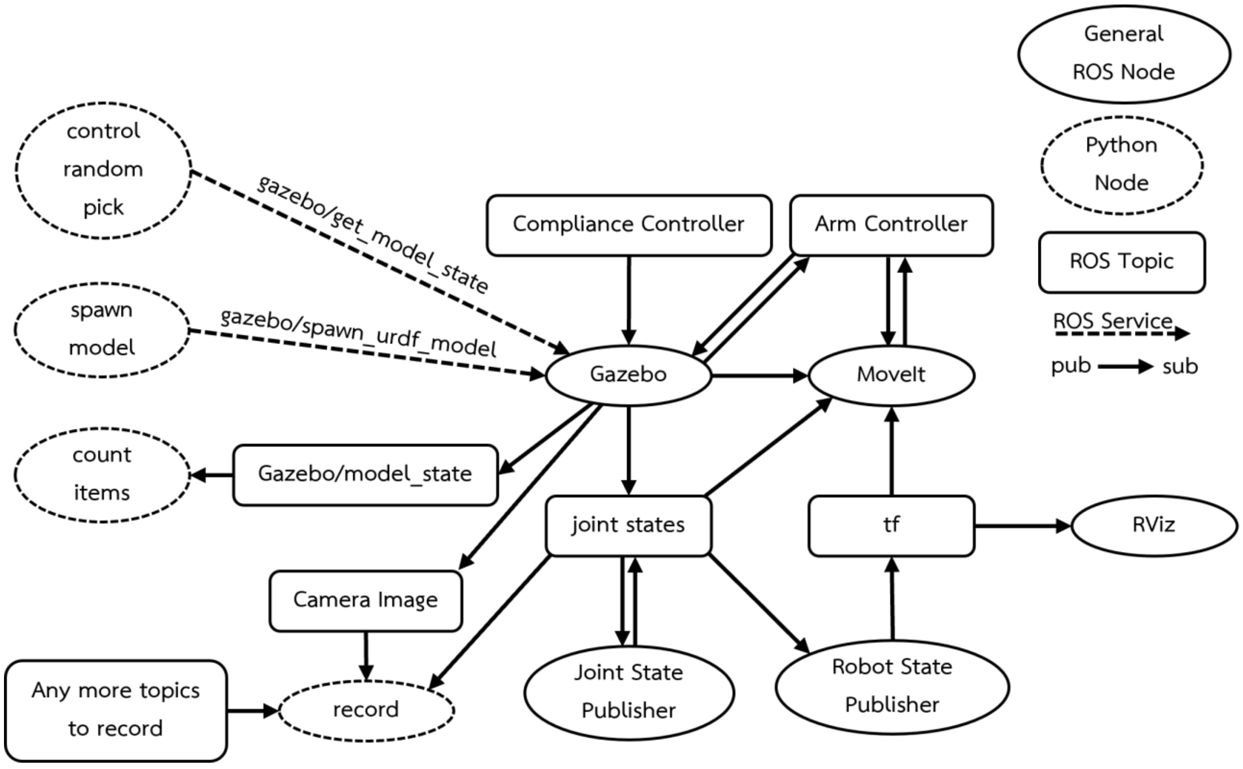


Figure 2. Node structure for significant topics and services transferred in the system

According to Figure 2, there are 9 major nodes in this system. Five nodes in the pictures are general ROS node, which means it comes with basic ROS packages and can be searched for further information. However, the basic usage of these programs are that joint state publisher and robot state publisher are used to auto control and publish joint states and tf so that every node will see the same position of the robot and allow human to understand current stage of the robot. MoveIt is one of the controllers of the robot, which can be used to insert input to set a goal state of the robot and control it to that state, by using feed-forward controller. Gazebo and RViz are node to display data in the system. The other four nodes, which are called python node in this place, are developed though out this project and will be explain their details later.

3. ROS parameter in created node

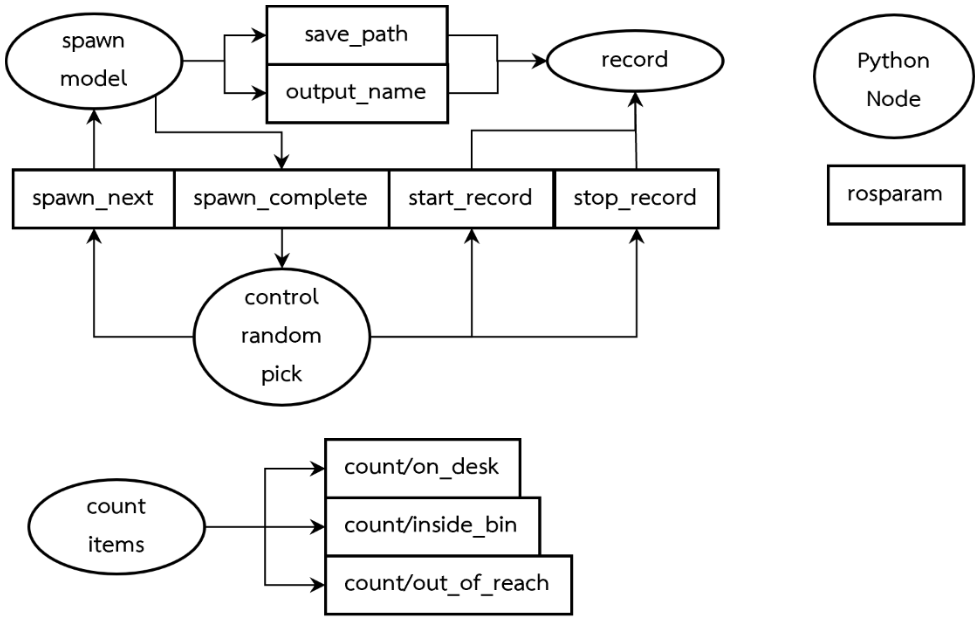


Figure 3. Node structure for created rosparam which are transferred in the system

In figure 3, there are four nodes, which are python node, and they use rosparam to contact to the others. The principle of rosparam is that it is parameters stored in the ROS server so that one node can "set" the value of the parameters, and then other node can "get" the value of these parameters. There are currently 9 rosparam in the system, which all use namespace "asl", for instance save_path is /asl/save_path on the server. Details of each parameter will be in the next part, together with how each of this node works.

4. Details of python node

4.1 spawn model

A node which reads xml file in specific condition, parse it and autofill some missing argument to get URDF filetype of each item, then call rosservice /gazebo/spawn_urdf_model to spawn items in Gazebo

Table 4.1. Arguments of spawn_model.py

Arguments (Optional)	Definition	Default Value
- open	path to xml file or folder	Path to ur5_data_collect_fw/config/item_spawn.xml
- save_path	path to save generated complete xml file	Path to ur5_data_collect_fw/rec/
- world_items	items which will not be deleted	['ground_plane', 'kinect', 'kinect_pilar', 'robot']

Rosparam list:

- asl/save_path (output)

  Path to save generated xml file from this node, together with other files from another node

- asl/output_name (output)

  Initial name of output. For example, if set to 1, the output xml file of this node will be "1.xml".

- asl/spawn_complete (output)

  When set to True, tells other node that all items in current xml file have already spawned.

- asl/spawn_next (input, output)

  When set to True, this node will delete previous spawn models, and spawn the new one.

Example and argument of input xml file

```
<?xml version="1.0" encoding="UTF-8"?>

<root>
   <models>
      <model id ="cylinder" type="cylinder" color="red" amount="3" mass="0.1"
   radius="0.038" length="0.1" mu1="2.0" mu2="2.0" />
      <model id ="box" type="box" color="blue" amount="1" mass="0.1"
   box_size="0.05 0.05 0.1"/>
      <model id="box_sq" filename="/urdf/box_sq.urdf" xyz="0.0 0.0 1.0"/>
   </models>
</root>
```

Table 4.1.2 Arguments of input xml file of spawn_model.py

Arguments (Optional)	Definition
id	Name of the item
type	Type of the item, e.g., box, cylinder, sphere, urdf
color	Color of the item shown in Gazebo
xyz	Position of the item in format “x y z”, e.g., “1 1 1”
filename	Path to existed URDF file
amount	Number of this item spawn in Gazebo
mass	Mass of the item
box_size	Size of this item if type is “box” in format “x y z” for its length in each direction, e.g., “1 1 1”
radius	Radius of this item if type is “cylinder” or “sphere”
length	Length or height of this item if type is “cylinder”
mu1	Friction coefficients μ for the principal contact directions along the contact surface as defined by the Open Dynamics Engine (ODE)
mu2	

** For those arguments which might be missing, the program will autofill with default value, including random xyz position. However, if there are invalid input type of the arguments of an item, that item will not spawn, but the others will spawn normally. Then, all the item which are completely spawned will be written to output xml file.

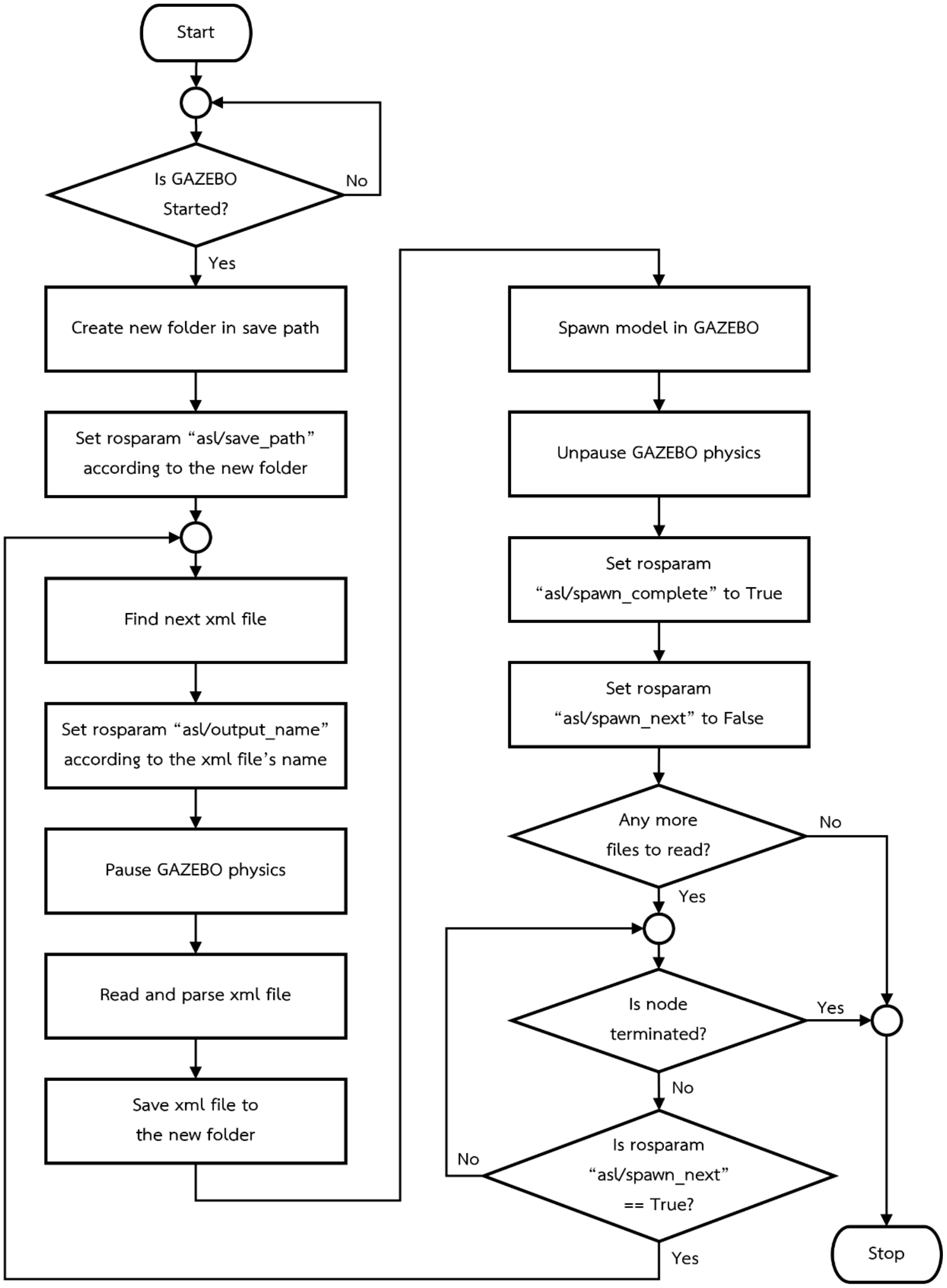

Start
Is GAZEBO Started?
No
Yes
Create new folder in save path
Set rosparam "asl/save_path" according to the new folder
Find next xml file
Set rosparam "asl/output_name" according to the xml file's name
Pause GAZEBO physics
Read and parse xml file
Save xml file to the new folder
Spawn model in GAZEBO
Unpause GAZEBO physics
Set rosparam "asl/spawn_complete" to True
Set rosparam "asl/spawn_next" to False
Any more files to read?
No
Yes
Is node terminated?
Yes
No
Is rosparam "asl/spawn_next" == True?
No
Yes
Stop

Figure 4.1. Flowchart of spawn_model.py

4.2 record

A node which records rostopics from ROS server to different file types, which are bag file, video files as type avi or mp4, and csv of joint states.

Table 4.2. Arguments of record.py

Arguments (Optional)	Definition	Default Value
- topic	list of topics to record	None, but will get all topic later
- joint_topic	list of sensor_msgs/JointState to record as csv	Empty List
- video_topic	list of sensor_msgs/Image topic to record as other video format	Empty List
- video_type	video recording format -> [avi, mp4]	mp4
- save_path	place to save file	Path to ur5_data_collect_fw/bag/
- compress	compress bag file or not -> [No, lz4, bz2]	lz4
- unique	extract topic which already save as other type from bag file	False
- no_bag	record to bag file or not	False

Rosparam list:

- asl/save_path (input)

  Path to save generated record file from this node, together with other files from another node

- asl/output_name (input)

  Initial name of output, e.g., if set to 1, the output bag will be "1_{datetime}.bag"

- asl/start_record (input, output)

  When set to True, this node will start recording

- asl/stop_record (input, output)

  When set to True, this node will stop recording

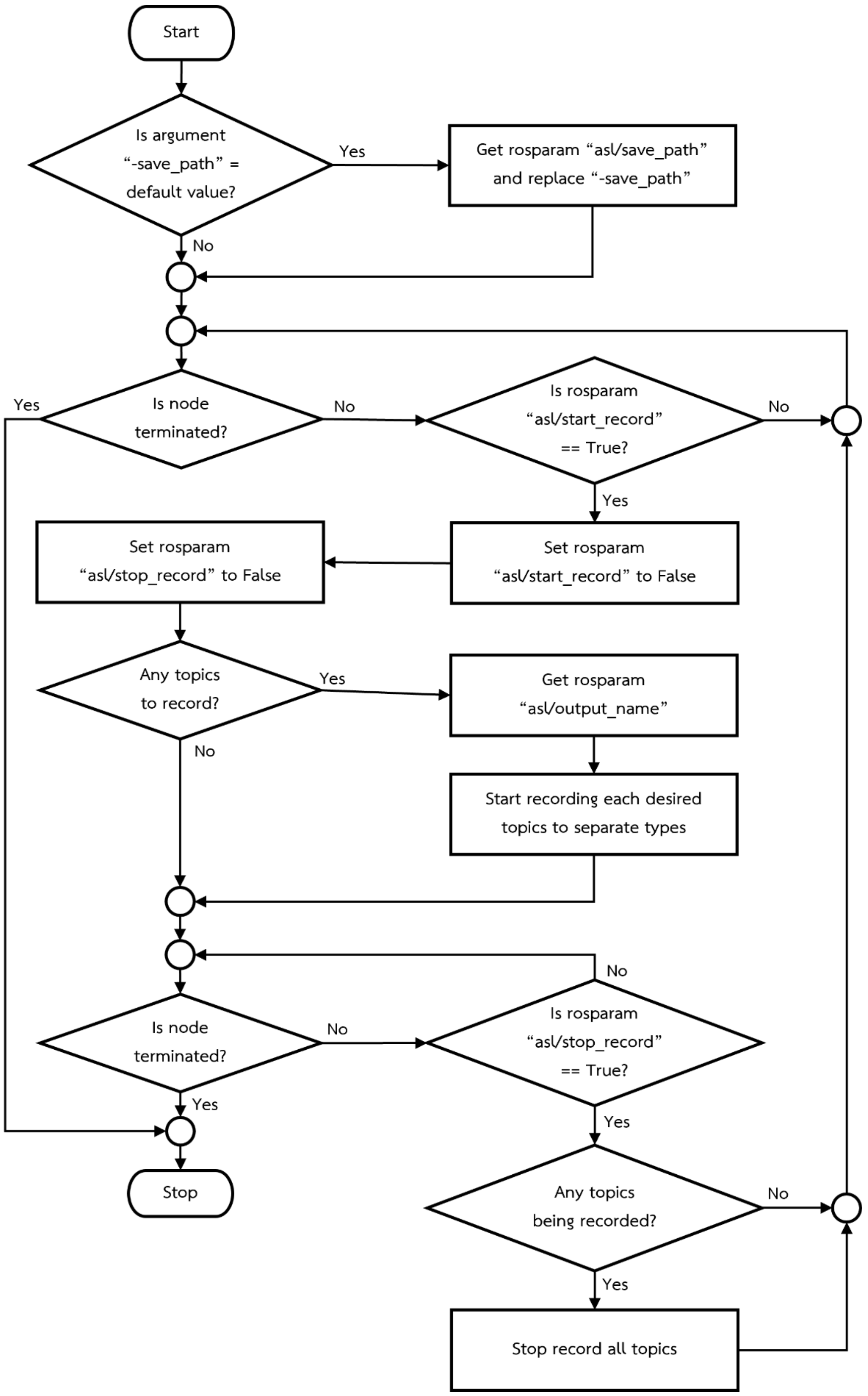

Start
Is argument "-save_path" = default value?
Yes
Get rosparam "asl/save_path" and replace "-save_path"
No
Yes
Is node terminated?
No
Is rosparam "asl/start_record" == True?
No
Yes
Set rosparam "asl/stop_record" to False
Set rosparam "asl/start_record" to False
Any topics to record?
Yes
Get rosparam "asl/output_name"
No
Start recording each desired topics to separate types
No
Is node terminated?
No
Is rosparam "asl/stop_record" == True?
Yes
Yes
Stop
Any topics being recorded?
No
Yes
Stop record all topics

Figure 4.2. Flowchart of record.py

4.3 control random pick

A node which uses MoveIt to control manipulator and gripper to try once to pick up each item in the system, excluding robot and camera, then put it in the bin.

Rosparam list:

- asl/spawn_complete (input)

  When set to True, this node starts using MoveIt to pick and place items

- asl/spawn_next (output)

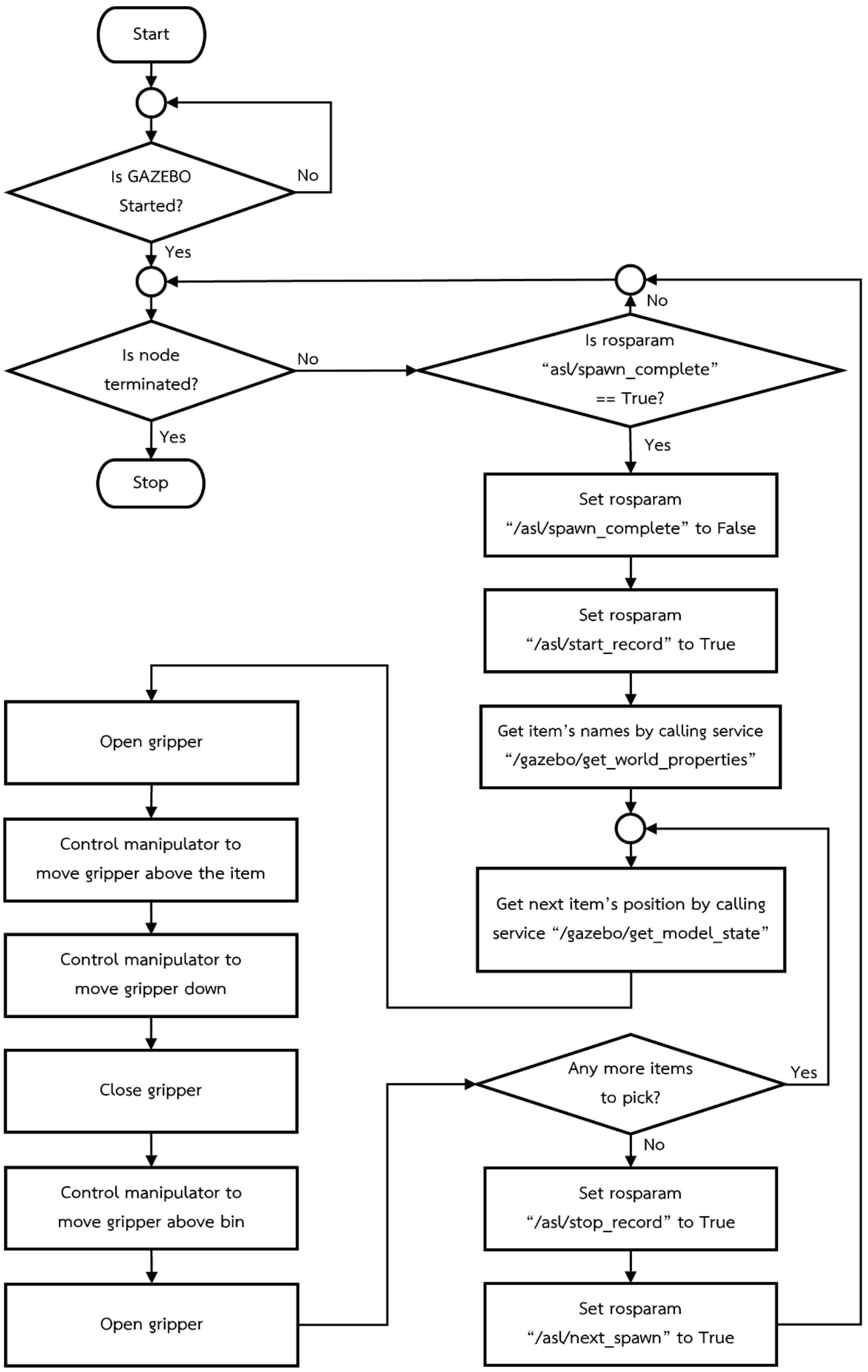
  Use to tell other node to remove current items and spawn new items from next config file

- asl/start_record (output)

  Use to tell other node to start recording

- asl/stop_record (output)

  Use to tell other node to stop recording



Start
Is GAZEBO Started?
No
Yes
Is node terminated?
No
Yes
Stop
Is rosparam “asl/spawn_complete” == True?
No
Yes
Set rosparam “/asl/spawn_complete” to False
Set rosparam “/asl/start_record” to True
Get item’s names by calling service “/gazebo/get_world_properties”
Get next item’s position by calling service “/gazebo/get_model_state”
Open gripper
Control manipulator to move gripper above the item
Control manipulator to move gripper down
Close gripper
Control manipulator to move gripper above bin
Open gripper
Any more items to pick?
Yes
No
Set rosparam “/asl/stop_record” to True
Set rosparam “/asl/next_spawn” to True

Figure 4.3. Flowchart of control_random_pick.py

4.4 count items

A node which counts items on each place which are on table, inside bin, and out of reach, then publish to separate rosparam.

Rosparam list:

- /asl/count/on_desk (output)

  Number of items on the desk, which is located higher than desk's height

- /asl/count/inside_bin (output)

  Number of items inside the bin, which is located inside 2D base of the bin

- /asl/count/out_of_reach (output)

  Number of items left in the system which is not the two above

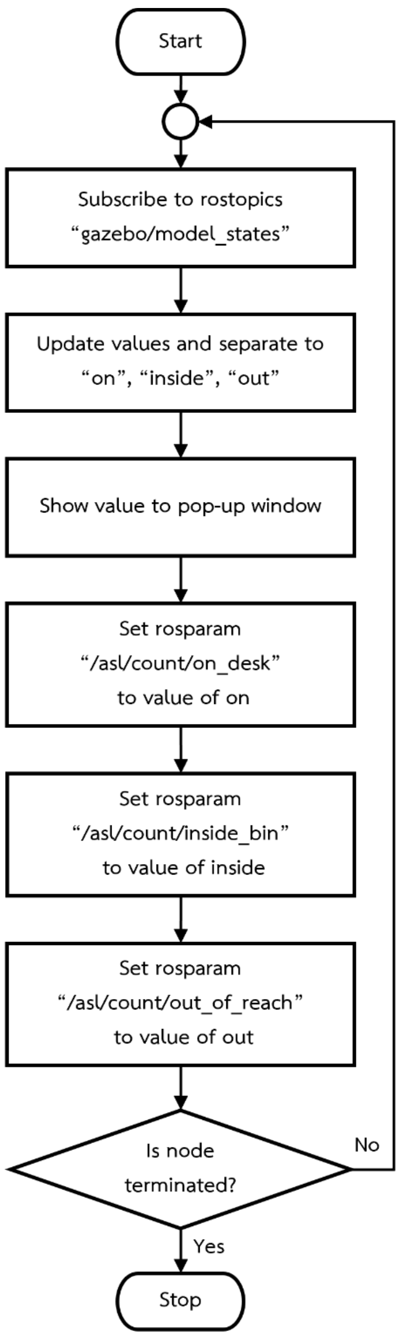

Start
Subscribe to rostopics "gazebo/model_states"
Update values and separate to "on", "inside", "out"
Show value to pop-up window
Set rosparam "/asl/count/on_desk" to value of on
Set rosparam "/asl/count/inside_bin" to value of inside
Set rosparam "/asl/count/out_of_reach" to value of out
Is node terminated?
No
Yes
Stop

Figure 4.4. Flowchart of record.py