Program Structure of Robot Data Collection Framework Using ROS and Gazebo

1. Directory Structure

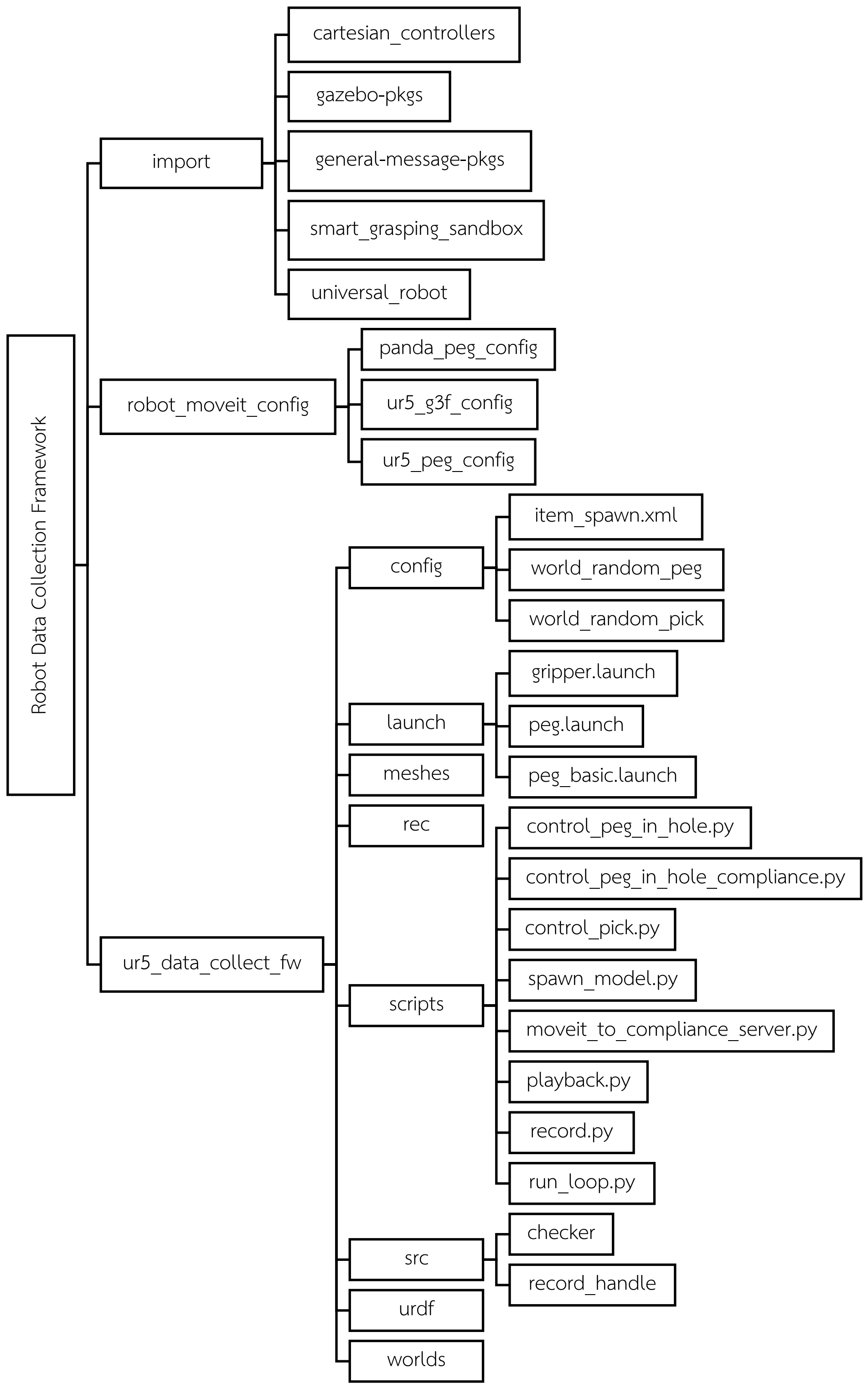


Figure 1. Structure of significant files and directory in robot data collection framework

From the figure above, there are 3 types of ROS packages in the system. The first one is “import”, which means it is created in another repository, e.g., cartesian_controllers is the plugin to use cartesian controller, gazebo-pkgs and general-message-pkgs are for gripper plugin, which allows gripper to grasp and pick something up in Gazebo, smart_grasping_sandbox is the package containing mesh files for the gripper, universal_robot is the package which contains UR5 files, including meshes and its basic controllers. Next, “robot_moveit_config” is generated from “MoveIt setup assistant” and is used to generate controller that allow MoveIt to control the robot.

The last type of package is “ur5_data_collect_fw”, which is the main package of this project as many important files are kept here. The details are as follow.

config	keeps config file of the system, including controller config and world items config as in item_spawn.xml and in folder random_peg_config and random_pick_config
launch	keeps launch file for start simulating any tasks; peg in hole or pick and place
meshes	keeps mesh file in format stl or dae which is used later to create urdf file
rec	create later to save recoding file from the system
scripts	keeps python program which has action in itself
src	keeps python module, which only has class and need to be used in another program
urdf	keeps urdf file, which is a format to combine mesh to create robot in ROS
worlds	keeps Gazebo world file. Static items and camera sensor in the system are specified here

2. Node Structure

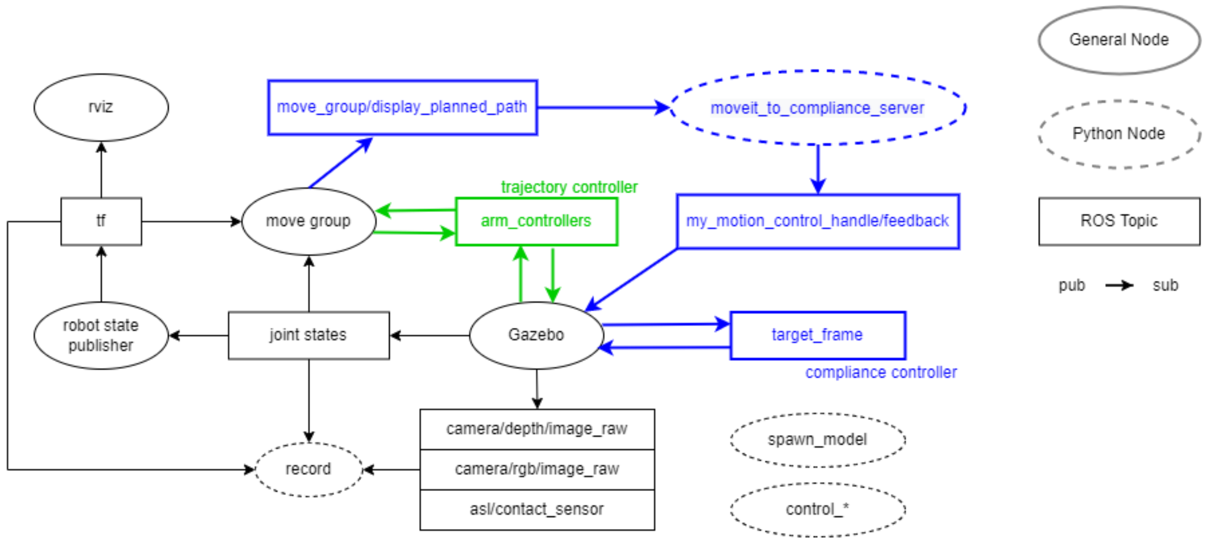


Figure 2. Node structure for significant topics and services transferred in the system

According to Figure 2, there are two types of nodes in the system. The first one is python node, which are created in this project and will be explained later. The second one is general node in ROS system, which means it comes with basic ROS packages and can be searched for further information. However, the basic usage of these programs are that Gazebo is the simulation program which used ODE method to calculate how objects in the system move, RViz is the visualizing program which receives any topics in the system and make it visualizable for human in 2D or 3D; for example, receiving raw data of tf and changing it to the visual of the current position of the robot, Move Group is the package from MoveIt, which is used to do path planning and make commands to the controllers, so that robot can move to the desired position, and robot state controller is a node receiving data from joint_states, which is the value of the angular position of each joints, then transform it into cartesian position in tf.

Another thing which can be seen from Figure 2 is the node related to the controllers used. For joint trajectory controller, move_group would send command, which is path detail, directly to arm_controller, the name of joint trajectory controller in this simulation. Then, the robot could move according to that path. However, for cartesian compliance controller, there are more nodes and topics involved, as shown in blue path. After using move_group to calculate path, the command is not sent to the controller directly, but it is transformed from angular position to cartesian position of the end effector before publishing to my_motion_controller_handle, which is the marker in RViz, to command the controller.

3. ROS parameter in python node

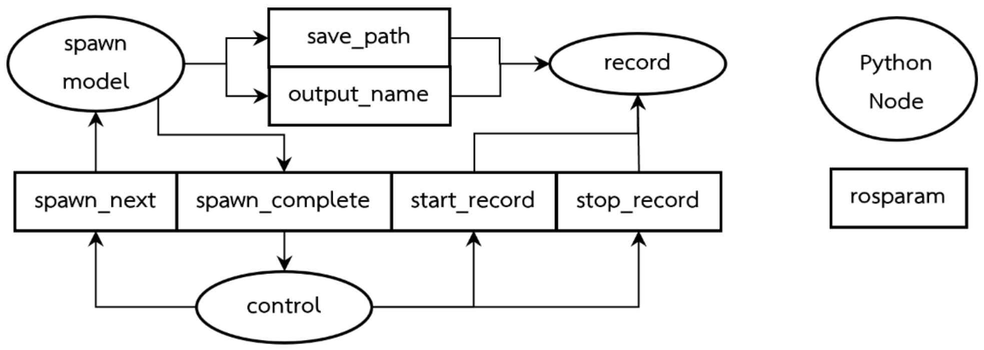


Figure 3. Node structure for created rosparam which are transferred in the system

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 6 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 directory	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/
- fix_path	fix path to save generated complete xml file (no new folders)	-
- 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.

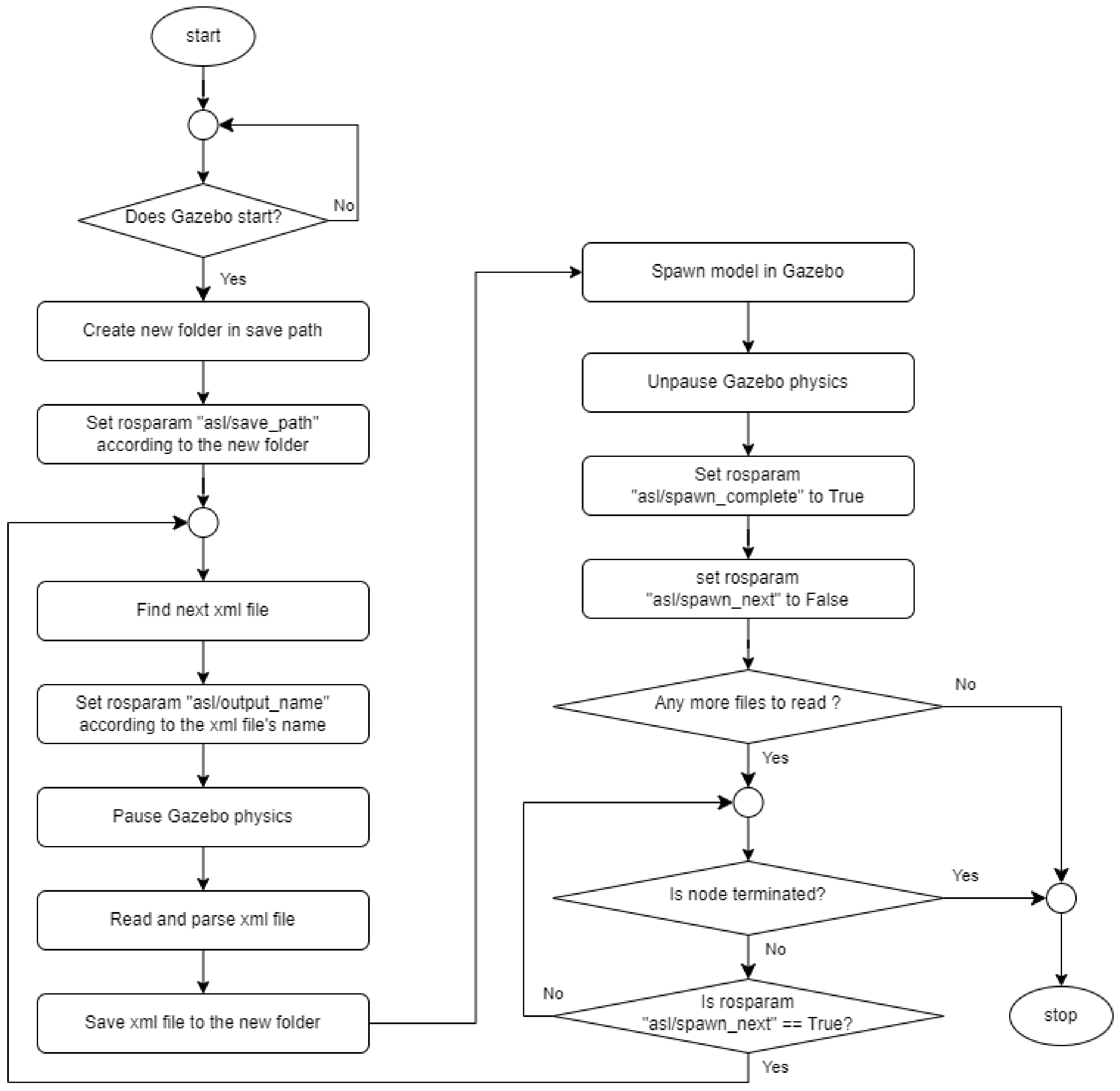

start
Does Gazebo start?
No
Yes
Create new folder in save path
Set rosparam "asl/save_path" according to the new folder
Find next xml file
Set rosparam "asl/output_name" according to the xml file's name
Pause Gazebo physics
Read and parse xml file
Save xml file to the new folder
Spawn model in Gazebo
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
-bag_topic	list of topics to record in bag, 'all' means record all topics	Empty list
-bag_compress	compress bag file or not -> [No, lz4, bz2]	lz4
-contact_topic	list of gazebo_msgs/ContactsState topic to record as csv	Empty list
-joint_topic	list of sensor_msgs/JointState to record as csv	Empty list
-pose_record	record pose using moveit or not	False
-pose_planning_group	moveit's planning group to get pose	Manipulator
-video_rgb_topic	a sensor_msgs/Image topic, rgb type, to record as other video format	“”
-video_depth_topic	a sensor_msgs/Image topic, depth type, to record as other video format	“”
-video_type	video recording format -> [avi, mp4]	mp4

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

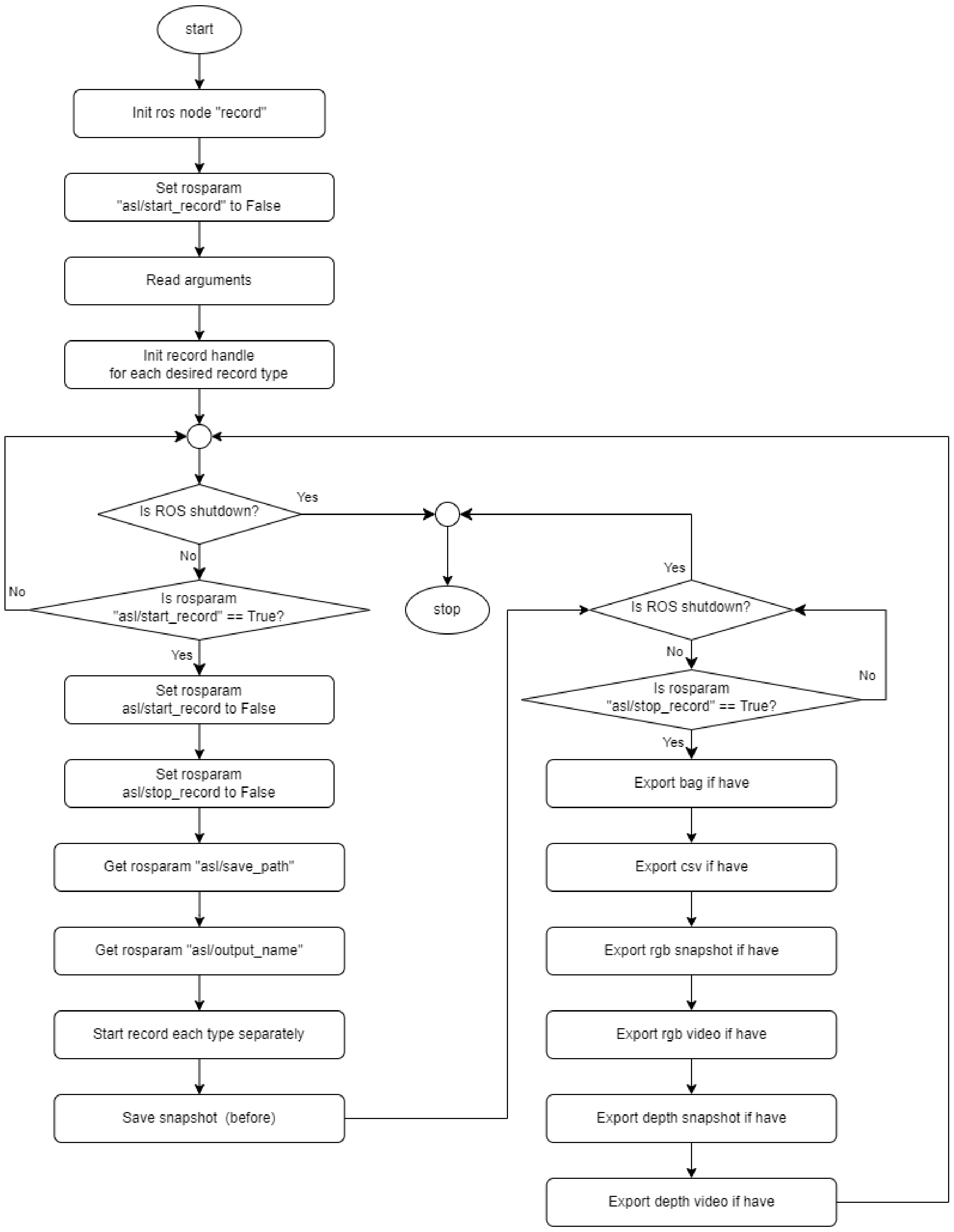

start
Init ros node "record"
Set rosparam "asl/start_record" to False
Read arguments
Init record handle for each desired record type
Is ROS shutdown?
Yes
No
Is rosparam "asl/start_record" == True?
No
Yes
Set rosparam asl/start_record to False
Set rosparam asl/stop_record to False
Get rosparam "asl/save_path"
Get rosparam "asl/output_name"
Start record each type separately
Save snapshot (before)
stop
Yes
Is ROS shutdown?
No
Is rosparam "asl/stop_record" == True?
No
Yes
Export bag if have
Export csv if have
Export rgb snapshot if have
Export rgb video if have
Export depth snapshot if have
Export depth video if have

Figure 4.2. Flowchart of record.py

4.3 control group

A node which uses MoveIt to control manipulator with peg or gripper to do the task given. There are three files included. The details are as follow.

1. control_peg_in_hole for instering peg in a square hole of a box using joint trajectory controller; this node use MoveIt planner (compute cartesian path; draw a line directly between two or more points) to control the robot to the destination
2. control_peg_in_hole_compliance for instering peg in a square hole of a box using cartesian compliance controller; this node use MoveIt planner (compute cartesian path) to calculate the path to the object, transform the path, which is the joint state trajectory, by using moveit_to_compliance_server.py to the path of the position of the end effector, then publish to the interactive marker named “motion control handle” which is linked with the cartesian compliance controller to control the robot
3. control_pick for picking and placing objects on the desk using joint trajectory controller; this node use MoveIt Planner (compute cartesian path) to calculate path to pick and place objects. The detail of the path is that the robot starts moving toward “home” position, then move above the object, move down, and grab the object, move up, go to the edge of the desk (to avoid collision with the robot itself), then move above the bin and release the object. Continue doing this until all objects on the desk is picked once.

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

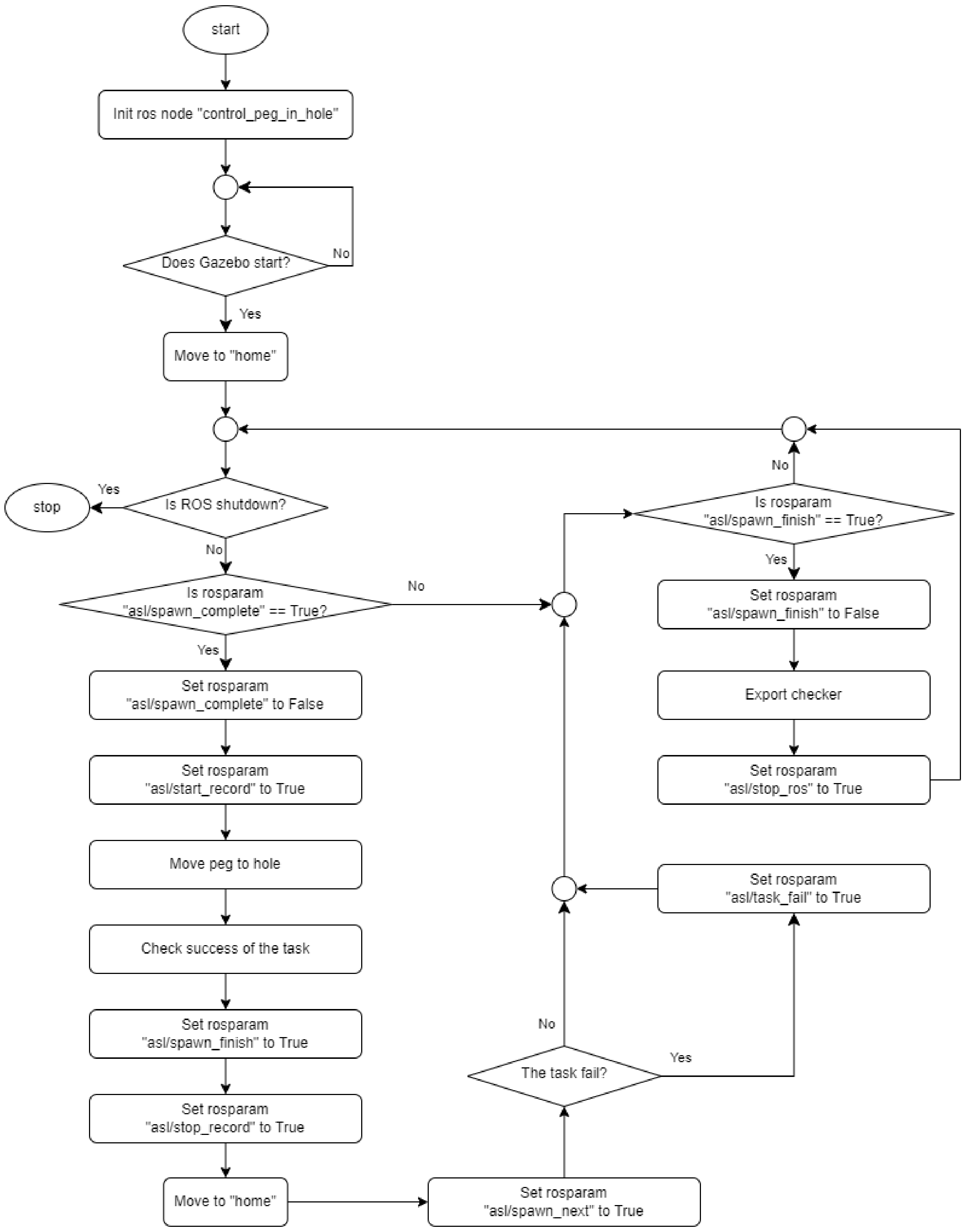

start
Init ros node "control_peg_in_hole"
Does Gazebo start?
No
Yes
Move to "home"
Is ROS shutdown?
Yes
stop
No
Is rosparam "asl/spawn_complete" == True?
No
Yes
Set rosparam "asl/spawn_complete" to False
Set rosparam "asl/start_record" to True
Move peg to hole
Check success of the task
Set rosparam "asl/spawn_finish" to True
Set rosparam "asl/stop_record" to True
Move to "home"
Set rosparam "asl/spawn_next" to True
The task fail?
No
Yes
Set rosparam "asl/task_fail" to True
Is rosparam "asl/spawn_finish" == True?
No
Yes
Set rosparam "asl/spawn_finish" to False
Export checker
Set rosparam "asl/stop_ros" to True

Figure 4.3. Flowchart of control_peg_in_hole.py

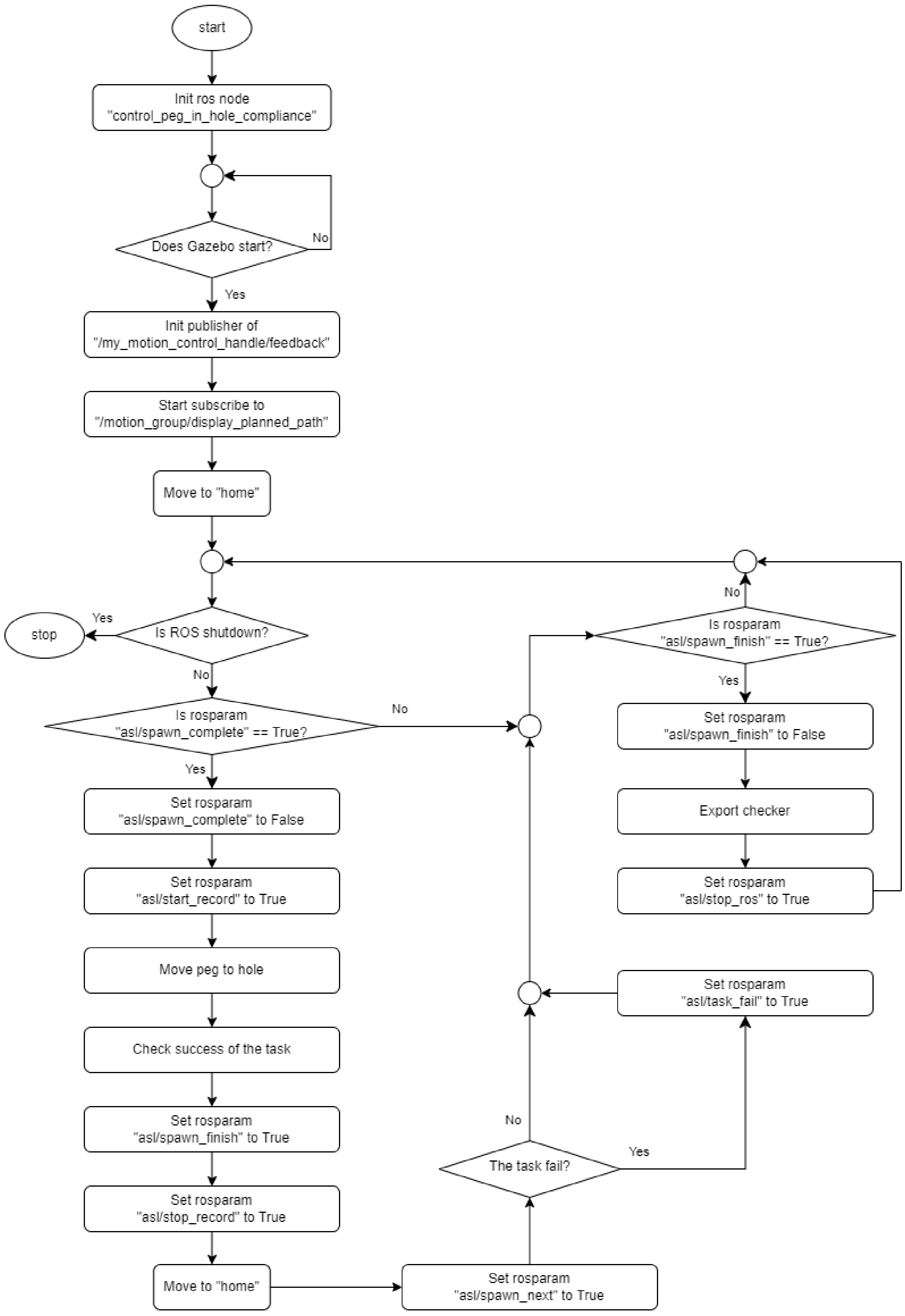

start
Init ros node "control_peg_in_hole_compliance"
Does Gazebo start?
No
Yes
Init publisher of "/my_motion_control_handle/feedback"
Start subscribe to "/motion_group/display_planned_path"
Move to "home"
Is ROS shutdown?
Yes
stop
No
Is rosparam "asl/spawn_complete" == True?
No
Yes
Set rosparam "asl/spawn_complete" to False
Set rosparam "asl/start_record" to True
Move peg to hole
Check success of the task
Set rosparam "asl/spawn_finish" to True
Set rosparam "asl/stop_record" to True
Move to "home"
Set rosparam "asl/spawn_next" to True
The task fail?
No
Yes
Set rosparam "asl/task_fail" to True
Is rosparam "asl/spawn_finish" == True?
No
Yes
Set rosparam "asl/spawn_finish" to False
Export checker
Set rosparam "asl/stop_ros" to True

Figure 4.4. Flowchart of control_peg_in_hole_compliance.py

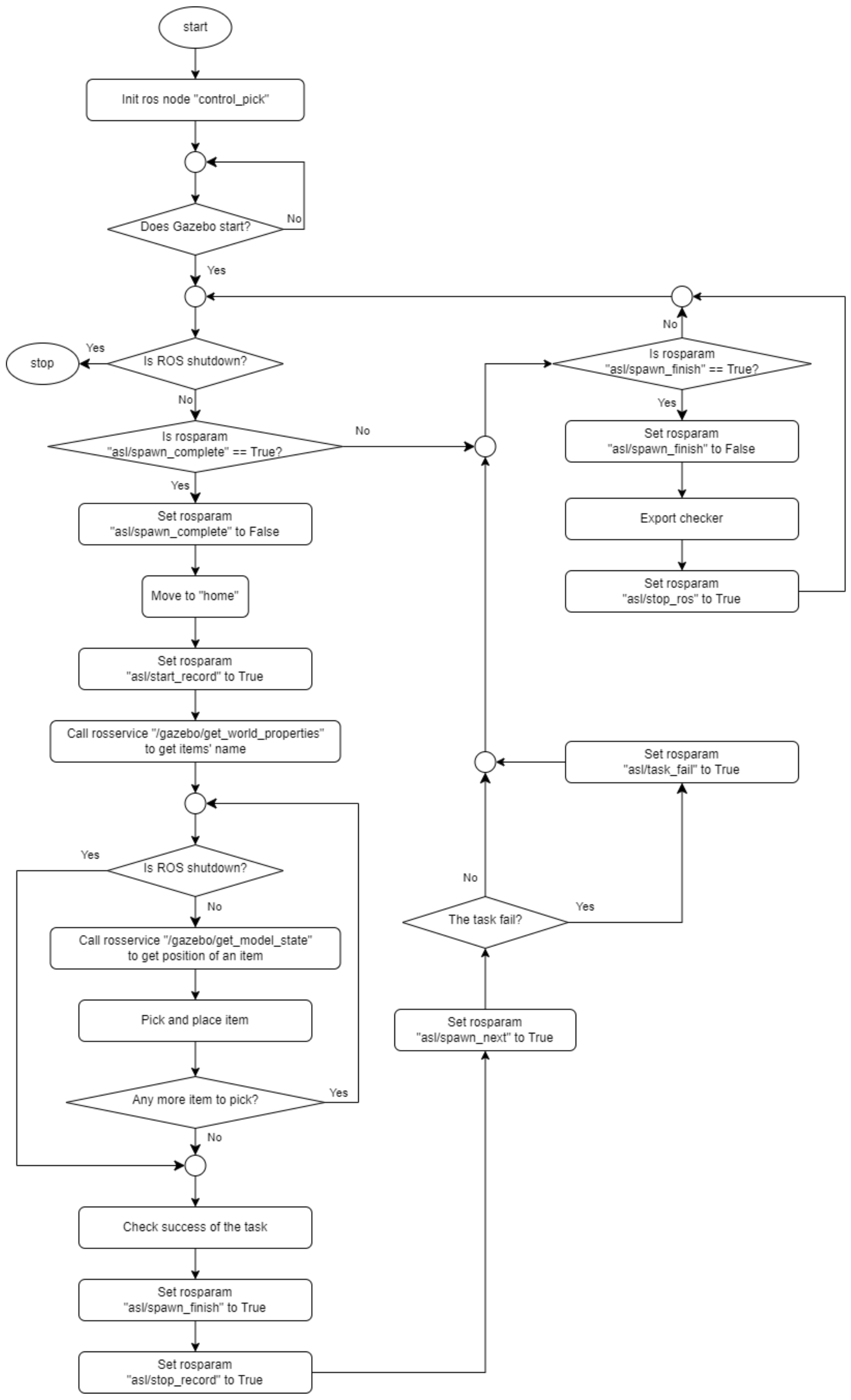

start
Init ros node "control_pick"
Does Gazebo start?
No
Yes
Is ROS shutdown?
Yes
stop
No
Is rosparam "asl/spawn_complete" == True?
No
Yes
Set rosparam "asl/spawn_complete" to False
Move to "home"
Set rosparam "asl/start_record" to True
Call rosservice "/gazebo/get_world_properties" to get items' name
Is ROS shutdown?
Yes
No
Call rosservice "/gazebo/get_model_state" to get position of an item
Pick and place item
Any more item to pick?
Yes
No
Check success of the task
Set rosparam "asl/spawn_finish" to True
Set rosparam "asl/stop_record" to True
Set rosparam "asl/spawn_next" to True
The task fail?
No
Yes
Set rosparam "asl/task_fail" to True
Is rosparam "asl/spawn_finish" == True?
No
Yes
Set rosparam "asl/spawn_finish" to False
Export checker
Set rosparam "asl/stop_ros" to True

Figure 4.5. Flowchart of control_pick.py

4.4 moveit_to_compliance_server

A node server to receive the path calculating by MoveIt in the topic "move_group/display_planned_path" and transform from joint state trajectory to cartesian path, then send this to the marker linking with the controller in the topic "my_motion_control_handle/feedback" to control the robot.

4.5 playback

A node server to receive data from bag files and send to control the robot. By using this node, user must record joint_states or rosmsg type sensor_msgs/JointState.msg as a bag file and then playback the file using command "rosbag play {filename}.bag --topics /joint_states /joint_states:=/joint_states_playback". This node will then subscribe to the topic, separate the data of the joint positions, then send each position to each controller in type joint position controller (need to be active in the system to be used) to move the robot according to the path in the bag.

5. Details of non-node python programs

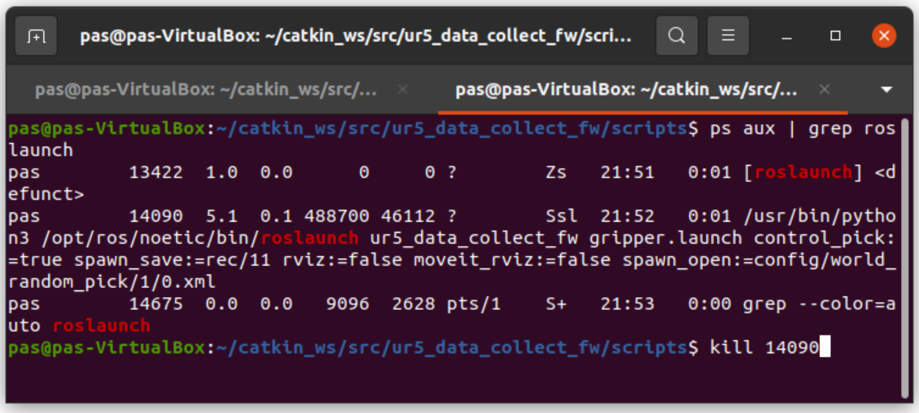
5.1 checker

A program used for checking the success of the task. This file is written to use as a class method. Currently, there are three conditions that can be checked. For the task of inserting peg, two conditions could be checked, which is to check the distance between the peg and the box with hole, and to check the force acting at the end effector. For the task of picking and placing objects, one condition could be checked, which is the number of items at each position in the system, e.g., *in the bin* for x and y less than 0.25 m from the middle of the bin, *on the desk* for the items which have z or their height more than the height of the desk (0.8 m), and *out of reach* or not in the first two conditions.

5.2 run_loop

A program used to handle the error of using single launch file for too long. In this file, launch file is used as a subprocess. Each subprocess is used to simulate only one items config file, and if while the simulation is running, this program will check the conditions of doing the task. If the task is finished completely, the system will select the next config file to spawn items; however, if the task is not finished, whether the task failed by collision of the objects in the system, the task failed by the failure of the controller, or the time using to simulate exceeded the limit, the system will reuse the same config file in the next round. These failures could occur at the maximum of five rounds before the system records the condition checking on the 6th round.

If want to terminate this program, as it uses subprocess, users must use *ctrl* + *z* in the current terminal, then the command *ps aux* should be used together with *grep roslaunch* for searching the process pid. After using *ps aux* | *grep roslaunch*, users must type in *kill* command to stop the current running process explicitly as shown in the picture below.


pas@pas-VirtualBox: ~/catkin_ws/src/ur5_data_collect_fw/scri...
pas@pas-VirtualBox: ~/catkin_ws/src/...
pas@pas-VirtualBox: ~/catkin_ws/src/...

```
pas@pas-VirtualBox:~/catkin_ws/src/ur5_data_collect_fw/scripts$ ps aux | grep ros
launch
pas        13422  1.0  0.0      0     0 ?        Zs   21:51   0:01 [roslaunch] <d
efunct>
pas        14090  5.1  0.1 488700 46112 ?        Ssl  21:52   0:01 /usr/bin/pytho
n3 /opt/ros/noetic/bin/roslaunch ur5_data_collect_fw gripper.launch control_pick:
=true spawn_save:=rec/11 rviz:=false moveit_rviz:=false spawn_open:=config/world_
random_pick/1/0.xml
pas        14675  0.0  0.0   9096  2628 pts/1    S+   21:53   0:00 grep --color=a
uto roslaunch
pas@pas-VirtualBox:~/catkin_ws/src/ur5_data_collect_fw/scripts$ kill 14090
```

Rosparam list:

- asl/stop_ros (input)

When set to True, the program checks the condition as success. It stops current subprocess and select next config file for the next round of running subprocess.

- asl/task_fail (input)

When set to True, the program checks the condition as fail. It stops current subprocess and select previous config file for the next round of running subprocess.

- asl/stop_record (output)

Use to tell other node to stop recording

Table 5.2. Arguments of record.py

Arguments (Optional)	Definition	Default Value
-open	path to xml file or folder	config/item_spawn.xml
-num_open	number of file config in the -open folder to open	0
-save	path to save generated complete xml file	-
-task	task specification: 1 for peg with trajectory, 2 for peg with complaince, 3 for pick and place, 4 for panda peg	1
-timeout	time limit for doing the task in one round (in sec)	300

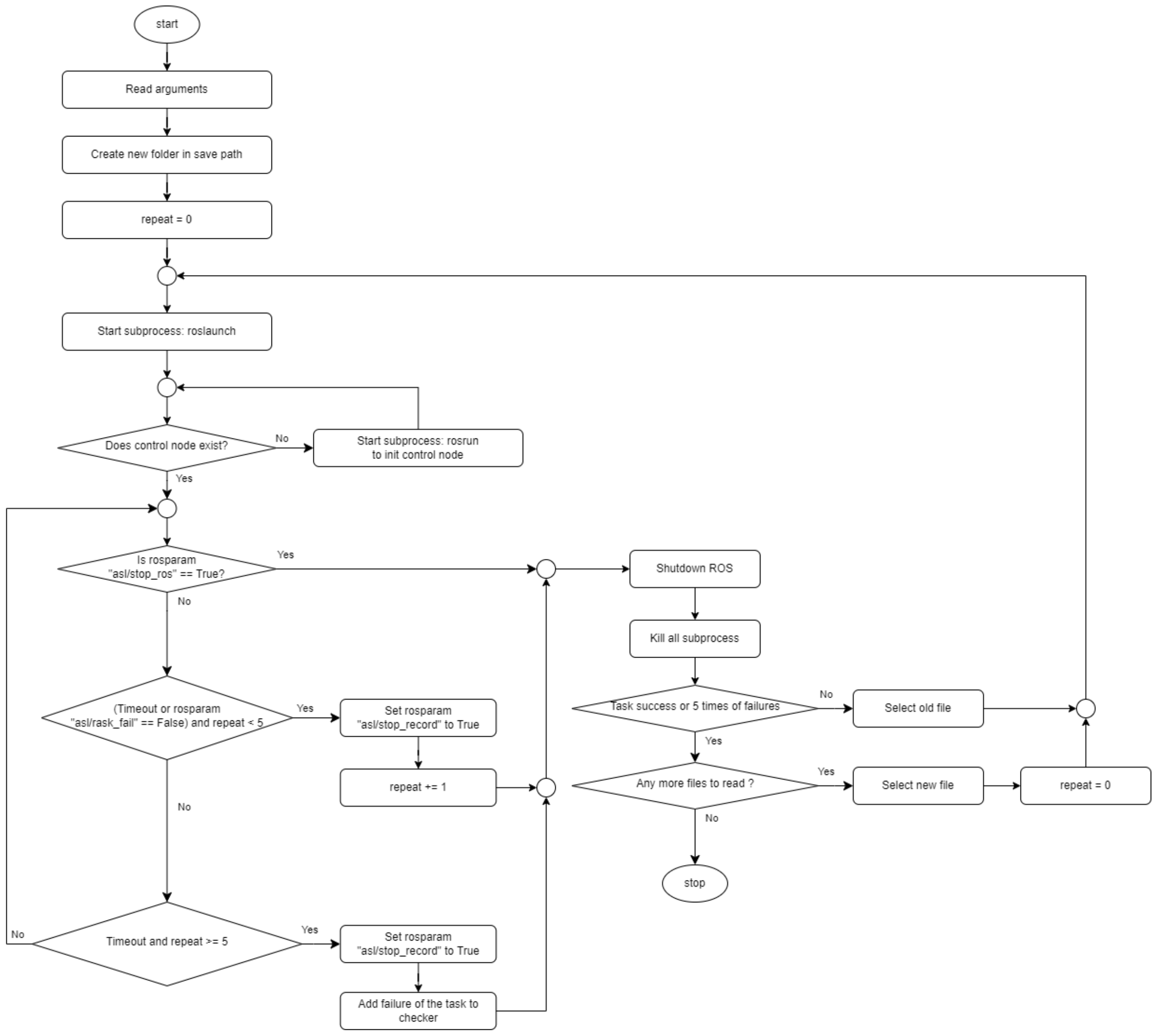

start
Read arguments
Create new folder in save path
repeat = 0
Start subprocess: roslaunch
Does control node exist?
No
Start subprocess: rosrun to init control node
Yes
Is rosparam "asl/stop_ros" == True?
Yes
No
Shutdown ROS
Kill all subprocess
Task success or 5 times of failures
No
Select old file
Yes
Any more files to read ?
Yes
Select new file
repeat = 0
No
stop
(Timeout or rosparam "asl/rask_fail" == False) and repeat < 5
Yes
Set rosparam "asl/stop_record" to True
repeat += 1
No
No
Timeout and repeat >= 5
Yes
Set rosparam "asl/stop_record" to True
Add failure of the task to checker

Figure 5.1. Flowchart of run_loop.py

6. Details of launch file

6.1 empty world remap

A launch file for specify gazebo world with a reference to package gazebo_ros/launch/empty_world.launch, with additional remapping for a compatibility to use cartesian compliance controller

6.2 gripper

A launch file to launch UR5 robotic arm with gripper as an end effector for doing the task of picking and placing objects in the system by using joint trajectory controller

6.3 peg

A launch file to launch UR5 robotic arm with peg as an end effector for doing the task of inserting peg by using joint trajectory controller or cartesian compliance controller

6.4 peg_basic

A launch file to launch UR5 robotic arm with peg using joint position controller to visualizing playback data from bag file in the joint_states topic to the simulation

6.5 peg_panda

A launch file to launch PANDA robotic arm with peg as an end effector for doing the task of inserting peg by using joint trajectory controller

Table 6. Arguments of launch files, e.g., gripper and peg

Arguments (Optional)	Definition	Default Value
rviz	Open RViz for UI of compliance controller	true
moveit_rviz	Open RViz for UI of MoveIt	true
compliance_controller	Want to use compliance controller or not, if not joint trajectory controller will be selected	false
control_peg	Plan paths for the task and command the robot to move along the paths	false
control_peg_compliance		
control_pick		
spawn_open	Path to xml config	“”
spawn_save	Path to save output	“”