1. Appendix

1.1. Example Task Definition

Below we provide full code listings for a generated task definition, simulation, and test that result from running GRS on a real-world image.

Listing 1. Example task definition.

```
    "task-name": "organize-food-items",
    "task-description": "Pick up all the food items
         on the table and place them into the open box
          in the center.",
    "assets-used": [
        "HOPE/GranolaBars.urdf",
        "HOPE/MacaroniAndCheese.urdf",
        "HOPE/OrangeJuice.urdf",
        "HOPE/Raisins.urdf",
        "HOPE/Mustard.urdf",
        "HOPE/Cookies.urdf"
    ]
```

Listing 2. Example task code.

```
class PackFoodItemsInBox(Task):
    """Pick up all the food items and place them
         inside the open box."""

    def __init__(self):
        super().__init__()
        self.max_steps = 20  # Increased max steps to
             give the oracle more time
        self.lang_template = "pack the {} in the
            brown box"
        self.task_completed_desc = "task completed."
        self.sixdof = True  # Allowing six degrees of
             freedom for more accurate placement
        self.additional_reset()

    def reset(self, env):
        super().reset(env)

        # Object bounding boxes.
        object_bounding_boxes = [
            {'bbox_corners': [np.array([0.76183139,
                -0.0314516, -0.00410459]), np.array
                ([0.62986134, 0.13964655, 0.0253295])
                ], 'urdf': 'HOPE/GranolaBars.urdf'},
            {'bbox_corners': [np.array([0.51239947,
                0.20819839, 0.01941574]), np.array
                ([0.2688748, 0.42548292, 0.21593083])
                ], 'urdf': 'HOPE/MacaroniAndCheese.
                urdf'},
            {'bbox_corners': [np.array([0.53965911,
                -0.01996892, 0.05628724]), np.array
                ([0.45665461, 0.05610127,
                0.24530269])], 'urdf': 'HOPE/
                OrangeJuice.urdf'},
            {'bbox_corners': [np.array([0.43465213,
                -0.16156002, 0.02437929]), np.array
                ([0.378738, -0.08377419, 0.14193851])
                ], 'urdf': 'HOPE/Raisins.urdf'},
            {'bbox_corners': [np.array([0.53083534,
                0.16836134, -0.02701451]), np.array
                ([0.4384108, 0.31221087, 0.0206661])
                ], 'urdf': 'HOPE/Mayo.urdf'},
            {'bbox_corners': [np.array([0.6035398,
                0.10533048, -0.01784448]), np.array
                ([0.52428974, 0.26218855,
                0.03146984])], 'urdf': 'HOPE/Mustard.
                urdf'},
            {'bbox_corners': [np.array([0.64027918,
                -0.45123034, 0.05872372]), np.array
                ([0.48848391, -0.34876966,
                0.23379576])], 'urdf': 'HOPE/Cookies.
                urdf'}
        ]

        # Add the box.
        box_size = (0.3, 0.2, 0.1)
        box_pose = ((0.5, 0, box_size[2] / 2), (0, 0,
             0, 1))  # Ensure the box is centered and
             above the table
        box_template = 'container/container-template.
            urdf'
        replace = {'DIM': box_size, 'HALF': (box_size
            [0] / 2, box_size[1] / 2, box_size[2] /
            2)}
        box_urdf = self.fill_template(box_template,
            replace)
        box_id = env.add_object(box_urdf, box_pose, '
            fixed')

        # Add food items.
        objects = []
        for bbox in object_bounding_boxes:
            urdf = bbox['urdf']
            corners = bbox['bbox_corners']
            obj_size = corners[1] - corners[0]

            # Ensure the object is placed randomly
                within the bounds and above the table
            obj_pose, obj_rot = self.get_random_pose(
                env, obj_size)
            while obj_pose is None:
                obj_pose, obj_rot = self.
                    get_random_pose(env, obj_size)

            obj_id = env.add_object(urdf, (obj_pose,
                obj_rot))
            objects.append(obj_id)

        # Ensure the object poses are above the table
             and within bounds
        valid_targ_poses = []
        for obj_id in objects:
            pos, _ = p.getBasePositionAndOrientation(
                obj_id)
            if (self.bounds[0, 0] <= pos[0] <= self.
                bounds[0, 1] and
                self.bounds[1, 0] <= pos[1] <= self.
                    bounds[1, 1] and
                self.bounds[2, 0] <= pos[2] <= self.
                    bounds[2, 1]):
                valid_targ_poses.append((pos, (0, 0,
                    0, 1)))

        # Define the goals.
        for i, obj_id in enumerate(objects):
            self.add_goal(
                objs=[obj_id],
                matches=np.int32([[1]]),
                targ_poses=[box_pose],
                replace=False,
                rotations=True,
                metric='pose',  # Changed to 'pose'
                    for simpler task completion
                params=[],
                step_max_reward=1 / len(objects),
                language_goal=self.lang_template.
                    format(object_bounding_boxes[i]['
                    urdf'].split('/')[1].split('.')
                    [0])
            )
```

Listing 3. Example test code.

```
class TestPackFoodItemsInBox(unittest.TestCase):

    def setUp(self):
        assets_root = os.path.join(os.environ.get('
            CLIPORT_PATH', os.getcwd()), "cliport/
            environments/assets/")
        self.env = Environment(
            assets_root,
```

```
            disp=False,
            hz=480,
            record_cfg=[],
        )
        self.task = PackFoodItemsInBox()
        self.env.set_task(self.task)
        self.env.reset()

    def test_oracle_succeeds(self):
        oracle_agent = self.task.oracle(self.env)
        obs = self.env.reset()
        info = self.env.info

        total_reward = 0
        for _ in range(self.task.max_steps):
            act = oracle_agent.act(obs, info)
            if act is None:
                break  # Ensure we handle cases where
                    the oracle cannot generate an
                    action
            obs, reward, done, info = self.env.step(
                act)
            total_reward += reward

            if done:
                break

        # Ensure that the task is completed
            successfully.
        self.assertTrue(done, "Oracle agent did not
            complete the task within the allowed
            steps.")
        self.assertAlmostEqual(total_reward, 1.0,
            delta=0.01, msg="Oracle agent did not
            achieve maximum reward.")

    def test_oracle_picks_and_places_objects(self):
        oracle_agent = self.task.oracle(self.env)
        obs = self.env.reset()
        info = self.env.info

        pick_count = 0
        place_count = 0

        for _ in range(self.task.max_steps):
            act = oracle_agent.act(obs, info)
            if act is None:
                break  # Ensure we handle cases where
                    the oracle cannot generate an
                    action
            obs, reward, done, info = self.env.step(
                act)

            if act and 'pose0' in act and 'pose1' in
                act:
                pick_count += 1
                place_count += 1

            if done:
                break

        # Ensure that the oracle agent attempted to
            pick and place objects.
        self.assertGreater(pick_count, 0, "Oracle
            agent did not attempt to pick any objects
            .")
        self.assertGreater(place_count, 0, "Oracle
            agent did not attempt to place any
            objects.")

    def test_task_completion_description(self):
        oracle_agent = self.task.oracle(self.env)
        obs = self.env.reset()
        info = self.env.info

        total_reward = 0
        for _ in range(self.task.max_steps):
            act = oracle_agent.act(obs, info)
            if act is None:
                break  # Ensure we handle cases where
                    the oracle cannot generate an
                    action
            obs, reward, done, info = self.env.step(
                act)
            total_reward += reward

            if done:
                break

        # Ensure that the task completion description
            is accurate.
        self.assertTrue(done, "Oracle agent did not
            complete the task within the allowed
            steps.")
        self.assertEqual(self.task.
            task_completed_desc, "task completed.", "
            Task completion description does not
            match.")

if __name__ == '__main__':
    unittest.main()
```

1.2. Prompts

Below we provide the prompts used for the oracle test generation, including the prompt and example of the oracle API.

Listing 4. Prompt for behavior test.

```
"""You are an AI in robot simulation code and task
    design.
My goal is to design diverse and feasible tasks for
    tabletop manipulation.

Write tests to ensure a robotic policy could solve
    this task.
Do not write tests for the environment set or goal
    structures, only the behavior of the robot policy
     itself.
The task is defined is in a file named "task_code.py
    ". Only import the task from that file.

Provide your response as a fully runnable python
    unittest script.
Only provide the script content; do not provide
    instructions on how to run it.
Do NOT mock ANY inputs to the test.
Use the guidance below to set up and initialize the
    environment for tests.

Here is the task definition:
{task}

Here is task code:
```python
{code}
```"""
```

Listing 5. Oracle prompt code.

```
"""Here is how to run the oracle agent on a task (`
    task` subclassing `Task`) in an environment (`env
    ` of type `Environment`):

```
oracle_agent = task.oracle(env)
obs = env.reset()
info = env.info

total_reward = 0
for _ in range(task.max_steps):
    act = oracle_agent.act(obs, info)
    obs, reward, done, info = env.step(act)
    total_reward += reward
```
```

```
    if done:
        break
```"""