Force Sensors

Rigid Body Force Sensors

Force sensors can be attached to rigid bodies to measure forces and torques experienced at user-specified reference frames. The readings represent the net forces and torques experienced by the parent body, which include external forces, contact forces, and internal forces applied by the solver (e.g., due to joint drives). A body resting on the ground plane will have a net force of zero.

Force sensors are created on assets. This way, you only need to define the force sensors once and they will be created on every actor instanced from the asset.

To create a force sensor, you specify the rigid body index to which the sensor will be attached and the relative pose of the sensor with respect to the body origin. Multiple sensors can be attached to the same body:

```
body_idx = gym.find_asset_rigid_body_index(asset, "bodyName")

sensor_pose1 = gymapi.Transform(gymapi.Vec3(0.2, 0.0, 0.0))
sensor_pose2 = gymapi.Transform(gymapi.Vec3(-0.2, 0.0, 0.0))

sensor_idx1 = gym.create_asset_force_sensor(asset, body_idx, sensor_pose1)
sensor_idx2 = gym.create_asset_force_sensor(asset, body_idx, sensor_pose2)
```

You can also pass additional properties for each sensor, which determine how the forces will be computed:

```
sensor_props = gymapi.ForceSensorProperties()
sensor_props.enable_forward_dynamics_forces = True
sensor_props.enable_constraint_solver_forces = True
sensor_props.use_world_frame = False

sensor_idx = gym.create_asset_force_sensor(asset, body_idx, sensor_pose, sensor_props)
```

See `isaacgym.gymapi.ForceSensorProperties` for more details.

After creating actors, you can get the number of attached force sensors and access the individual force sensors using their index:

```
actor_handle = gym.create_actor(env, asset, ...)

num_sensors = gym.get_actor_force_sensor_count(env, actor_handle)
for i in range(num_sensors):
    sensor = gym.get_actor_force_sensor(env, actor_handle, i)
```

During simulation, you can query the latest sensor readings like this:

```
sensor_data = sensor.get_forces()
print(sensor_data.force)   # force as Vec3
print(sensor_data.torque)  # torque as Vec3
```

Two sensors attached to the same body will report the same force, but will generally differ in torques if their relative poses are different. Forces are measured at the body's center of mass, but torques are measured at the sensor's local reference frame.

The total number of force sensors in a simulation can be obtained by calling `gym.get_sim_force_sensor_count(sim)`.

Tensor API

The function `acquire_force_sensor_tensor` returns a Gym tensor descriptor, which can be wrapped as a PyTorch tensor as discussed in the Tensor API documentation:

```
_fsdata = gym.acquire_force_sensor_tensor(sim)
fsdata = gymtorch.wrap_tensor(_fsdata)
```

The shape of this tensor is (num_force_sensors, 6) and the data type is float32. For each sensor, the first three floats are the force and the last three floats are the torque.

After each simulation step, you can get the latest sensor readings by calling:

```
gym.refresh_force_sensor_tensor(sim)
```

Limitations

Force sensors are currently only available with the PhysX backend.

Joint Force Sensors

To enable reading forces on each degree-of-freedom of articulated actors, call the `enable_actor_dof_force_sensors` method when setting up the envs:

```
actor = gym.create_actor(env, ...)
gym.enable_actor_dof_force_sensors(env, actor)
```

These sensors are not enabled by default for performance reasons. Although the performance impact is usually quite small, it is best to only enable the sensors when needed. During simulation, the forces can be retrieved using the `get_actor_dof_forces` function:

```
forces = gym.get_actor_dof_forces(env, actor_handle)
```

This returns a numpy array of total forces acting on this actor's DOFs. Note that the returned forces will always be zero if `enable_actor_dof_force_sensors` was not previously called on the actor.

Tensor API

The function `acquire_dof_force_tensor` returns a Gym tensor descriptor, which can be wrapped as a PyTorch tensor as discussed in the Tensor API documentation:

```
_forces = gym.acquire_dof_force_tensor(sim)
forces = gymtorch.wrap_tensor(_forces)
```

This is a one-dimensional tensor of float32 values corresponding to each DOF in the simulation.

After each simulation step, you can get the latest sensor readings by calling:

```
gym.refresh_dof_force_tensor(sim)
```

Limitations

DOF force sensors are only supported in the PhysX backend.