

AWS RoboMaker ハンズオン

「迷路ランナー - 迷路から抜け出す方法を見つけよう！」

2019/10/8

ここでは強化学習を行うロボットアプリケーションを使い、AWS RoboMaker の機能のうち、「開発環境」と「シミュレーション」機能についてみていきます。

ロボットアプリケーションは AWS RoboMaker の「シミュレーション」上で動作します。シミュレーション環境は、壁で囲まれた迷路のような部屋となっており、ロボットはこの部屋を壁にぶつからないように GOAL の場所までたどり着けるよう学習します。シミュレーション内では Turtlebot3 Burger というロボットを操作します。このロボットには 360 度、障害物までの距離を捉えることができる、LiDAR センサーが搭載されています。ロボットアプリケーションはこのセンサーからの情報を環境情報として受け取り、壁にぶつからずに迷路を通過する方法を学習していきます。



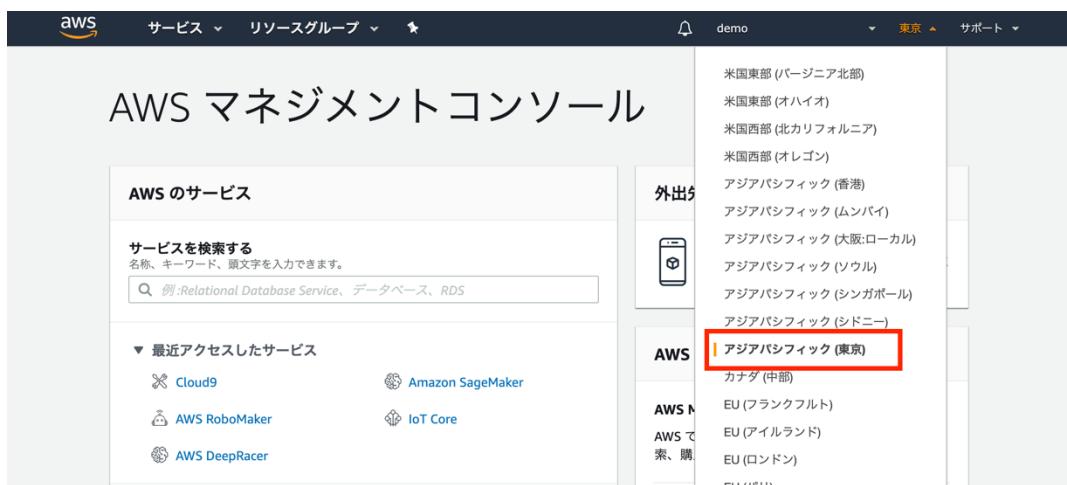
強化学習には Coach というフレームワークを利用しておおり、また OpenAI Gym が提供するライブラリを使って強化学習用の環境を実装しています。Coach、OpenAI Gym それぞれについて詳しくは次を参照してください。

<https://github.com/NervanaSystems/coach>

<https://gym.openai.com/>

1. 準備 - ハンズオン環境のセットアップ

- 1-1. AWS マネージメントコンソール (<https://console.aws.amazon.com>) を開きます。リージョンは アジアパシフィック（東京）を使用します。



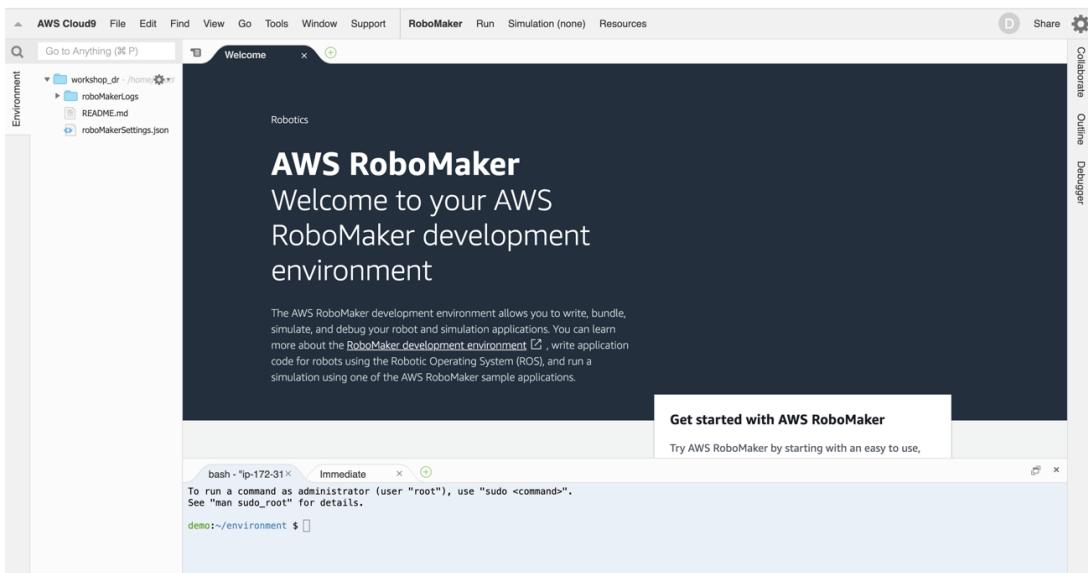
- 1-2. AWS マネジメントコンソールでサービスから RoboMaker を選んで開き、左のナビゲーションペインで **開発** -> **開発環境**を選びます。「開発環境」の画面が開かれたら右上の**[環境の作成]** ボタンをクリックします。



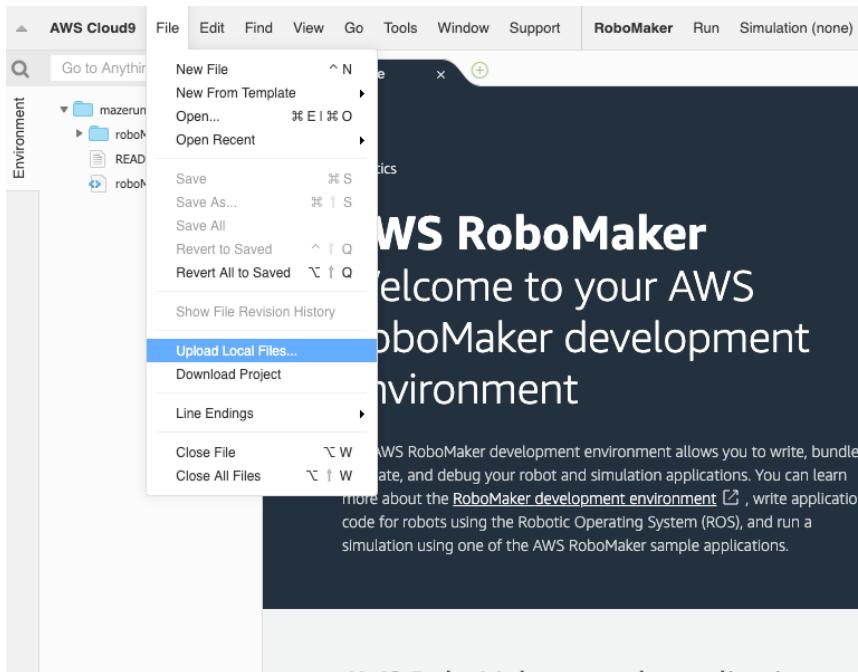
- 1-3. 名前に任意の名前を設定、サポートされているソフトウェアスイートは **ROS Kinetic** を、インスタンスタイプはデフォルトの **m4.large** を選択、ネットワークの設定は VPC はブレダウンからデフォルトの VPC を選択し、サブネットはブレダウンから任意の一つを選択し、最後に [作成] ボタンをクリックして、開発環境の構築を開始します。



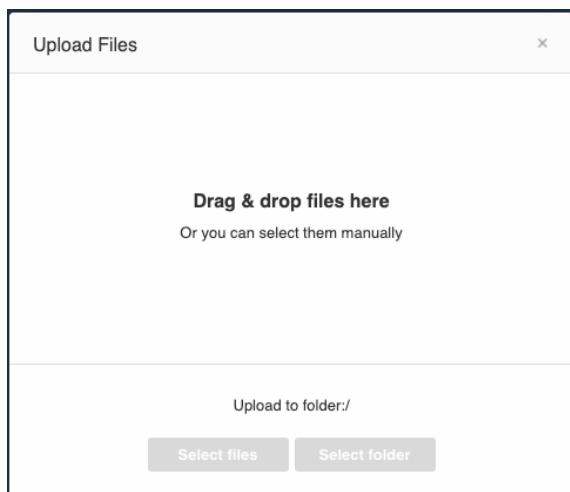
1-4. AWS のクラウド IDE サービス、AWS Cloud9 をベースに AWS RoboMaker 用に機能拡張した開発環境が開きます。



1-5. メニューから *File -> Upload Local Files…* を選択します。



1-6. ポップアップウィンドウ「Update Files」が開きます。

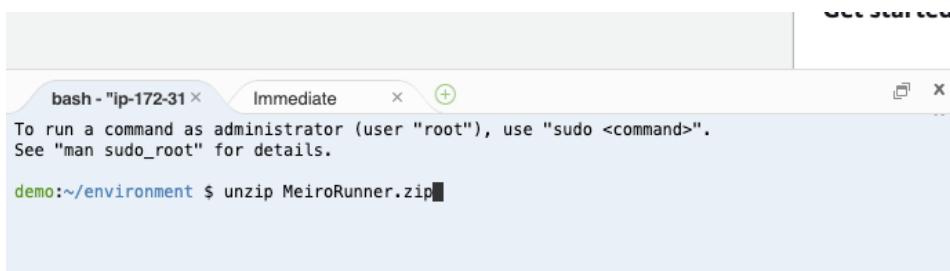


あらかじめダウンロードしておいた MeiroRunner.zip ファイルを「**Drag & drop files here**」の領域にドラッグ&ドロップしてファイルを開発環境にアップロードします（または [Select files] ボタンをクリックして、ファイルを選択することもできます）。

1-7. ファイルが開発環境に取り込まれます。ファイルのアップロードが完了したらこのウィンドウは不要なので、「**Update Files**」ウィンドウ右上の [x] ボタンをクリックして、ウィンドウを閉じます。

1-8. 開発環境下方にあるターミナルウィンドウをクリック、次のコマンドを入力します。

```
unzip MeiroRunner.zip
```



A screenshot of a terminal window titled "bash - "ip-172-31". The window shows the command "unzip MeiroRunner.zip" being typed. A message at the top of the terminal window reads: "To run a command as administrator (user "root"), use "sudo <command>". See "man sudo_root" for details."

1-9. さらに次のコマンドを入力します。

```
cd MeiroRunner  
./setup.sh
```



A screenshot of a terminal window titled "bash - "ip-172-31". The window shows the commands "cd MeiroRunner" and "./setup.sh" being typed. The line "cd MeiroRunner" is highlighted with a red box.

今回のハンズオン環境の構築が開始されます。

このスクリプトは次のことを行なっています。

- 開発環境を最新のソフトウェアバージョンに更新
- 演習に必要なリソースの準備
 - 演習で使う S3 バケットの作成
 - 演習で使う IAM ロールの作成
 - Security グループと Subnet の確認
 - シミュレーションアプリケーションの名前の作成
 - 開発環境の設定ファイル roboMakerSettings.json ファイルの更新
 - 初回のビルド(初回のビルドは時間がかかるのでここで実行しています)

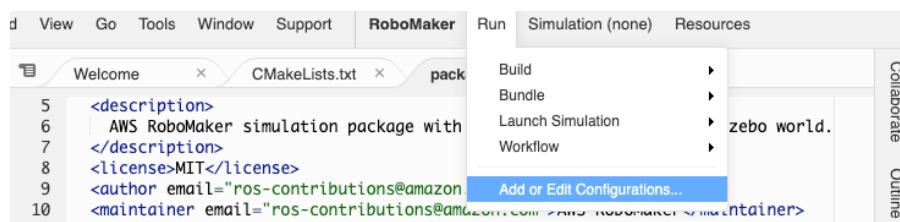
途中でエラーが出たらお知らせください。

この処理は完了までおよそ 20 分かかります。終了すると次のような表示になります。この作業の完了を待っている間に今回のハンズオンの内容を説明します。

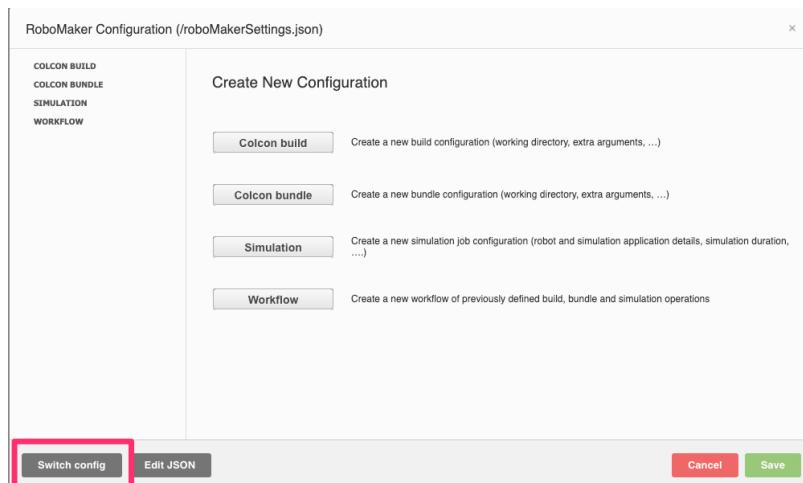


```
bash - "colcon bi" Immediate (Java) x +  
1..1 pyyaml-dateutil-2.8.0 pytz-2019.2 requests-2.22.0 requests-oauthlib-1.2.0 rfc  
each-slim-0.11.1 rospkg-1.1.7 rsa-4.0 s3transfer-0.1.13 scikit-image-0.15.0 scipy-0.19.0 setup  
ools-41.1.0 six-1.12.0 tensorflow-1.12.2 tensorboard-1.12.2 tensorflow-1.12.2 termcolor-1.1.0 tornado-6.0.3 urlli  
b3-1.25.3 wcwidth-0.1.7 websocket-client-0.56.0 werkzeug-0.15.5 wheel-0.33.6 zipp-0.5.2  
Creating bundle archive V2...  
Archiving complete!  
=>OK  
demo:~/environment/MeiroRunner $
```

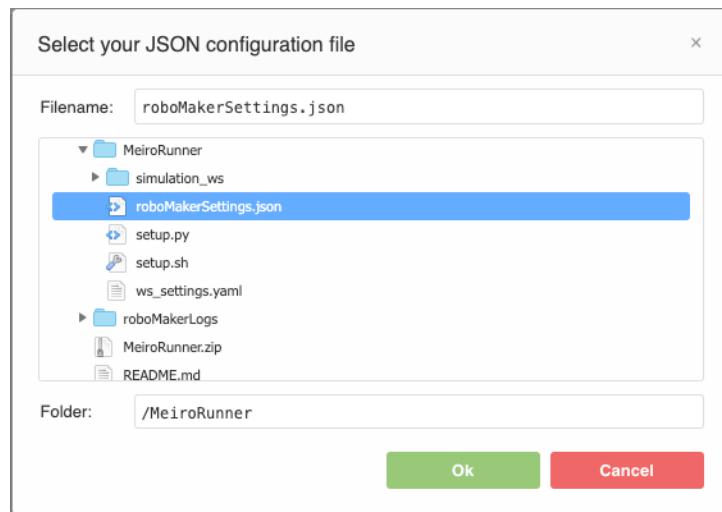
- 1-10. 開発環境のメニューから *Run -> Add or Edit Configurations…* を選択します。



- 1-11. 「RoboMaker Configuration」 ウィンドウが開かれます。左下にある [Switch config] ボタンをクリックします。

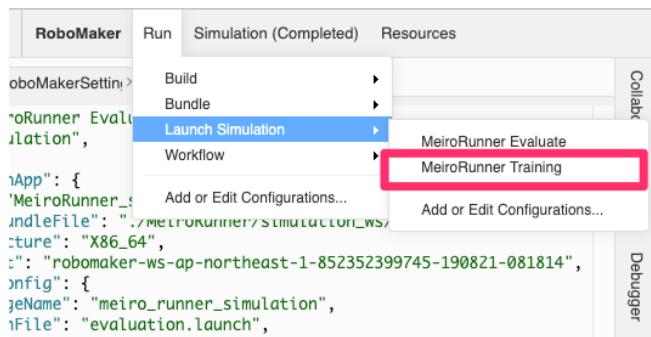


- 1-12. 「Select your JSON configuration file」 ウィンドウが開きます。 MeiroRunner ディレクトリの中にある roboMakerSettings.json ファイルを選択して[Ok] ボタンをクリック。 「RoboMaker Configuration」 ウィンドウで [Save] ボタンをクリックして、設定を保存します。

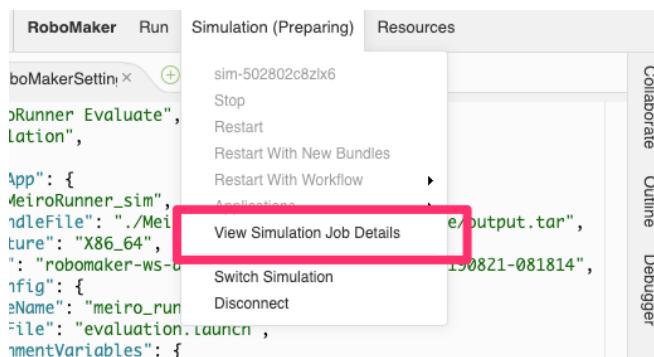


2. アプリケーションを試す

- 2-1. 開発環境のメニューより *Run -> Launch Simulation -> MeiroRunner Training* を選びます。シミュレーションが開始されます。



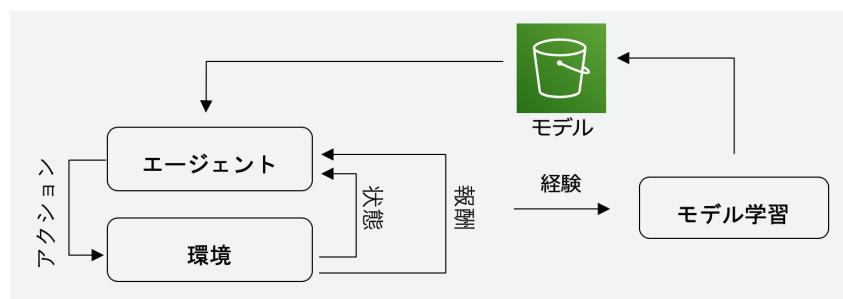
- 2-2. 開発環境のメニューより *Simulation -> View Simulation Job Details* を選びます。シミュレーションジョブの詳細画面が開きます。



- 2-3. “Status” の表示が “Running” になるまで待ちます。これは数分かかる場合があります。“Running” 状態になると各ツールを開くためのアイコンが表示されます。
[Gazebo] アイコンをクリックします。 Gazebo はシミュレータです。AWS RoboMaker 開発環境で開発したアプリケーションは Gazebo シミュレータで実行することができます。

2-4. Gazebo のシミュレーションウィンドウが開かれます。

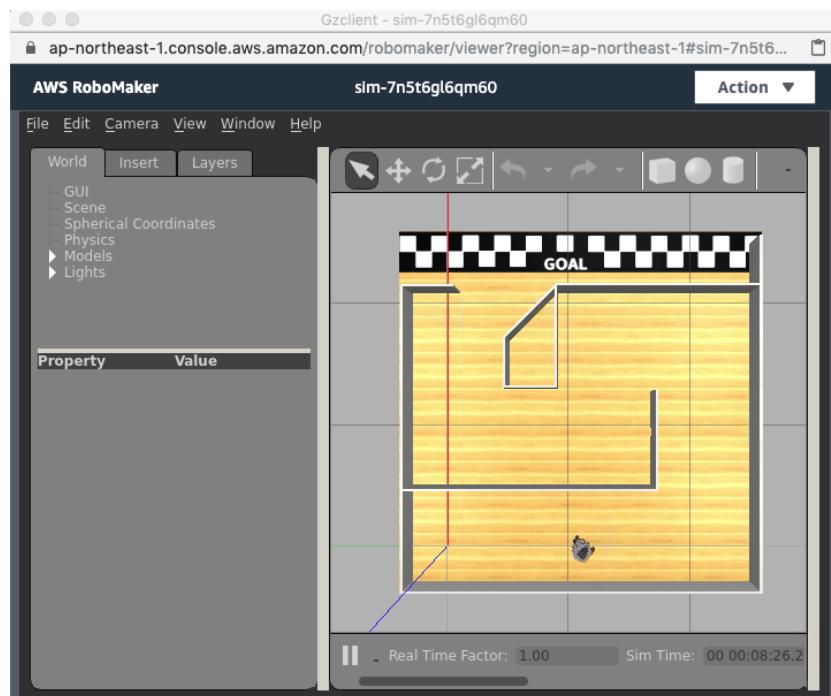
- 左下から移動を始めるのがロボットです。Turtlebot3 Burger というロボットをシミュレーション環境で動かしています。
- アプリケーションは強化学習を行なっています。強化学習ではエージェントが環境に対してアクションを起こし、更新された環境の状態とアクションの良さに応じた報酬を受け取ります。アクション、状態、報酬は経験として蓄積され、より良いアクションを起こせるようモデルが改善されます。今回の例ではロボットとロボットが移動する部屋が強化学習においての環境となり、エージェントは環境であるロボットのハンドルとアクセルを操作します。壁にぶつかると1回の試行は終了し、ロボットはスタート地点に戻されます。およそ1.5時間の学習でロボットは GOAL 地点までたどり着くことができるようになります。



強化学習についての詳細は利用している強化学習フレームワーク Coach を参考にしてください。

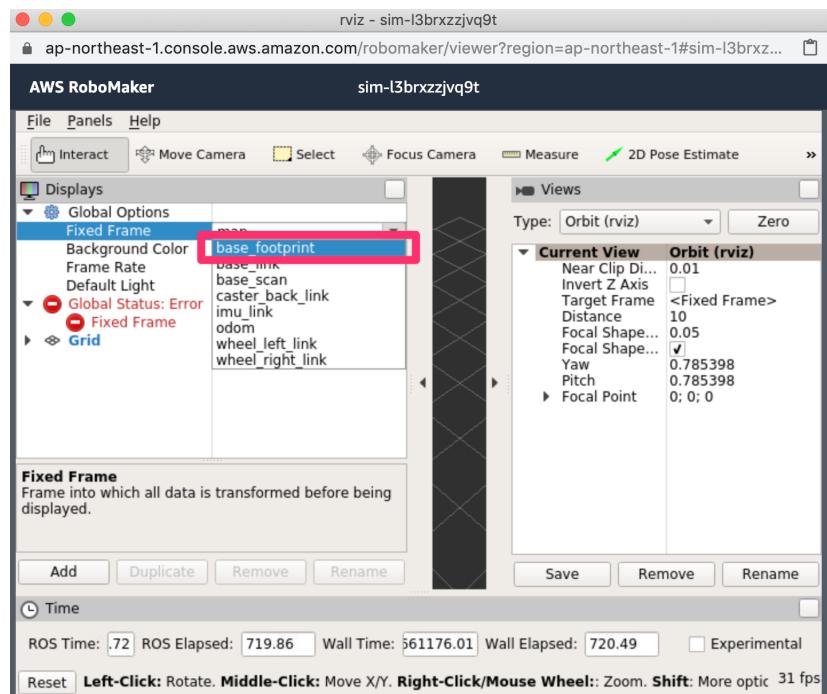
<https://github.com/NervanaSystems/coach>

Gazebo の画面は次のようになっています。

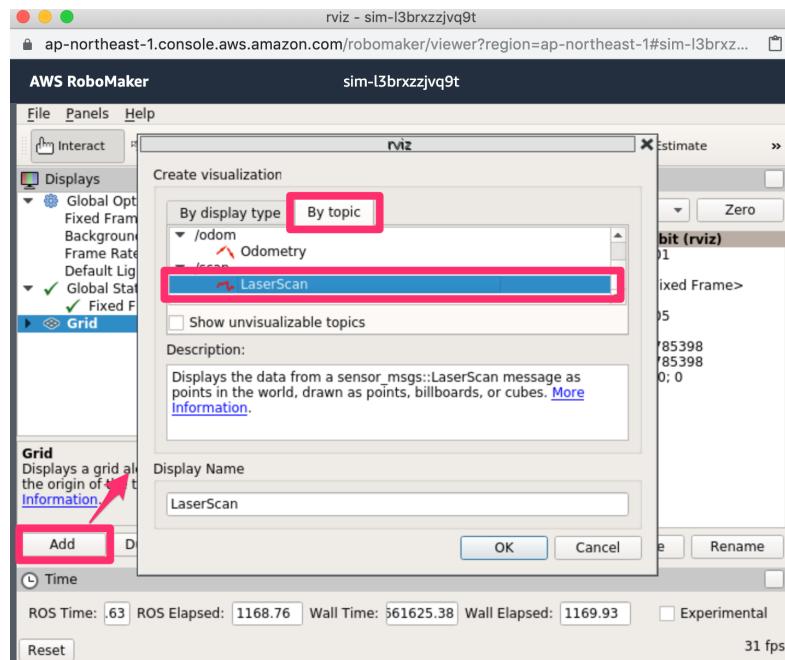


- 2-5. シミュレーションジョブの詳細画面では Gazebo 以外にも様々なプラグインを提供する「rqt」、視覚化ツール「rviz」、コマンド入力によりロボットアプリケーションを操作、デバッグするための「Terminal」ツールが提供されています。ここでは rviz についても少し確認しましょう。rviz は「ロボットが見えている世界」を視覚化するツールです。今回扱う Turtlebot3 Burger は 360 度、周辺の障害物までの距離を測定できる LiDAR センサーを搭載しています。LiDAR センサーの情報を視覚化しましょう。

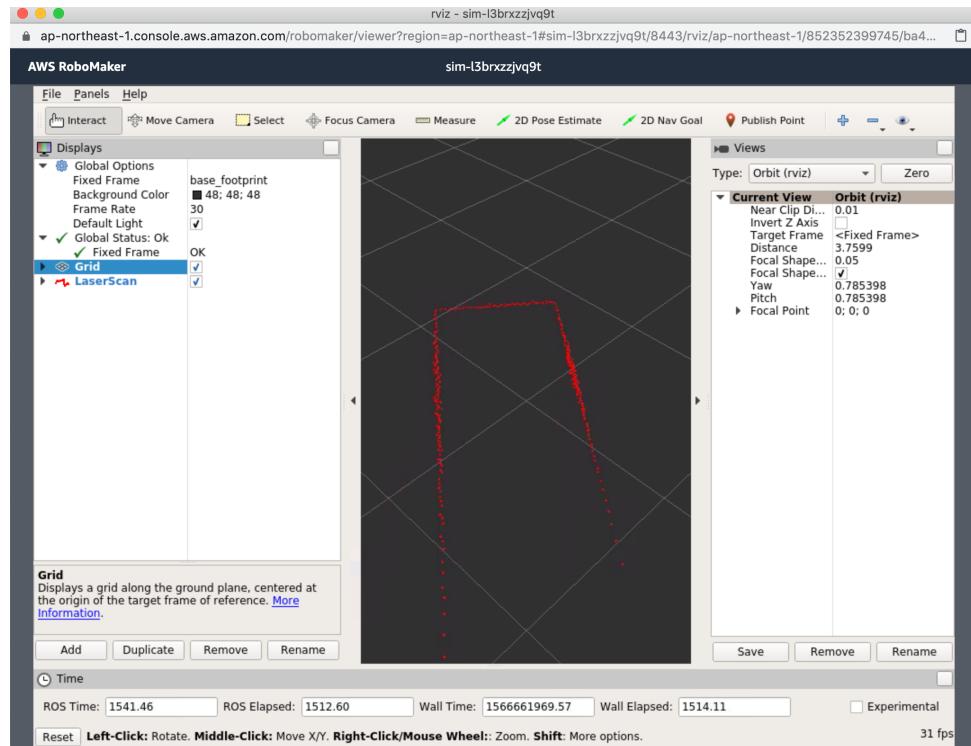
まずシミュレーションジョブの詳細画面にある「rviz」のアイコンをクリックして rviz のウィンドウを開きます。rviz ウィンドウの中 Displays の **Fixed Frame** の設定について、プルダウンリストを開き **base_footprint** に変更します。（これは視覚化する際の座標系を指定しています）



次に 左下 [Add] ボタンをクリックします。「Create visualization」ウィンドウが開きます。[By topic] タブを選択し、/scan -> LaserScan を選択、[Ok] ボタンをクリックします。

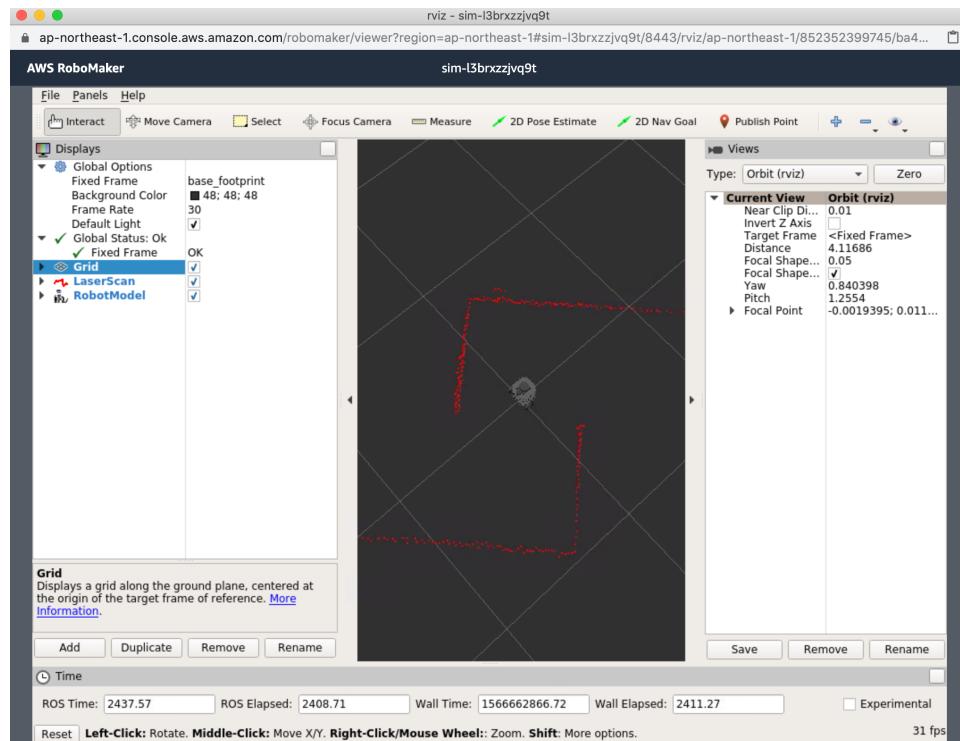
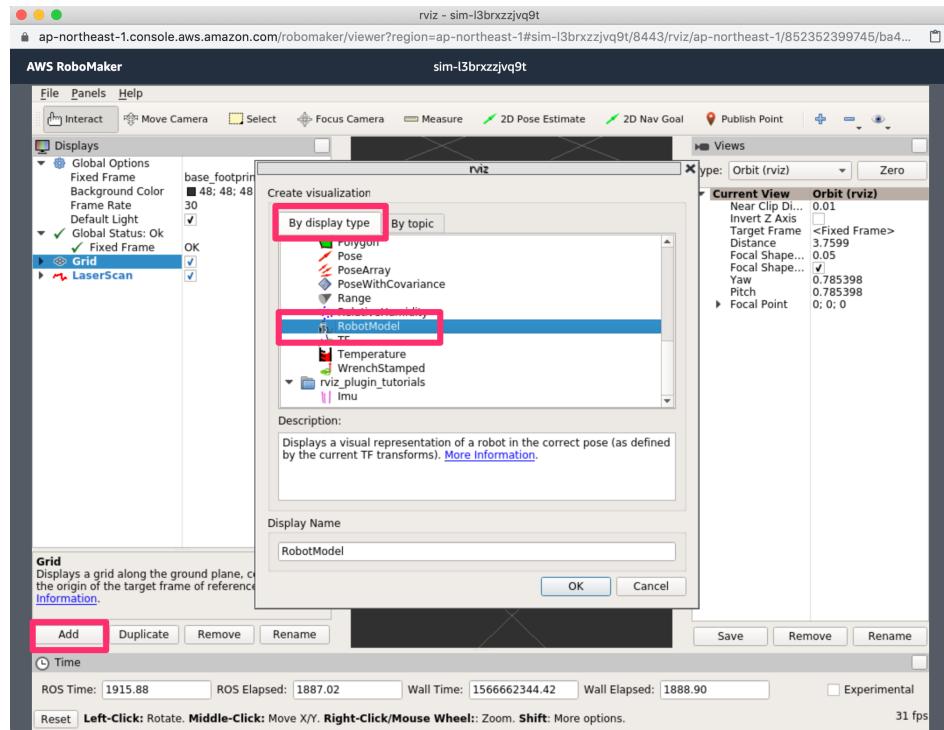


少しウィンドウのサイズを大きくして視覚化領域（真ん中の黒い領域）が見えるようにしましょう。赤い点で表示されているのが LiDAR が捉えている障害物です（マウス操作で、見やすいサイズ、アングルに調整してください。マウスホイールで視点を近づけ、遠ざけることができます）。強化学習はこの LiDAR の情報を環境の状態として受け取っています。



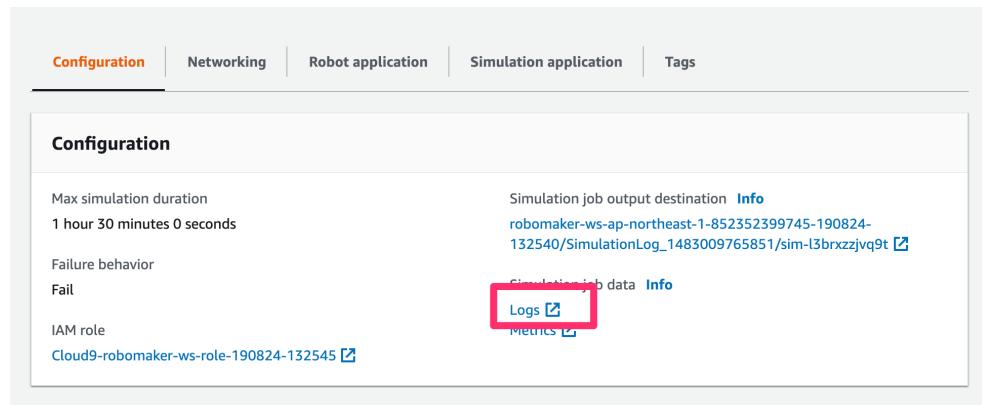
視覚化されているものの中にロボット自体も表示されていた方が視覚化されたものとロボット本体との相対関係がわかり理解しやすいでしょう。次にロボット自体を視覚化します。再度 [Add] ボタンをクリックし「Create visualization」ウィンドウが開きます。

[By display type] タブから **RobotModel** を選び [Ok] ボタンをクリックします。視覚化領域にロボット本体も表示されます。

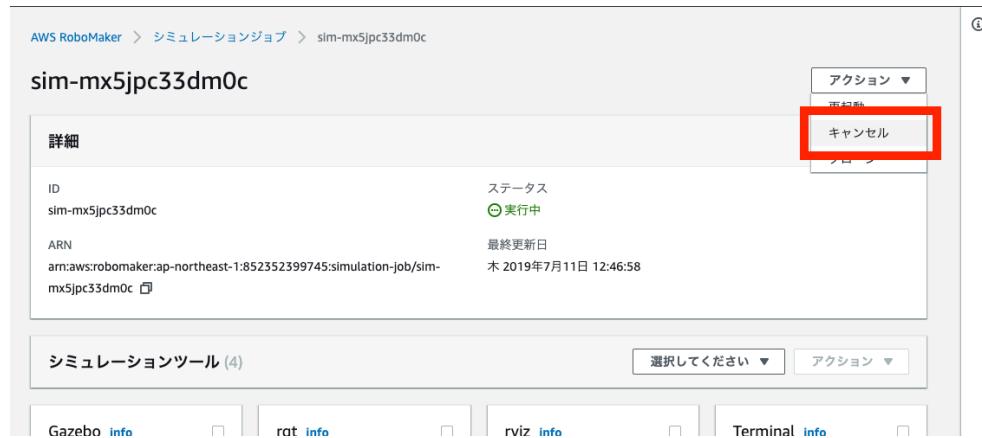


(部屋を固定して、その中をロボットが移動するように見えた方がわかりやすいかもしれません。こうしたい場合は、最初に変更した **Fixed Frame** の設定を **odom** に変更します)

- 2-6. シミュレーションジョブ詳細画面の下方、「**Configuration**」から実行中のアプリケーションのログを確認することができます。今回のアプリケーションでは強化学習の進行状況などがログで出力されています。



- 2-7. 一通り確認が終わったらシミュレーションジョブを終了させます。シミュレーションジョブの終了は、シミュレーションジョブ詳細画面の [アクション] から [キャンセル] を選んで行います。



- 2-8. シミュレーションが終わったらシミュレーションジョブ詳細画面下側の「**Configuration**」タブの **Simulation job output destination** にあるリンクをクリックしてみましょう。S3 バケットが開きます。ここには完了したシミュレーションの

rosbag ファイル（シミュレーションジョブの中で交換された ROS のメッセージの記録）と各 ROS ノードの出力が保存されています。

The screenshot shows the AWS RoboMaker Configuration page. At the top, there are tabs: Configuration (which is selected and highlighted in orange), Networking, Robot application, Simulation application, and Tags. Below the tabs, there's a section titled "Configuration". Under "Configuration", there are three main sections: "Max simulation duration" (set to 1 hour 30 minutes 0 seconds), "Failure behavior" (set to Fail), and "IAM role" (Cloud9-robomaker-ws-role-190824-132545). To the right of these, under "Simulation job output destination", there is an "Info" link and a URL: roboMaker-1483009765851-sim-l3bxzzjvq9t. This URL is highlighted with a red rectangular box. Below this, there are links for "Simulation job data", "Logs", and "Metrics".

3. ソースコードを理解する

ソースコードの中で重要な箇所をいくつか紹介します。

`MeiroRunner/simulation_ws/src/rl_agent/markov` 配下

environments/meiro_runner_env.py

OpenAI の Gym ライブラリを利用して強化学習における「環境」が実装されています。ここではエージェントからの指示に従いロボットを操作し、その結果起こったロボットの変化（LiDAR のセンシングデータの変化など）から報酬を計算し、更新されたステートと報酬をエージェントに返します。報酬関数はこのファイルの中に実装されています。関数名は `infer_reward_state` です。また ロボットの操作は `send_action` 関数を呼び出すことで行われています。ROS の観点ではここで `rl_coach` というノードを起動しています。

`presets/meiro_runner.py`

機械学習のモデルや学習アルゴリズム、各種パラメータを設定しています。

single_machine_training_worker.py

単一マシンでの強化学習を開始します。今回の作業では、これを実行することで機械学習を開始しています。ROS の観点ではこのプログラムは `meiro_runner_simulation` パッケージの `local_training.launch` ファイルから `run_local_rl_agent.sh` というスクリプトファイルを経由して実行されています。

evaluation_worker.py

学習済みのモデルを評価します。開発環境のメニューから `Run -> Launch Simulation -> MeiroRunner Evaluate` を選択することで、これを実行することができます。

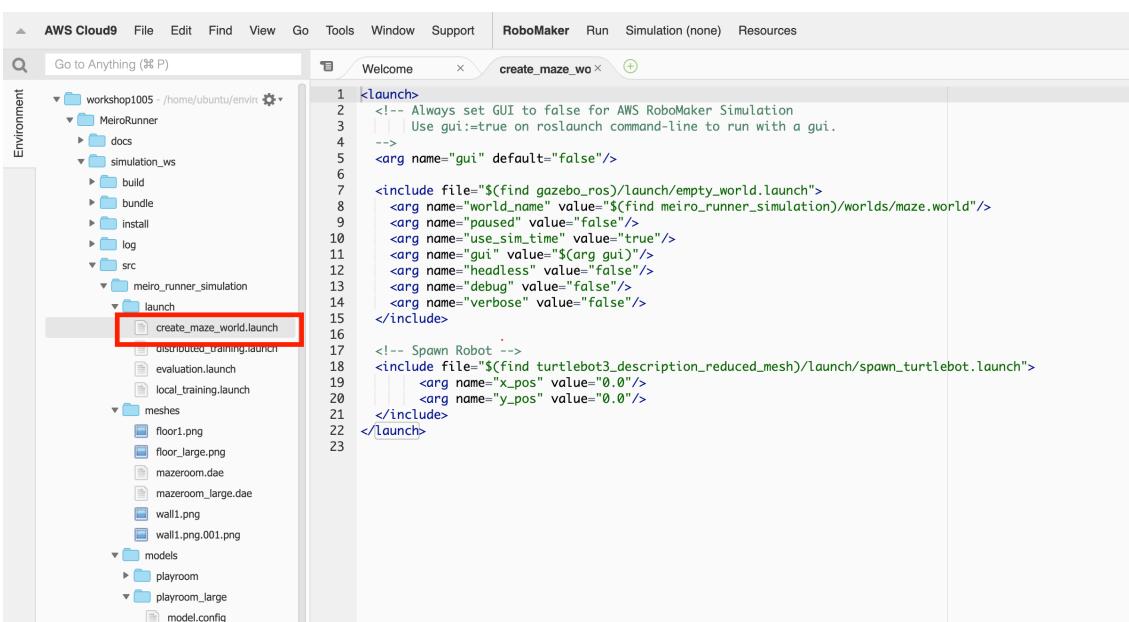
rollout_worker.py

分散強化学習を開始します。Amazon SageMaker で機械学習を、その環境に AWS RoboMaker という構成をとることができます。ここでは具体的な方法については扱いませんが、AWS DeepRacer が Amazon SageMaker と AWS RoboMaker で分散強化学習を行なっている例の一つです。

追加課題 1. 迷路を変える

ここではシミュレーションワールドに展開される迷路を変更する作業を通じて、AWS RoboMaker での開発のフローを確認していきます。

AWS RoboMaker 開発環境を開きます。ナビゲーションペイン（ウィンドウ左のファイル選択域）でファイル MeiroRunner -> simulation_ws -> src -> meiro_runner_simulation -> launch -> create_maze_world.launch を選択、ダブルクリックして開きます。



```
<launch>
  <!-- Always set GUI to false for AWS RoboMaker Simulation
       Use gui:=true on roslaunch command-line to run with a gui.
  -->
  <arg name="gui" default="false"/>
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find meiro_runner_simulation)/worlds/maze.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="$(arg gui)"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
    <arg name="verbose" value="false"/>
  </include>
  <!-- Spawn Robot --
  <include file="$(find turtlebot3_description_reduced_mesh)/launch/spawn_turtlebot.launch">
    <arg name="x_pos" value="0.0"/>
    <arg name="y_pos" value="0.0"/>
  </include>
</launch>
```

ファイルの 8 行目を次のように書き換えます。

変更前

```
<arg name="world_name" value="$(find meiro_runner_simulation)/worlds/maze.world"/>
```

変更後

```
<arg name="world_name" value="$(find meiro_runner_simulation)/worlds/maze_large.world"/>
```

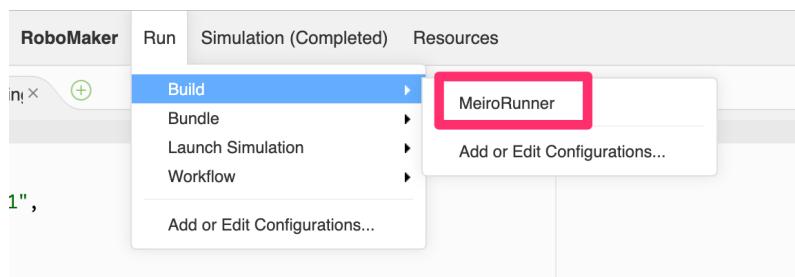
これはシミュレーションワールドの環境情報を読み込んでいる箇所です。読み込むファイルを maze.world から maze_large.world に書き換えました。書き換えたら変更を保存するために [Ctrl] + s を押します。

メモ :

変更の箇所はシミュレーションが起動された時 meiro_runner_simulation -> worlds 配下の maze_large.world を参照します。maze_large.world ファイルは迷路の壁の 3D モデルをロードします。

迷路の壁のモデルは meiro_runner_simulation -> meshes 配下の mazeroom_large.dae です。.dae は COLLADA と呼ばれる 3D モデル交換用のファイルの形式で、多くの 3D 編集ソフトウェアがこの形式に対応しています。AWS RoboMaker がサポートするシミュレータである Gazebo は 3D モデルの外観を定義するファイルの形式として STL, OBJ, Collada 形式をサポートしています。

変更した内容を反映するための**ビルド**を行います。ビルドはターミナルから行うかメニューから行うこともできます。今回はメニューから行いましょう。開発環境のメニューから Run -> Build -> MeiroRunner を選択します。

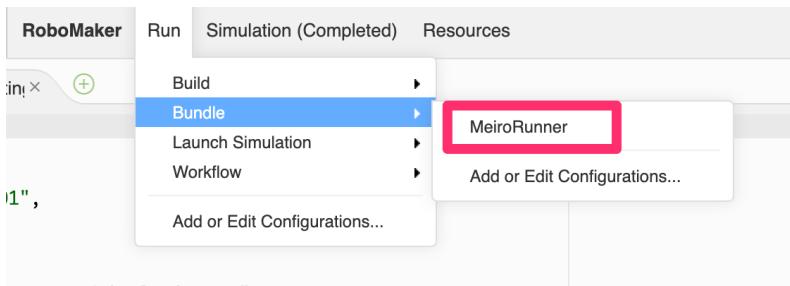


ビルドの途中経過はターミナル表示領域に表示されます。

終了するとターミナルの表示は次のようにになります。

A screenshot of a terminal window titled 'bash - "colcon bu' with several tabs. The current tab is 'Colcon Build - St' and it is active. The command entered is 'source /opt/ros/kinetic/setup.bash && rosdep install --from-paths src --ignore-src -r -y && colcon build'. The terminal output shows: 'Finished <<< turtlebot3_description_reduced_mesh [0.51s]' and 'Summary: 3 packages finished [2.42s]'. The entire output block is highlighted with a red rectangle. At the bottom, it says 'Process exited with code: 0'.

ビルドが無事完了したら、次は**バンドル**を行います。**バンドル**は**ビルド**したファイルを一つのアーカイブファイルにまとめる作業です。**ビルド**と同様**バンドル**もターミナルからとメニューから行うか、メニューから行うことができます。今回はメニューから行いましょう。開発環境のメニューから Run -> Bundle -> MeiroRunner を選択します。



バンドルの途中経過はターミナル表示領域に表示されます。

終了するとターミナルの表示は次のようにになります。

```

bash - "colcon bu" * Immediate * Colcon Build - St * Colcon Bundle - +
Run Colcon Bundle Command: source /opt/ros/kinetic/setup.bash && colcon bundle
Local dependencies not changed, skipping dependencies update...
Creating bundle archive V2...
Archiving complete!

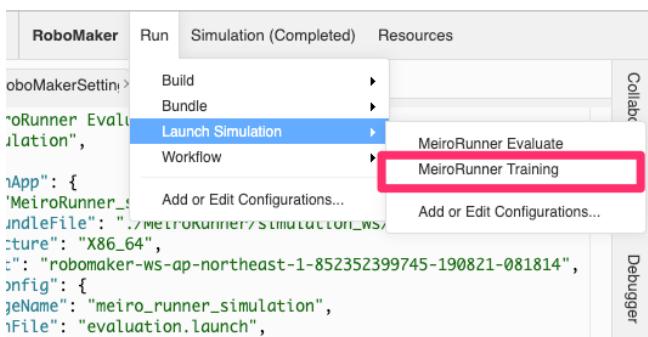
Process exited with code: 0

```

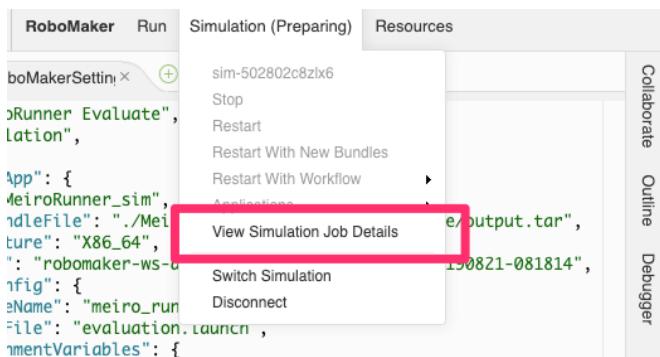
メモ :

ビルド、バンドルがメニューから操作ができるのは、`roboMakerSettings.json` ファイルに該当の設定が記述されているからです。`roboMakerSettings.json` ファイルはメニューから操作できる内容を定義します

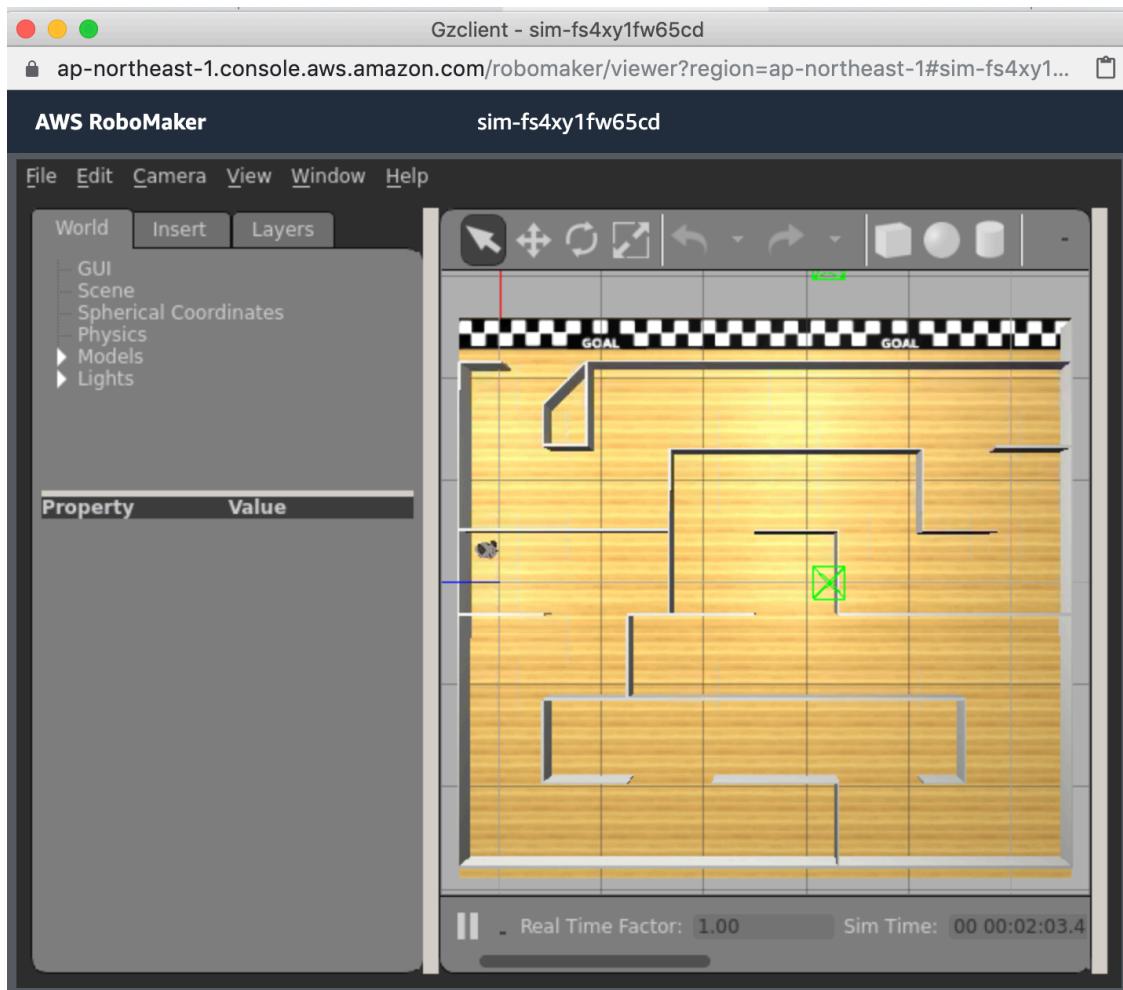
シミュレーションを起動します。



シミュレーションジョブの詳細画面を開きます。



シミュレーションジョブ詳細画面より [Gazebo] をクリックして、シミュレータ画面を開きます。
迷路が切り替わっていることを確認してください。



メモ :

シミュレーションは 1.5 時間で自動的に終了します。おそらく 1.5 時間はこの迷路を脱出するためのモデルを構築するには学習時間として不十分でしょう。シミュレーションの継続時間を延長するには roboMakerSettings.json ファイルの MeiroRunner Training の項目の maxJobDurationInSeconds の値を変更します。

roboMakerSettings.json ファイル

```
{  
    "id": "MeiroRunner_SimulationJob2",  
    "name": "MeiroRunner Training",  
    "type": "simulation",  
    "cfg": {  
        :  
    },  
    "simulation": {  
        "maxJobDurationInSeconds": 5400,  
        :  
    }  
},  
},
```

シミュレーション時間を延長するときはこれに対してかかる課金についても注意してください。AWS RoboMaker のシミュレーションにかかる費用は次を参照してください。

<https://aws.amazon.com/jp/robomaker/pricing/>

追加課題 2. 報酬関数、アクションスペースを確認と変更

ここではプログラムの中を少しみていきます。強化学習における、報酬関数、アクションスペースなどをみていきます。またターミナルウィンドウから**ビルド**、**バンドル**を行う方法もみていきます。

AWS RoboMaker 開発環境を開きます。ナビゲーションペイン（ウィンドウ左のファイル選択域）でファイル MeiroRunner -> simulation_ws -> src -> rl_agent-> markov -> environments -> meiro_runner_env.py を選択、ダブルクリックして開きます。このファイルには強化学習における「環境」が実装されています。ここではエージェントからの指示に従いロボットを操作し、その結果起こったロボットの変化（LiDAR のセンシングデータの変化など）から報酬を計算し、更新されたステートと報酬をエージェントに返しています。

強化学習についての詳細は利用している強化学習フレームワーク Coach を参考にしてください。

<https://github.com/NervanaSystems/coach>

報酬関数は infer_reward_state 関数の中で実装されています。

- LiDAR センサーがセンシングした周辺の壁までの距離は self.ranges の中に入っています。
self.ranges は 1 度刻みで周辺 360 度の距離をメートル単位で保存する配列です。
- self.ranges から壁までの最短距離を求め min_distance という変数に保存しています。
- 壁に近づきすぎたら(13cm より壁に近い) 壁に衝突したこととして、施行を終了させます
- 壁から十分に離れている (19cm 以上離れている) 場合進行速度を報酬値としています
- 壁に近いところを移動している場合、壁からの距離に応じて報酬を与えています。また壁から離れる方向に移動している場合は追加の報酬を与えています

アクションスペースは 5 つの連続性のない動作として定義しており、5 つの動作は次の通り定義されています。

TurtleBot3MeiroRunnerDiscreteEnv クラスの step 関数

```
def step(self, action):
    # Convert discrete to continuous
    if action == 0:  # turn left
        steering = 1.159
        throttle = 0.08
    elif action == 1:  # turn right
        steering = -1.159
        throttle = 0.08
    elif action == 2:  # straight
```

```

        steering = 0
        throttle = 0.1
    elif action == 3: # steer to the left
        steering = 0.6
        throttle = 0.06
    elif action == 4: # steer to the right
        steering = -0.6
        throttle = 0.06
    else: # should not be here
        raise ValueError("Invalid action")

    continuous_action = [steering, throttle]

    return super().step(continuous_action)

```

ここで `action` がエージェントから与えられたアクションの番号、これに対し旋回速度を `steering` で前進するスピードを `throttle` で求めています。求めた値は `super().step(continuous_action)` でこのクラスの親クラスである `TurtleBot3MeiroRunnerEnv` クラスの `step()` 関数に送られ、ここで実際にロボットを操縦しています。

試しに

```

elif action == 2: # straight
    steering = 0
    throttle = 0.1

```

の箇所を

```

elif action == 2: # straight
    steering = 0
    throttle = 0.2

```

としてみます。これは直進時の速度を 2 倍にしています。

修正を保存したらビルドします。

ビルドは今度はターミナルから行ってみましょう。まず、ターミナルウィンドウを一つ開きます。ターミナルをターミナル表示領域の `[+]` ボタンをクリックして、プルダウンメニューを開き `[New Terminal]` を選びます。

```

64 # actions -> steering angle, throttle
65

Immediate (Java) Sim logs (9e9b9f) Colcon Bundle - x
Run Colcon Bundle Command: source /opt/...
Replace Replace All
New File
New Terminal
New Run Configuration
Open Preferences
Recently Closed Tabs
New Immediate Window
Runner: Shell command CWD ENV

```

ed_mes

Reading package lists... done
Building dependency tree... Done
Checking if dependency tarball exists...
Starting >>> meiro_runner_simulation
Finished <<< meiro_runner_simulation [0.42s]
Starting >>> rl_agent
Finished <<< rl_agent [0.07s]
Starting >>> turtlebot3_description_reduced_mesh
loading view [*default*] with rospkg loader
Finished <<< turtlebot3_description_reduced_mesh [0.08s]

Summary: 3 packages finished [17.1s]
Checking if local dependencies have changed since last bundle...
Local dependencies not changed, skipping dependencies update...
Creating bundle archive V2...
Archiving complete!

ターミナルウィンドウで次を実行します。

```
cd MeiroRunner/simulation_ws
colcon build
```

```

.*? aA ⌘ ⌘ ⌘ Find

Immediate (Java) Sim logs (9e9b9f) Colcon Bundle - x bash - "ip-172-31" +
demo:~/environment $ cd MeiroRunner/simulation_ws
demo:~/environment/MeiroRunner/simulation_ws $ colcon build

```

ビルドが成功したら次はバンドルです。バンドルもターミナルから実行できます。

```
colcon bundle
```

```

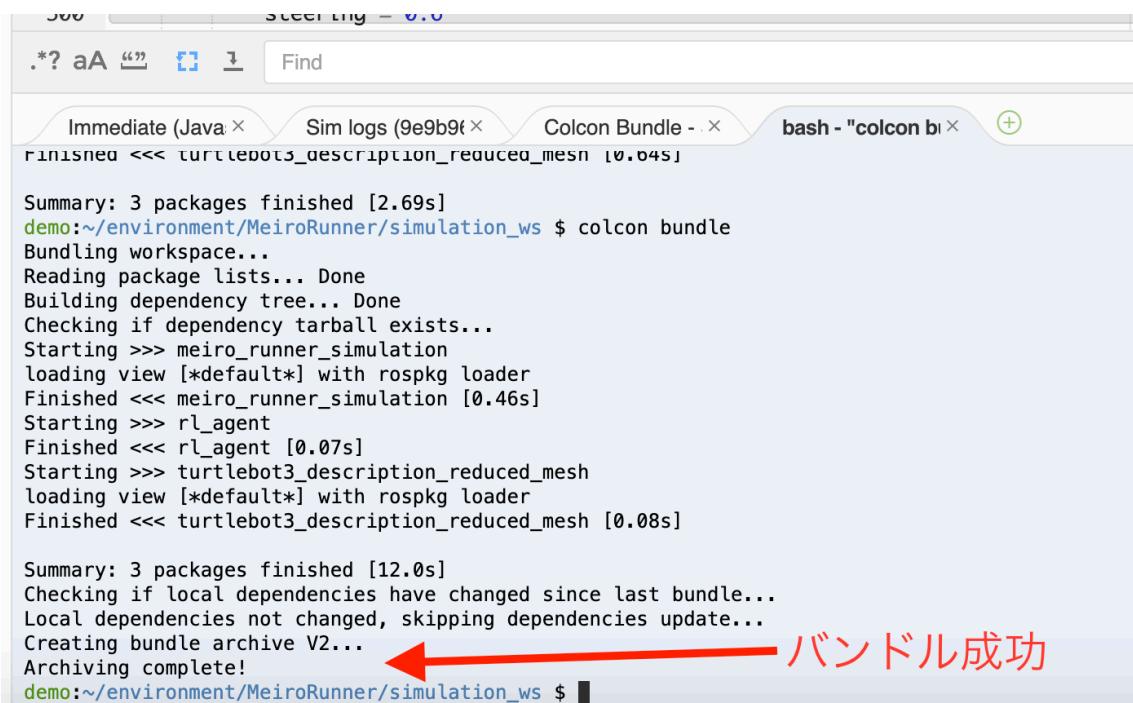
.*? aA ⌘ ⌘ ⌘ Find

Immediate (Java) Sim logs (9e9b9f) Colcon Bundle - x bash - "colcon b" +
demo:~/environment $ cd MeiroRunner/simulation_ws
demo:~/environment/MeiroRunner/simulation_ws $ colcon build
Starting >>> meiro_runner_simulation
Starting >>> rl_agent
Finished <<< rl_agent [1.31s]
Starting >>> turtlebot3_description_reduced_mesh
Finished <<< meiro_runner_simulation [1.67s]
Finished <<< turtlebot3_description_reduced_mesh [0.64s]
Summary: 3 packages finished [2.69s]
demo:~/environment/MeiroRunner/simulation_ws $ colcon bundle

```

ビルト成功

バンドル



```
Summary: 3 packages finished [2.69s]
demo:~/environment/MeiroRunner/simulation_ws $ colcon bundle
Bundling workspace...
Reading package lists... Done
Building dependency tree... Done
Checking if dependency tarball exists...
Starting >>> meiro_runner_simulation
loading view [*default*] with rospkg loader
Finished <<< meiro_runner_simulation [0.46s]
Starting >>> rl_agent
Finished <<< rl_agent [0.07s]
Starting >>> turtlebot3_description_reduced_mesh
loading view [*default*] with rospkg loader
Finished <<< turtlebot3_description_reduced_mesh [0.08s]

Summary: 3 packages finished [12.0s]
Checking if local dependencies have changed since last bundle...
Local dependencies not changed, skipping dependencies update...
Creating bundle archive V2...
Archiving complete! バンドル成功
demo:~/environment/MeiroRunner/simulation_ws $
```

"Archiving Complete!" のような表示が現れるとバンドル成功です。変更した内容でシミュレーションを実行する準備ができました。シミュレーションを開始して、ロボットの直進速度が速くなったことを確認します。

メモ :

colcon build は ROS アプリケーションのビルドツールの一つです。

<https://colcon.readthedocs.io/en/released/>

また、colcon bundle は colcon のビルドツールにプラグインされる追加機能で、これもオープンソースで公開されています。

<https://github.com/colcon/colcon-bundle>

つまりビルド、バンドルの開発フロー自体は AWS RoboMaker の開発環境以外でも構築可能で、このように AWS RoboMaker 以外でビルド、バンドルされた生成物を RoboMaker のシミュレーションやフリート管理に持ってきて、実行することも可能です。

この場合、シミュレーションやフリート管理の呼び出しもコマンドラインからまたは自動処理で行いたいかもしれません。

AWS CLI / API からの RoboMaker の各機能へのアクセス方法は次を参考にしてください。

<https://docs.aws.amazon.com/cli/latest/reference/robomaker/index.html>

https://docs.aws.amazon.com/robomaker/latest/dg/API_Reference.html

4. リソースの削除

ハンズオンが全て終了したらリソースを削除しましょう。生成されたリソースに関する情報は開発環境 MeiroRunner ディレクトリ配下の ws_settings.yaml ファイルの中に記述されています。

bucket_name : ハンズオン用に作成された S3 バケットです。

<https://console.aws.amazon.com/s3> よりバケットを確認し、不要であれば削除します。

iam_policy, iam_role : ハンズオン用に作成された IAM ロール、IAM ポリシーです。

<https://console.aws.amazon.com/iam/> よりそれぞれ確認し、不要であれば削除します。

simulation_app_name: シミュレーションアプリケーションの名前。AWS RoboMaker のコンソール、「シミュレーションアプリケーション」の中の一覧から確認、不要であれば削除します。

最後に AWS RoboMaker 開発環境をこれ以上参照しない場合、削除します。

<http://console.aws.amazon.com/robomaker> にて開発環境の一覧を表示、削除したい開発環境を選んだ後、[編集]ボタンから削除を行います。

S3 と AWS RoboMaker 開発環境については、削除せずにそのままにしておくと無料枠を超える場合があります。AWS RoboMaker の利用に関連して発生する費用について詳しくは次を確認してください。

<https://aws.amazon.com/jp/robomaker/pricing/>