LimeDemo における、Behavior Tree の記述

momoi.org

1.　まえがき

LimeDemo/LimeSimulDemo では、Python によって実装された Behavior Tree を使用することにより、アプリケーションのカスタマイズを容易にしている。ここでは、新たに Behavior Tree を用いてアプリケーションを構築する際に必要な情報について述べる。

2.　XML による Behavior Tree の記述

2.1.　記法

ここで用いる Behavior Tree の記述は、ROS 標準の BehaviorTree.CPP における記述と類似である。ただし、使用する Behavior は、py_trees が供給するものである。

<root>
　<BehaviorTree ID=<behavior tree id>>
--- 本体の記述
　</BehaviorTree>
</root>

の形式をとる。通常の Behavior は、
<<behavior type> name=<behavior name> args>
によって記述する。name の指定が不可欠である。args は、Python による behavior class 記述におけるコンストラクタの引数に対応し、各々は、
<arg name> = <arg value>
の形式をとる。arg value は、通常の文字列、もしくは、{}、あるいは、[]で囲まれた文字列が指定できる。{}で囲まれた文字列は、black board のタグ名であり、black board を参照した値が実引数となる。[]で囲まれた文字列は、Python プログラムとして評価される。したがって、たとえば、[True]は、Python の boolean 値の True、[1]は、Python の整数値 1 が実引数となる。

2.2.　使用できる control flow node

Behavior node は、実行を制御する control flow node と、実際のロボット動作を実現する execution node がある。ここでは、control flow node の実装状況について述べる。Control flow node は、さらに、複数の子ノードを持つ composites と、単一 node の状態を修飾する Decorator に分類される。

○デフォルトで使用できる Composites

- Selector
  複数の子ノードのどれかのステータスが Success になるまで優先度順に実行する. 最後まで Success にならなければ、全体が Failure。どれかが Success になれば Success
- Sequence
  複数の子ノードのどれかのステータスが Failure になるまで順番に実行する。Failure になった時点で、全体も Failure。最後まですべての子ノードが Failure にならなければ全体が Success。
- Parallel
  複数の子ノードを同時に実行して、どれかが Success になったら終了する。すべての子ノードが Failure になったら全体も Failure になる

○デフォルトで使用できる Decorators

- Condition
  子ノードが指定されたステータスになるまで繰り返す
- Count
  子ノードのステータス問い合わせ回数をカウントする
- EternalGuard
  指定された条件判断を行って子ノードを実行するかどうか決める
- Inverter
  子ノードのステータスの Success と Failure を逆転させる
- OneShot
  子ノードに 1 回だけステータス問い合わせを行う
- Repeat
  子ノードの実行を n 回繰り返す
- Retry
  子ノードのステータスが Failure で終わった場合、再実行する。再実行回数の上限を指定する
- StatusToBloackboard
  子ノードのステータスを blackboard に記入する
- Timeout
  子ノードの実行時間を制限する。制限時間になると、強制的に子ノードの実行を止めてステータスを Failure にする
- FailureIsRunning
  子ノードのステータスが Failure の場合、Running に変換する

- FailureIsSuccess
  子ノードのステータスが Failure の場合、Success に変換する
- RunningIsFailure
  子ノードのステータスが Running の場合、Failure に変換する
- RunningIsSuccess
  子ノードのステータスが Running の場合、Success に変換する
- SuccessIsFailure
  子ノードのステータスが Success の場合、Failure に変換する
- SuccessIsRunning
  子ノードのステータスが Success の場合、Running に変換する

詳細は、py_trees のドキュメント(https://py-trees.readthedocs.io/en/release-2.3.x/index.html)を参照されたい。

3. Lime Demo における実装

3.1. 実装方法

pytwb に登録されたワークスペース ディレクトリの下のディレクトリ、src/<package name>/<package name>/trees 下の Behavior Tree ファイルはすべて同時にパースされ、一種のデータベースを構成する。これによって、他の Behavior Tree の記述を部品として階層的に呼び出して利用することが可能である。

Lime Demo では、ワークスペース ディレクトリとして、/root/pytwb_ws を使用している。

Python で新規に記述された behavior は、src/<package name>/<package name>/behavior ディレクトリ下に置く必要がある。また、behavior のタイプ名は、クラス名がそのまま利用される。さらに、behavior の記述であることを示すアノテーションを付ける必要がある。したがって、下記のような記述になる。

```
import py_trees
from pytwb.common import behavior

@behavior
class Commander(py_trees.behaviour.Behaviour):
        pass
```

クラス本体の記述方法は、これまでの py_trees, py_trees_ros が要求する方針と同一である。

また、actor として実装されてる実行コードをそのまま Behavior としても使用する場合は、src/<package name>/<package name>/lib/actor_bt.py にラッピングクラス ActorBT が実装されており、これを利用して簡単に定義できるようになっている。

```
from lib.actor_bt import ActorBT

@behavior
class Adjust(ActorBT):
    desc = 'adjust arm angle'   # ヘルプメッセージとして表示される

    def __init__(self, name, node):
        super().__init__(name, 'ad')   # 使用する actor 名’ad’を指定
```

だけの記述でよい。

3.2.　実装されている Behavior

○ blackboard.py にて実装

- SetBlackboard
  引数を blackboard にセット
- ShowBlackboard
  指定された名前の blackboard の内容を表示

○manipulator.py にて実装

- Adjust
  グリッパ角度を、コーラ缶に合うように調整
- Fit, Fit2
  コーラ缶位置まで下げられたグリッパ角度を、左右均等になるように調整
- Pick
  グリッパを閉じて、対象物をピッキング
- Place
  把持中の対象物を、地面に設置
- Open
  グリッパ開
- ArmHome
  アームを標準姿勢にする
- Close

グリッパ閉

○ migration.py

・ Approach
ビジュアルフィードバックによって、コーラ缶に接近する（近距離用）

・ Mini_Walk
短距離を直進

・ Shift, ReachCoke
アームが下りた状態でビジュアルフィードバックを使用してコーラ缶に接近

・ Face
ロボット本体を転回させてコーラ缶に正対

・ CheckCoke
視野内にコーラ缶があることを確認。なければ、転回して再度確認する

○ navigation.py

・ GetLocation
Blackboard の pose_list 要素から移動先候補のリストを取得し、その先頭にある移動先を、bloackboard の target_pose にセット

・ GoToPose
Blackboard の target_pose にセットされている移動先に移動

○ setlocation.py

・ SetLocations
旧コードにつき不使用

○setwatchlocations.py

・ SetWatchLocations
vector_map ライブラリを用い、地図情報を分析して、探索すべき地点の候補リストを作成し、blackboard の watch_list にセットする

・ GetWatchLocation
Bloackboard の watch_list から、探索先の候補を一か所取得する

・ GetGlancedLocation
Blackboard の glanced_point から、コーラ缶を見かけた座標を取得

・ ScheduleDestination
コーラ缶の座標が確定した後に、ピッキング動作を行うために適した座標を計算

・ GetFoundPoint

コーラ缶の座標確定が失敗した場合、再試行する場所を計算

○ simvision.py

・ Viewer
カメラ画像を検出して自身の state を変更する

・ LookForCoke
カメラ画像から、コーラ缶の座標を発見し、blackboard の glanced_point にセット

・ Watch
コーラ缶座標を繰り返し計測し、より精度の高い座標値を取得する

○ tools.py

・ ObjectLocation
対象物座標の表示

・ Generic
指定された actor を呼び出す

・ Sleep
一定時間の動作停止

・ Pause
キー入力があるまで動作停止