Behavior Treeツールと地図ライブラリによるアプリケーション開発環境の提案

田胡　和哉, 三田 渉, 松岡 丈平
(東京工科大学)

ROS新時代：
“ブラックボックス”としてのROSを目指そう

- ROSの一般活用の広がり
  - 実業務で利用可能な安価なROS対応ロボットが揃いつつある
    - 近い将来、一般業務への適用事例が爆発的に増加する可能性
- ”特別なもの“から”普通に使われるもの“への移行
  - 現状では、（うんざりするぐらい）多くのことを知らないと使えない
  - 他分野の人が使えるROSの創成
    - 今後は、ROSを熟知している人が使うとは限らない
    - 他の業務アプリやAIとの連携の容易化
    - カスタマイズの容易化

内容
（https://momoi.org/?yada_wiki=momoi-org-ros関連プロジェクト）

- vector_map
  GitHub　https://github.com/RobotSpatialCognition/vector_map
  - SLAM出力をベクトルデータ化
    - 直線や曲線で地図を構成し、領域を認識できるようにする
  - CADデータ等、環境中のオブジェクト定義データを張り込んで使用

  ⇒ロボットの“認知機能”の強化
- pytwb　(py_trees workbench)
  GitHub　https://github.com/momoiorg-repository/pytwb
  - プランニング機能と開発ツール
    - PythonによるBehavior Treeの実行環境
      - XMLによるBehavior Tree記述を可能に
    - ビルド作業の省略/簡略化ツール

  ⇒ ロボットのプランニング機能とアプリケーション連携機能の強化

内容

- vector_map
  GitHub　https://github.com/RobotSpatialCognition/vector_map
  - SLAM出力をベクトルデータ化
    - 直線や曲線で地図を構成し、領域を認識できるようにする
  - CADデータ等、環境中のオブジェクト定義データを張り込んで使用

  ⇒ロボットの“認知機能”の強化
- pytwb
  GitHub　https://github.com/momoiorg-repository/pytwb
  - プランニング機能と開発ツール
    - Pythonによるbehavior treeの実行環境
      - XMLによるBehavior Tree記述を可能に
    - ビルド作業の省略/簡略化ツール

  ⇒ ロボットのプランニング機能とアプリケーション連携機能の強化

ROSにおけるBehavior Tree

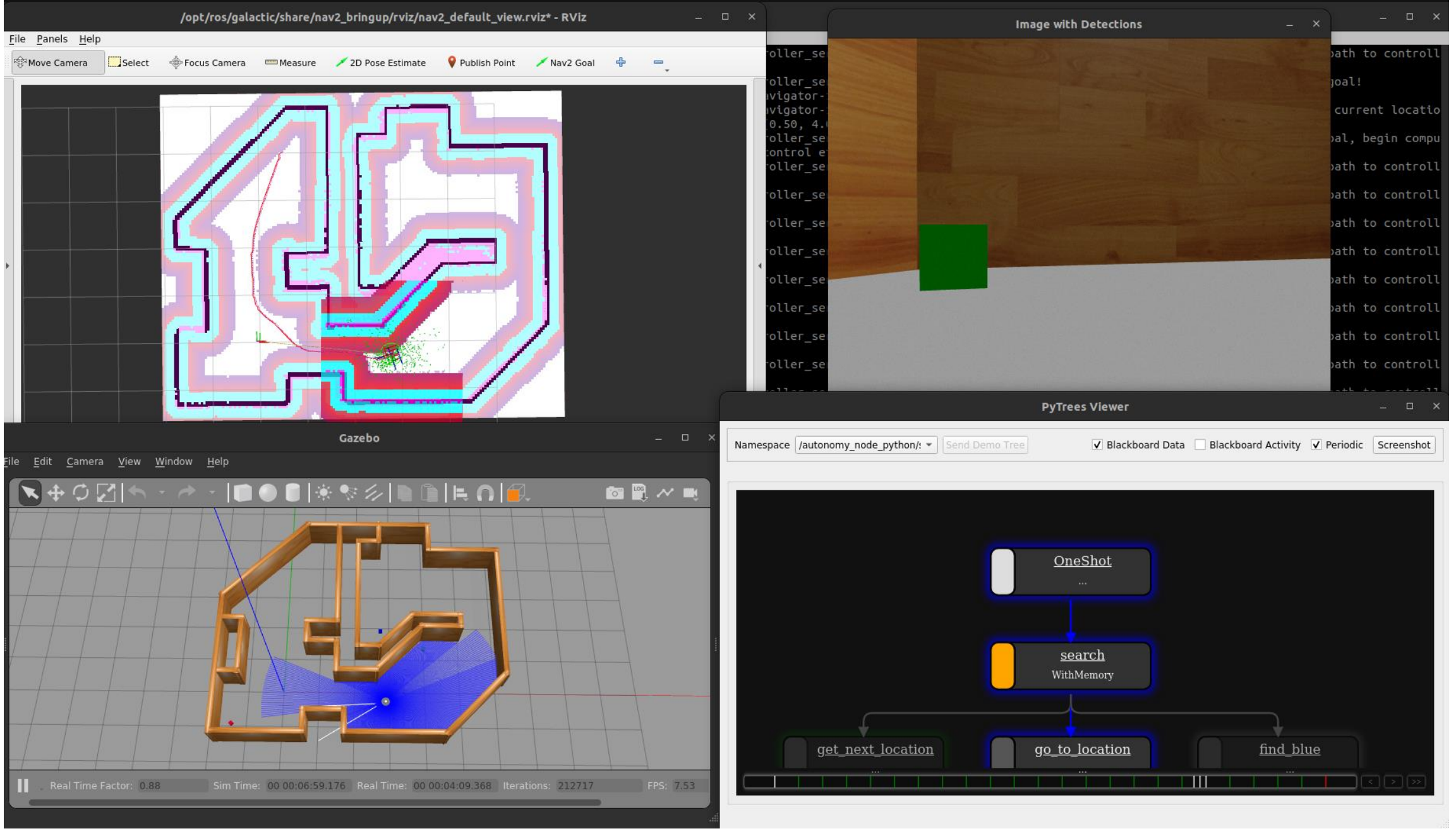
- BehaviorTree.CPPが本家
  - タスク単位(Behavior Node)の実行順を記述
  - 主にNav2実装で利用されている
- Pythonベースのpy_trees, py_trees_rosは別もの
  - BehaviorTree.CPPと相互乗り入れ不可
  - XMLによるツリー記述機能無し
  - Pythonなので、使うのは簡単

Behavior Treeのメリット

- ROSの詳細を知らなくてもROSアプリケーションが書ける
  - 一種のローコード環境
- ソフトウェアの部品化が促進される
  - Behavior Node = 部品
  - 分散開発、再利用が可能
- AIや他の業務アプリとの連携が容易になる
  - Behavior Nodeは、非同期処理と同期処理の橋渡しをしている
  - Behavior Nodeは、AIや他の業務アプリとの通信を行うブリッジとして利用できる

デモ開発

- ベースとなる実装
  - turtlebot3_behavior_demos(sea-bass)
    - GitHub　https://github.com/sea-bass/turtlebot3_behavior_demos
    - 迷路内のturtlebot3が、移動しながら立方体のオブジェクトを探索する
    - C++とPython双方の実装が含まれており、比較できる
- pytwbを用いてこれを拡張



出典：　https://github.com/sea-bass/turtlebot3_behavior_demos

拡張デモ(pytwb_demo)の開発

- コーラ缶のピッキングを想定した探索動作
- 効率のよい迷路探索場所の自動決定
  - vector_mapを用いたプランナの実装
  - カメラ探索と移動の同時実行
- pytwbとPythonを用いた開発
  - PythonであってもBehavior TreeをXMLで記述
  - ビルドなしの一発実行(direct run)による開発効率の改善

tutorial3.xml

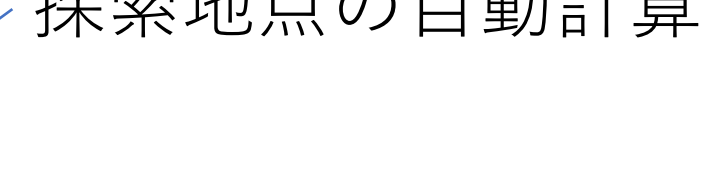
```
<root>
  <BehaviorTree ID="Tutorial3">
  <Sequence name="main">
    <Commander name="commander"/>
    <SetLocations name="set_locations"/>
    <Retry  name="find_loop" num_failures="[10]">
      <Sequence name="seek_target">
        <Parallel name="seek_and_go" policy="SuccessOnOne">
          <LookForCoke name="look_coke"/>
          <Sequence name="search_location">
            <GetLocation name="get_loc"/>
            <GoToPose name="go_to_loc"/>
          </Sequence>
        </Parallel>
        <SetWatchLocations name="set_location" />
      </Sequence>
    </Retry>
  </Sequence>
  </BehaviorTree>
</root>
```

tutorial3.xml

```
<root>
  <BehaviorTree ID="Tutorial3">
  <Sequence name="main">
    <Commander name="commander"/>
    <SetLocations name="set_locations"/>
    <Retry  name="find_loop" num_failures="[10]">
      <Sequence name="seek_target">
        <Parallel name="seek_and_go" policy="SuccessOnOne">
          <LookForCoke name="look_coke"/>
          <Sequence name="search_location">
            <GetLocation name="get_loc"/>
            <GoToPose name="go_to_loc"/>
          </Sequence>
        </Parallel>
        <SetWatchLocations name="set_location" />
      </Sequence>
    </Retry>
  </Sequence>
  </BehaviorTree>
</root>
```

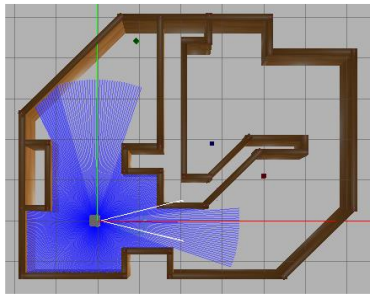
tutorial3.xml

```
<root>
  <BehaviorTree ID="Tutorial3">
  <Sequence name="main">
    <Commander name="commander"/>
    <SetLocations name="set_locations"/>
    <Retry  name="find_loop" num_failures="[10]">
      <Sequence name="seek_target">
        <Parallel name="seek_and_go" policy="SuccessOnOne">
          <LookForCoke name="look_coke"/>
          <Sequence name="search_location">
            <GetLocation name="get_loc"/>
            <GoToPose name="go_to_loc"/>
          </Sequence>
        </Parallel>
        <SetWatchLocations name="set_location" />
      </Sequence>
    </Retry>
  </Sequence>
  </BehaviorTree>
</root>
```

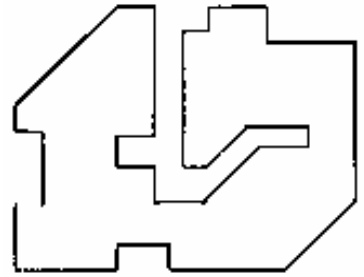


```
<SetLocations                    name="set_locations"/>
```

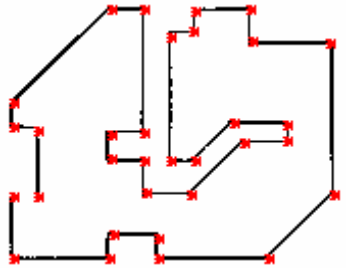
地図を分析して、探索場所を自動的に検出



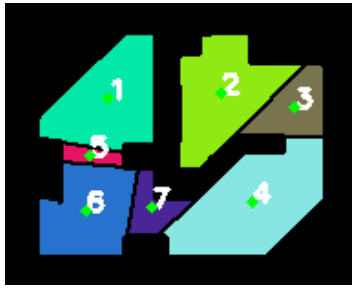
もとの地形



SLAMが作成した地図



角点と順番の自動検出



領域の自動分割
⇒　探索場所設定

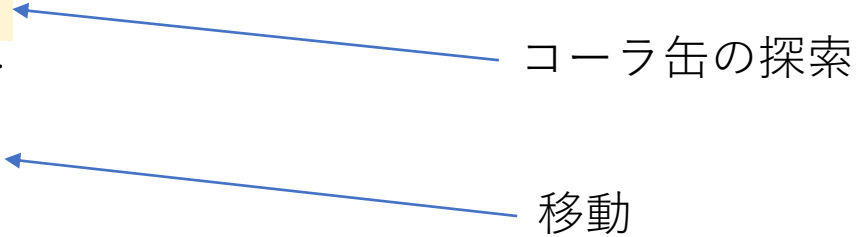
tutorial3.xml

```
<root>
  <BehaviorTree ID="Tutorial3">
  <Sequence name="main">
    <Commander name="commander"/>
    <SetLocations name="set_locations"/>
    <Retry  name="find_loop" num_failures="[10]">
      <Sequence name="seek_target">
        <Parallel name="seek_and_go" policy="SuccessOnOne">
          <LookForCoke name="look_coke"/>
          <Sequence name="search_location">
            <GetLocation name="get_loc"/>
            <GoToPose name="go_to_loc"/>
          </Sequence>
        </Parallel>
        <SetWatchLocations name="set_location" />
      </Sequence>
    </Retry>
  </Sequence>
  </BehaviorTree>
</root>
```

見つかるまで10回再試行

tutorial3.xml

```
<root>
  <BehaviorTree ID="Tutorial3">
  <Sequence name="main">
    <Commander name="commander"/>
    <SetLocations name="set_locations"/>
    <Retry  name="find_loop" num_failures="[10]">
      <Sequence name="seek_target">
        <Parallel name="seek_and_go" policy="SuccessOnOne">
          <LookForCoke name="look_coke"/>
          <Sequence name="search_location">
            <GetLocation name="get_loc"/>
            <GoToPose name="go_to_loc"/>
          </Sequence>
        </Parallel>
        <SetWatchLocations name="set_location" />
      </Sequence>
    </Retry>
  </Sequence>
  </BehaviorTree>
</root>
```



tutorial4.xml – 完成

```
<root>
  <BehaviorTree ID="tutorial4">
    <Sequence name="main">
      <tutorial3 />
      <GoToPose name="revisit_coke"/>
      <Watch name="watch_coke"/>
      <ScheduleDestination name="schedule_final_target"/>
      <GoToPose name="reach"/>
      <Viewer name="final_view" mode="one_shot" />
    </Sequence>
  </BehaviorTree>
</root>
```

他のBTの取り込み、再利用

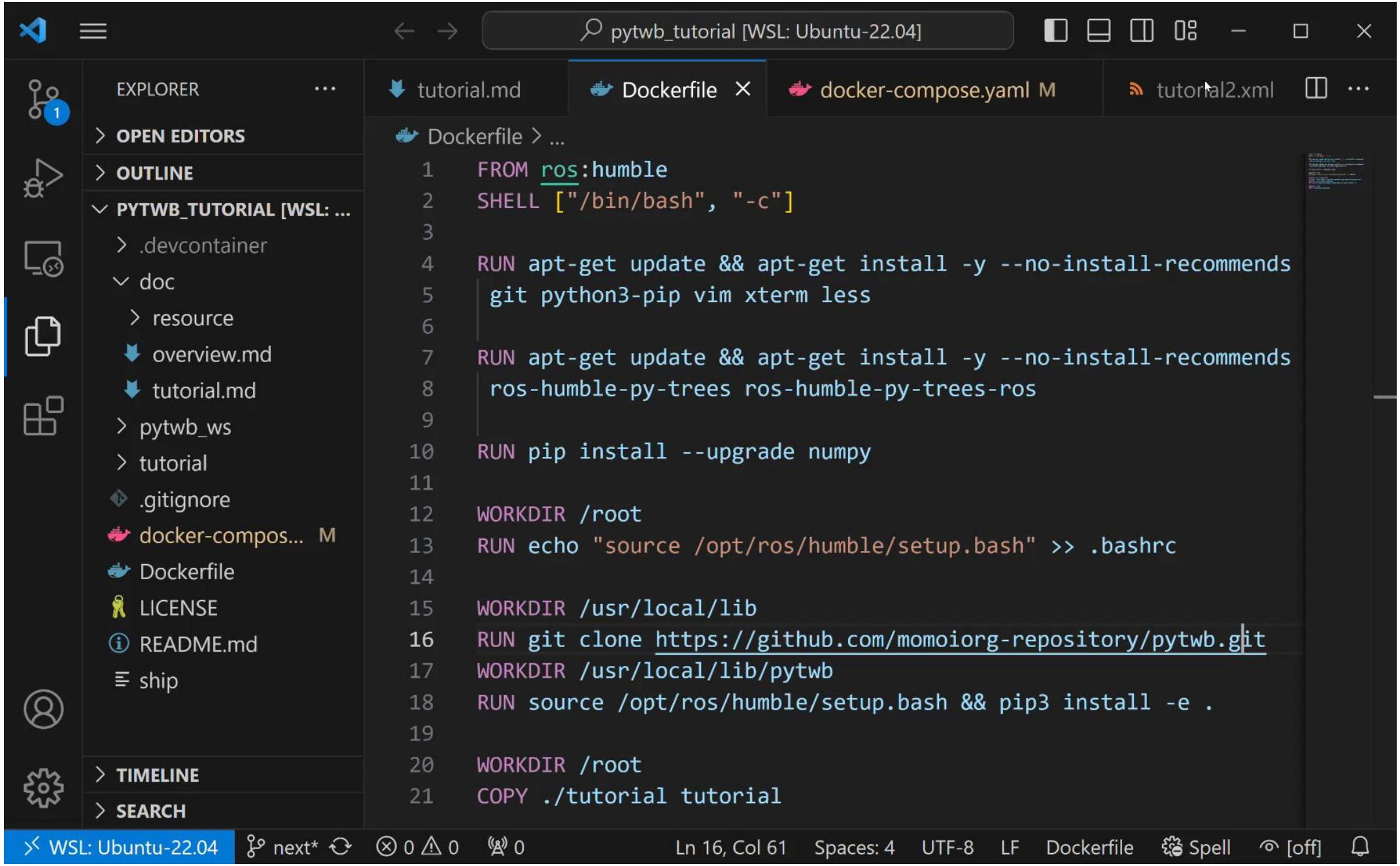
vector_mapを利用した、コーラ缶のピッキングに適した場所への移動


(x=183, y=45) ~ R:83 G:63 B:44

pytwbとは

- Dockerを用いたROSアプリケーション開発環境
- PythonベースのBehavior Tree実行
  - XML記述のパース
  - Pythonモジュールの自動再ロードによるビルドなし実行
- パッケージが依存するソフトウェアの追跡、管理
- Dockerfileの半自動生成によるプロジェクト定義
- Behavior Tree単位でのリポジトリ実装と再利用支援

WSL上での開発例


pytwb_tutorial [WSL: Ubuntu-22.04]
EXPLORER
OPEN EDITORS
OUTLINE
PYTWB_TUTORIAL [WSL: ...
.devcontainer
doc
resource
overview.md
tutorial.md
pytwb_ws
tutorial
.gitignore
docker-compos... M
Dockerfile
LICENSE
README.md
ship
TIMELINE
SEARCH
tutorial.md
Dockerfile
docker-compose.yaml M
tutorial2.xml
Dockerfile > ...
1 FROM ros:humble
2 SHELL ["/bin/bash", "-c"]
3
4 RUN apt-get update && apt-get install -y --no-install-recommends
5 git python3-pip vim xterm less
6
7 RUN apt-get update && apt-get install -y --no-install-recommends
8 ros-humble-py-trees ros-humble-py-trees-ros
9
10 RUN pip install --upgrade numpy
11
12 WORKDIR /root
13 RUN echo "source /opt/ros/humble/setup.bash" >> .bashrc
14
15 WORKDIR /usr/local/lib
16 RUN git clone https://github.com/momoiorg-repository/pytwb.git
17 WORKDIR /usr/local/lib/pytwb
18 RUN source /opt/ros/humble/setup.bash && pip3 install -e .
19
20 WORKDIR /root
21 COPY ./tutorial tutorial
WSL: Ubuntu-22.04
next*
0 0
0
Ln 16, Col 61
Spaces: 4
UTF-8
LF
Dockerfile
Spell
[off]

pytwbによるDockerfileの生成

- ROSアプリケーションの新しい開発手順の提案
  - 新規アプリの開発時
    1. 開発環境のDocker起動
    2. apt,pip,git等で必要モジュールをインストール
       - 必要モジュール(dependency)を自動記録
    3. プログラム開発と実行
       - Pythonを利用
    4. 最後に、必要モジュールのインストール手順記録によって、管理ファイル生成
       - Dockerfileの生成
       - 正規ビルドに必要なpackage.xml の生成(予定)
         - 正規ビルドを可能にする
         - rosdepも利用可能に
  - 開発したコードをベースに、新規プロジェクトとしてフォークする
    1. 生成されたDockerfileによってdocker起動
       - 次にdockerを起動したときには、必要モジュールは自動的にインストールされる

⇒　開発の過程では、これで十分。最後に正規ビルドする

```
<root>
  <BehaviorTree ID="goto">
    <Sequence name="main">
      <SetBlackboard       name="set_bloackboard"
key="target_pose" value="[(4.0, 0.0, 0.0)]"/>
      <GoToPose name="go_to_loc"/>
    </Sequence>
  </BehaviorTree>
</root>
```

プロジェクト１

Docker定義のブランチ

```
<root>
  <BehaviorTree ID="goto">
    <Sequence name="main">
      <SetBlackboard       name="set_bloackboard"
key="target_pose" value="[(4.0, 0.0, 0.0)]"/>
      <GoToPose name="go_to_loc"/>
    </Sequence>
  </BehaviorTree>
</root>
```

プロジェクト２

```
<root>
  <BehaviorTree ID="goto">
    <Sequence name="main">
      <SetBlackboard       name="set_bloackboard"
key="target_pose" value="[(4.0, 0.0, 0.0)]"/>
      <GoToPose name="go_to_loc"/>
    </Sequence>
  </BehaviorTree>
</root>
```

プロジェクト３

Docker定義のブランチ

```
<root>
  <BehaviorTree ID="goto">
    <Sequence name="main">
      <SetBlackboard       name="set_bloackboard"
key="target_pose" value="[(4.0, 0.0, 0.0)]"/>
      <GoToPose name="go_to_loc"/>
    </Sequence>
  </BehaviorTree>
</root>
```

プロジェクト４

今後の計画

- さらなるツール、環境の充実
  - ３Dの動作環境定義ツール
  - フォグコンピューティング環境
    - エッジサーバの実現
- クラウドを経由した利用環境提供
  - 開発ツール
  - 業務アプリ連携
  - 運用監視

書籍

- 「ROSロボットで学ぶ次世代のIoTアーキテクチャ」コロナ社
  - 10月11日発行
  - アマゾンから予約可能
  - https://www.coronasha.co.jp/np/isbn/9784339029383/