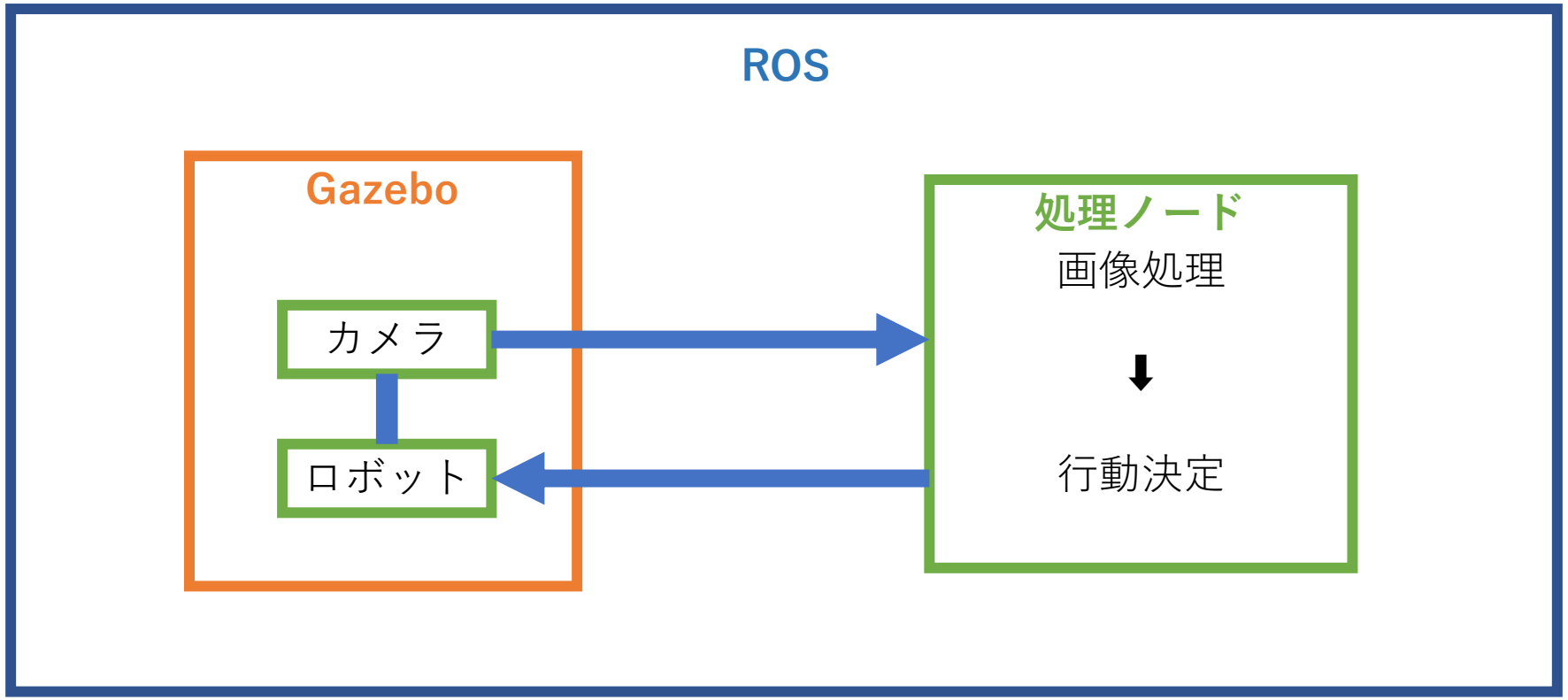
ROSを活用した
コンテスト開発環境

の提供

京都大学情報学研究科　田村　爽

ROSのシミュレーター



(ROSで動く)コンテストシミュレータ公開します！

- と言っても(ROSやったことない人が)使いにくいのは明白

- 使ってもらうための**ハードル**
  - 環境構築
  - ROSの使い方(pub/sub通信の仕組み)
  - シミュレーターの設定
  - そもそもROSでプログラミングすること自体が、無駄（実機にROS組み込む予定ないから）

ということは

- 環境構築を極力減らし
- シミュレーターも最初から構築されており
- ROSの使い方がわからなくてもアルゴリズム部分だけでも検証できる

Dockerの出番！

- のなら良いのでは🤔

Dockerとは？

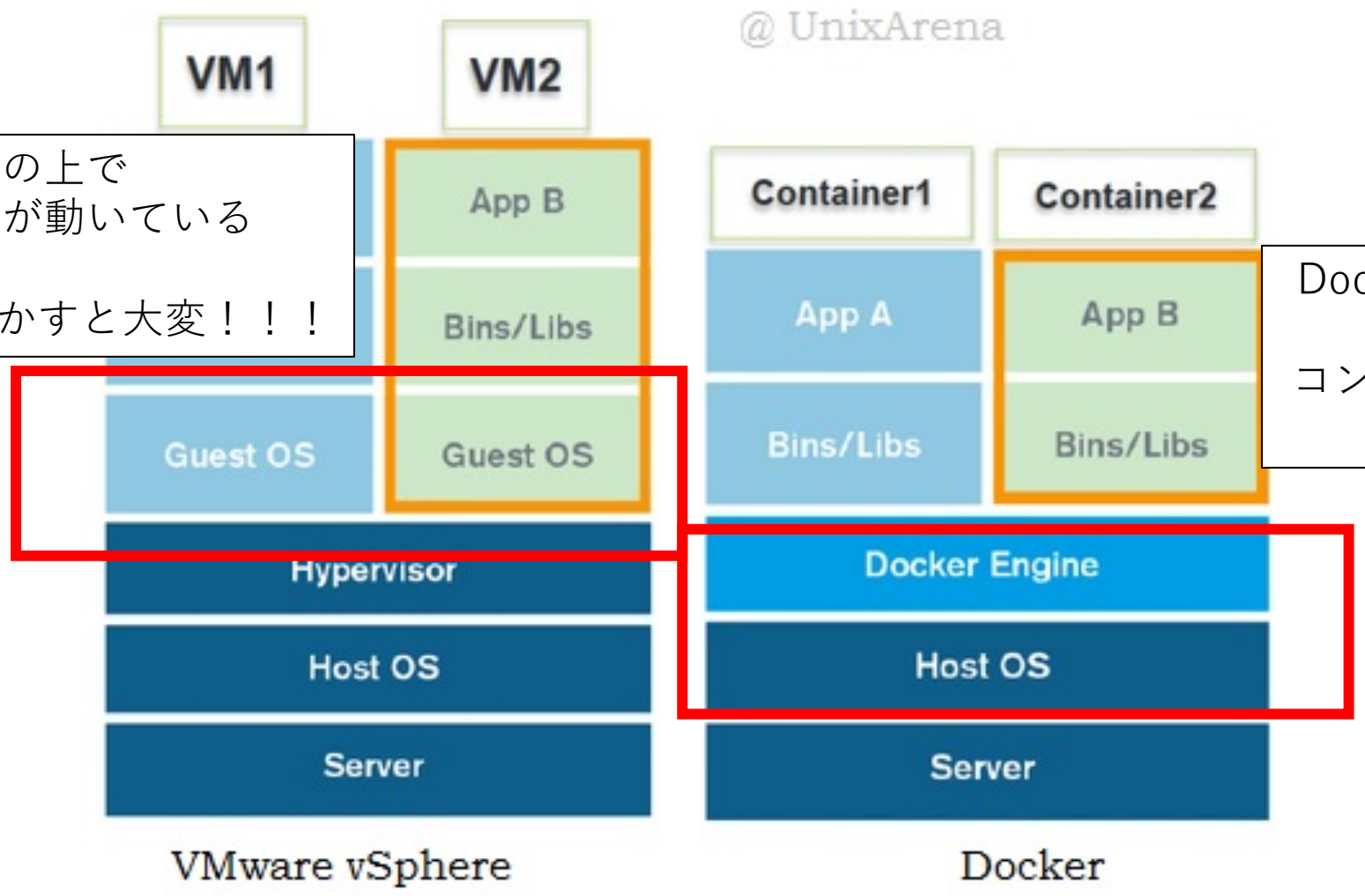
- OSレベルの仮想化環境(コンテナ)を作れるオープンソースソフトウェア
- Linux、macOS、（windows10 Pro）に対応
  - 個人的にはmacでUbuntu動かしたり、Ubuntu16.04で18.04動かしたい時に使う
  - サーバー建てられたり、環境構築グチャグチャになったらちょっと戻したり丸ごと消したりすることもできるので何かと便利

- Dockerfileという、環境構築のためのシェルスクリプトみたいなのがあり、これをGithubから勝手にビルドして公開できる仕組み（DockerHub）がある

Docker と VMの違い

HOST OSの上で
さらにGuest OSが動いている

いろんなGuestOS動かすと大変！！！



Docker Engineがいい感じにしてくれて
コンテナをプロセス単位で管理できる！

- まあ今回はやってることはVMとあんま変わらんので、Dockerインストールできればなんでもいいです

シミュレーター

- DockerHubで公開中
- sudo docker run sousou1/fpt-simulator –p 6080:80
  - 自動的にDockerhubからイメージをDLしてくる

- 中身：仮想デスクトップ＋ROS環境+シミュレーター+オリジナルのモデルデータ＋サンプルコード

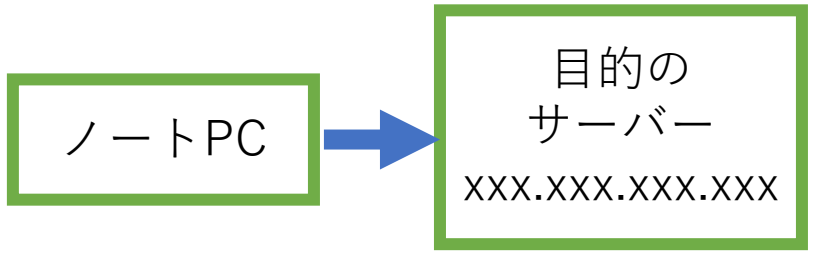
- ノートPCでは重いので、サーバーで動かすのが吉
  - ノートPCで2fps程度、サーバー使えば10fps

備考：サーバーで動かした場合のつなぎ方

- 同ネットワークにいる場合、もしくはグローバルにIP公開されていて直接IPを叩いて繋げられる場合

~/.ssh/config
```
Host lab_server
  Hostname xxx.xxx.xxx.xxx
  Port 22
  User tamura
  LocalForward 6080 xxx.xxx.xxx.xxx:6080
  Identityfile ~/.ssh/id_dsa
```

- ssh lab_server

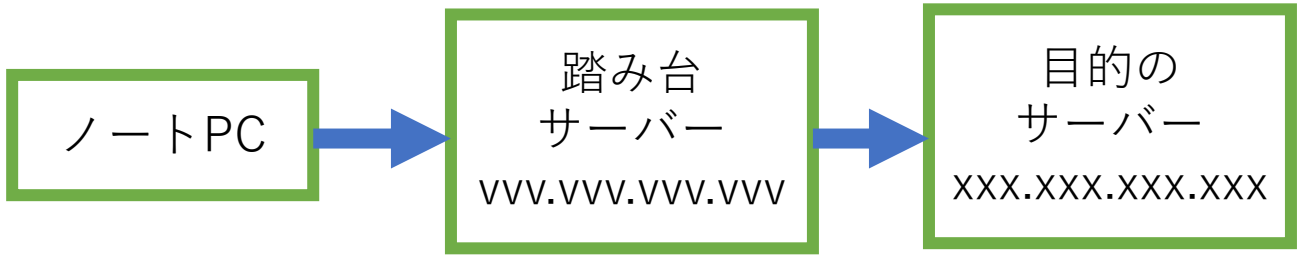


サーバーで動かした場合のつなぎ方2

外部から踏み台サーバー経由でつなげる場合

```
Host outlab
  User tamura
  ProxyCommand ssh vvv.vvv.vvv.vvv nc -w 10 vvv.vvv.vvv.vvv 22
  LocalForward 6080 xxx.xxx.xxx.xxx:6080


Host vvv.vvv.vvv.vvv
  Hostname vvv.vvv.vvv.vvv
  Port 22
  User tamura
  Identityfile ~/.ssh/id_dsa
```

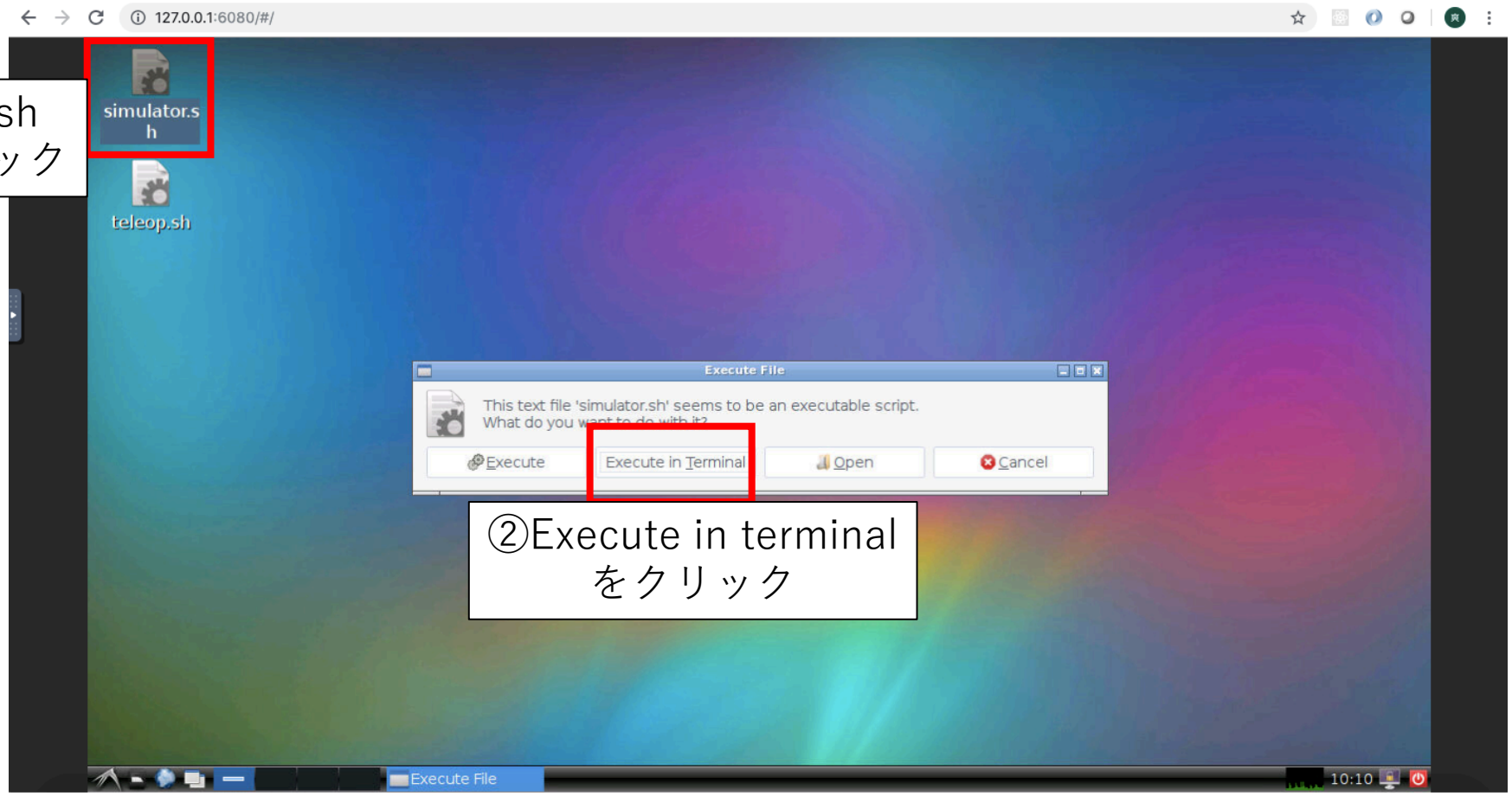


シミュレーター起動方法

- 事前準備 dockerが入ってるPC（or サーバー）
- ssh lab_server（サーバーで動かしてる場合）
- docker run -p 6080:80 -it sousou1/fpt-simulator:latest
- **http://127.0.0.1:6080/#/**
- いきなりデスクトップでてきます

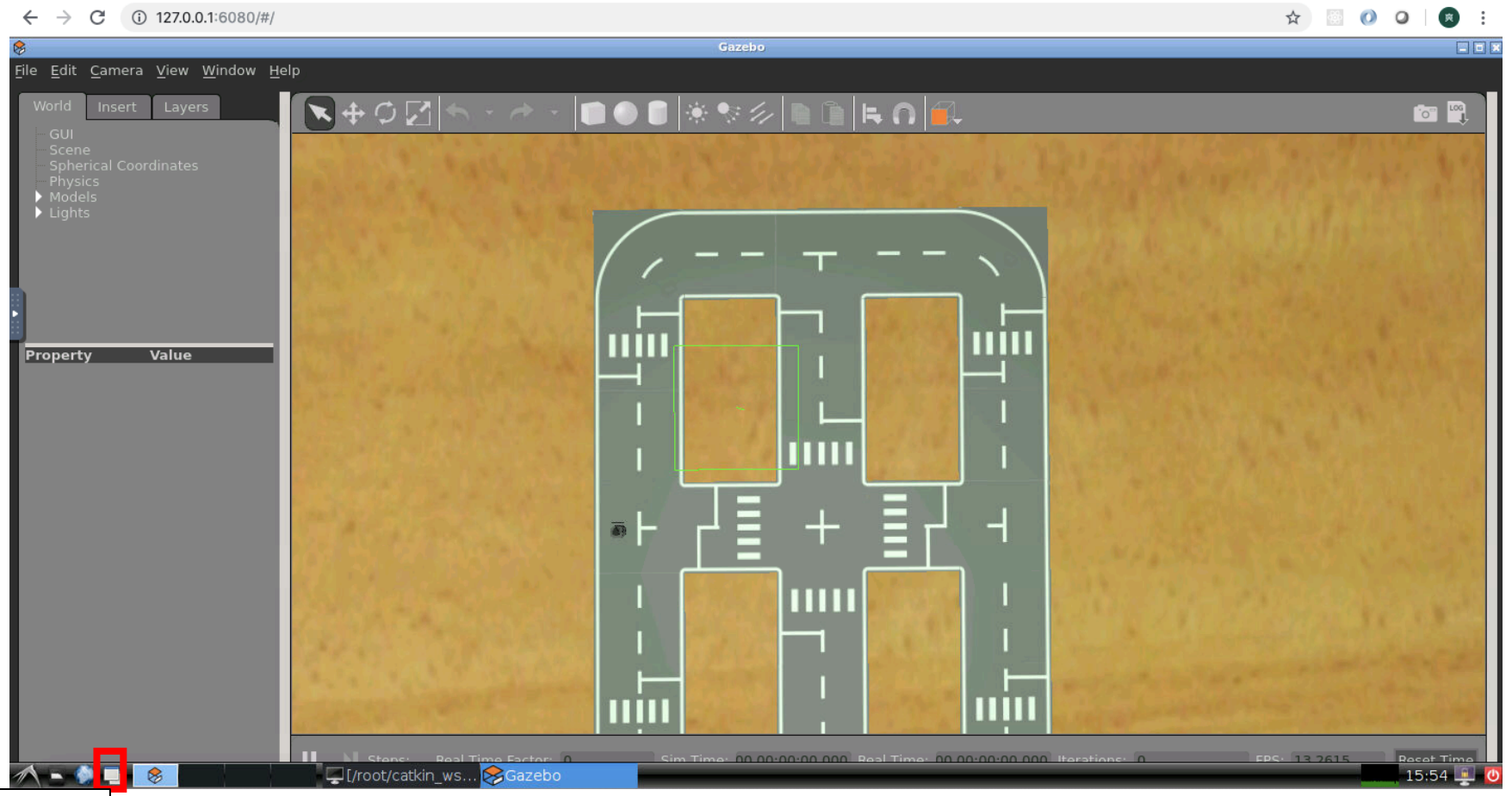
チュートリアル1

①simulator.sh
をダブルクリック



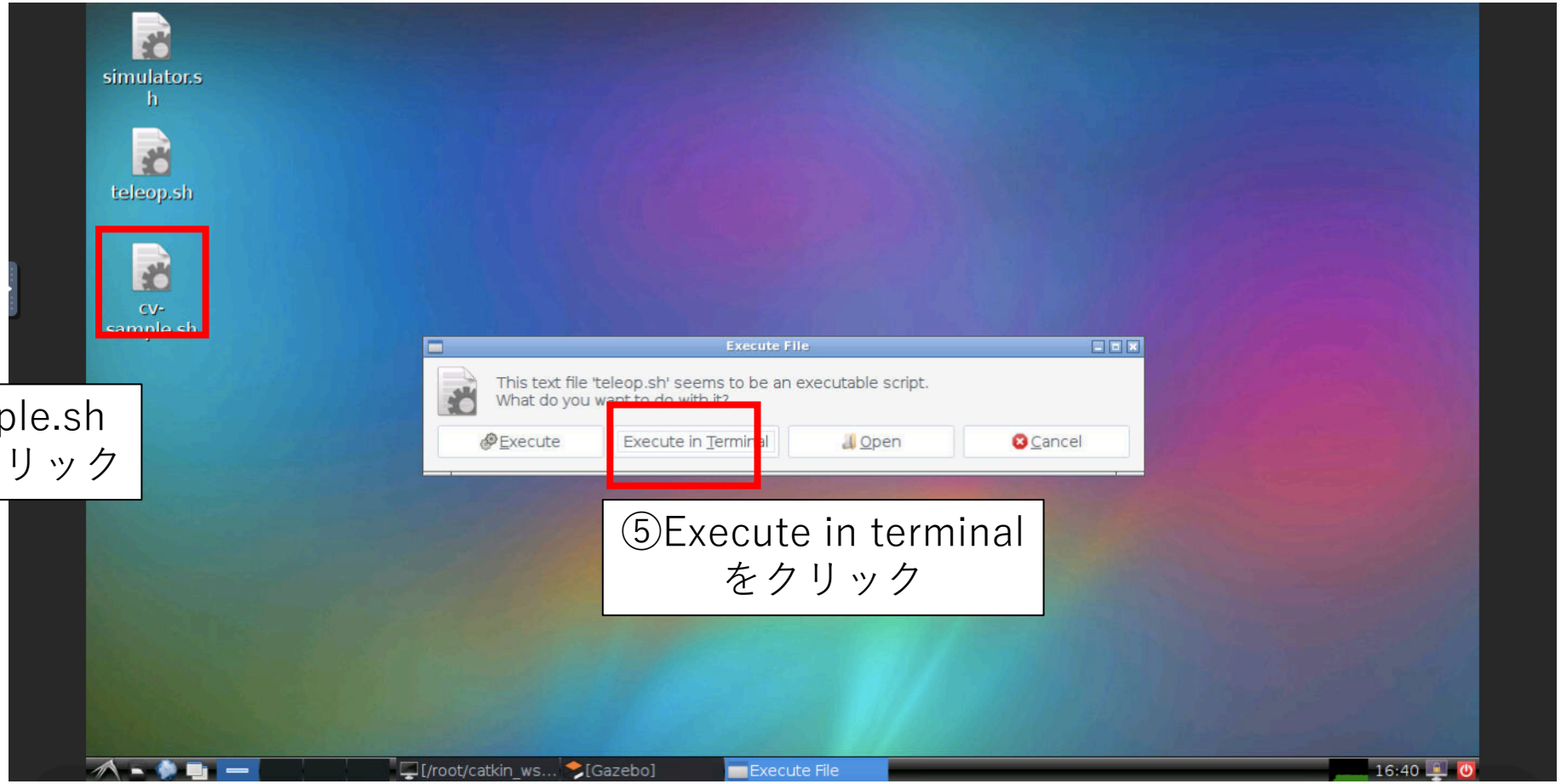
②Execute in terminal
をクリック

チュートリアル2



③最小化する

チュートリアル3

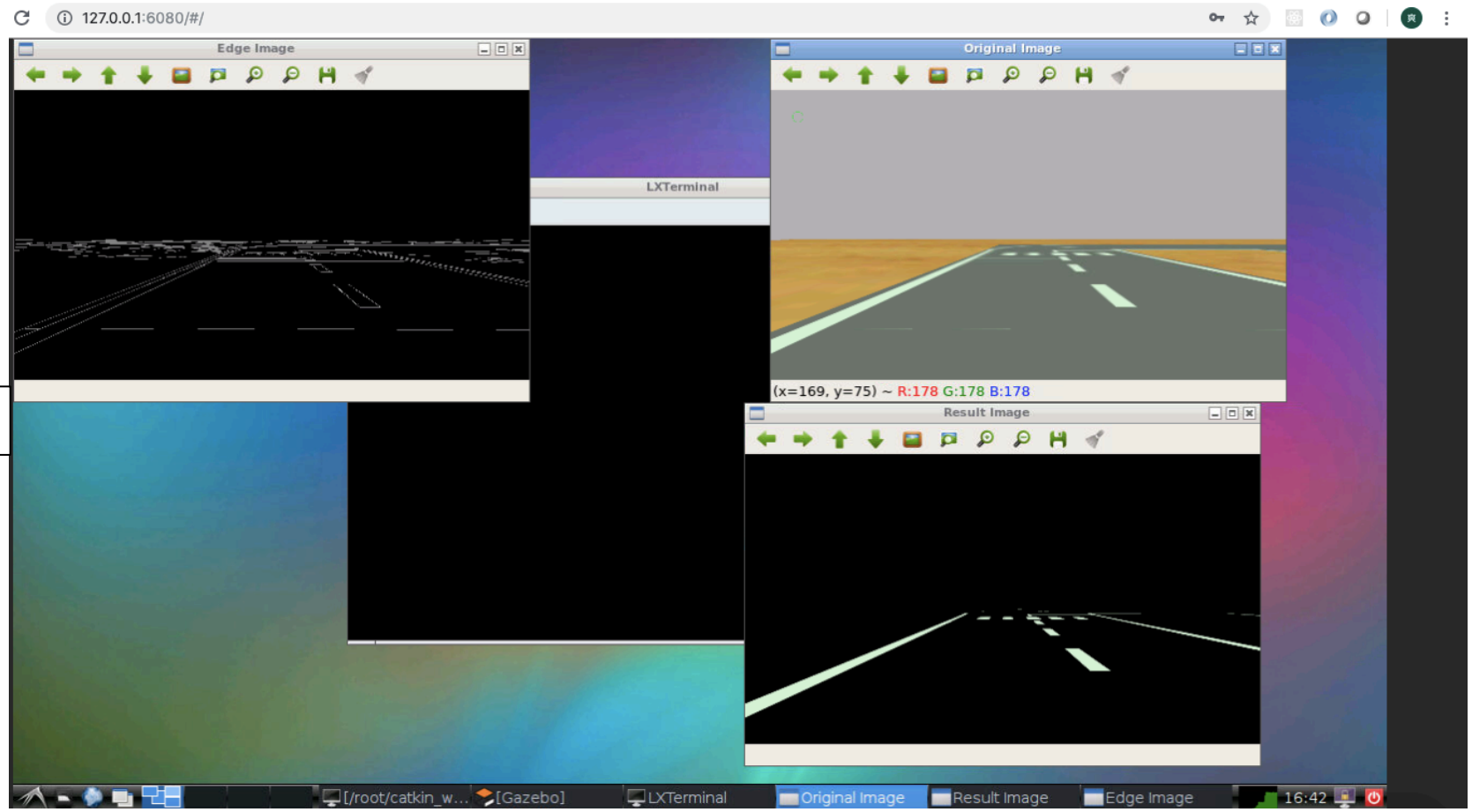


④cv_sample.sh
をダブルクリック

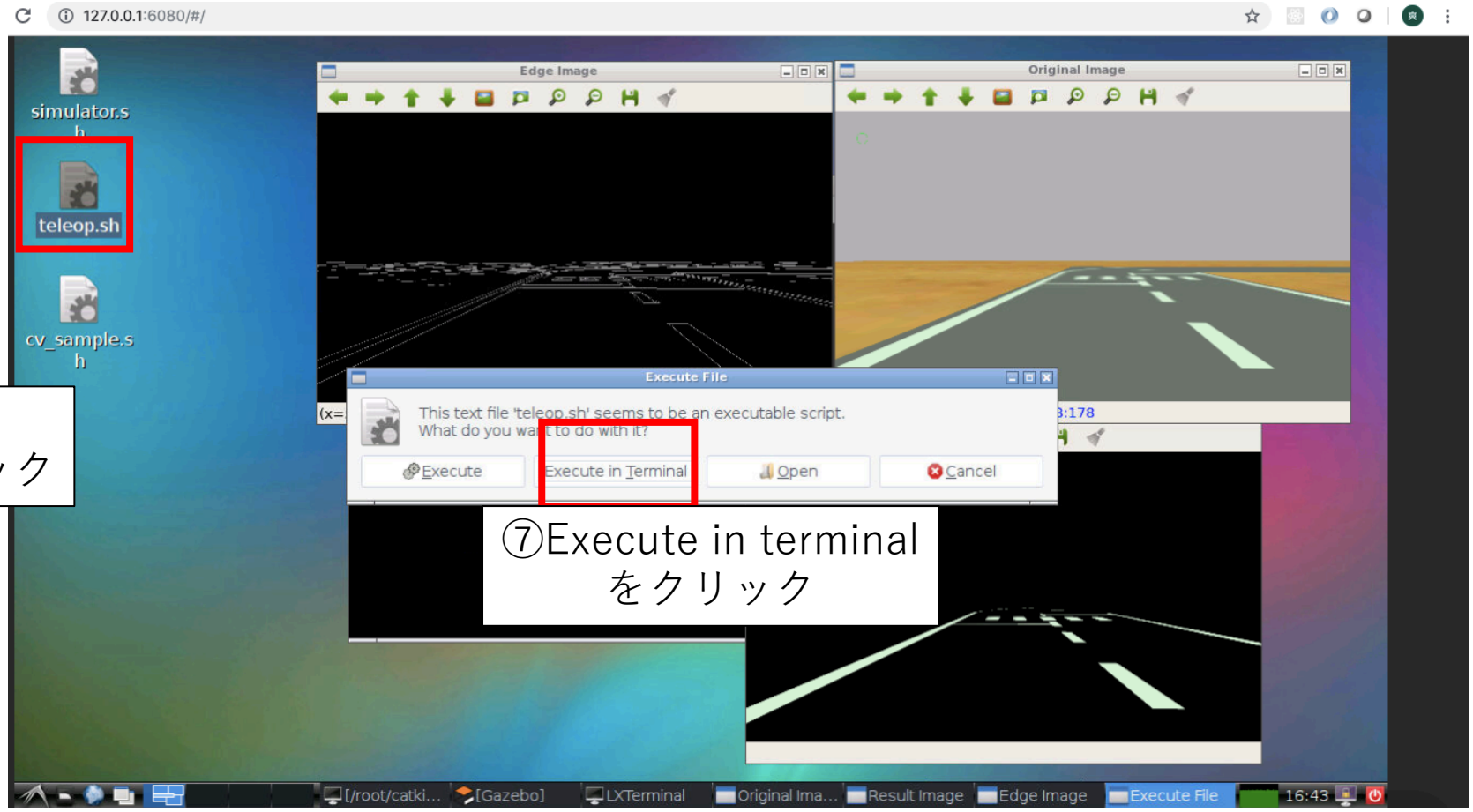
⑤Execute in terminal
をクリック

チュートリアル3

画面が出る



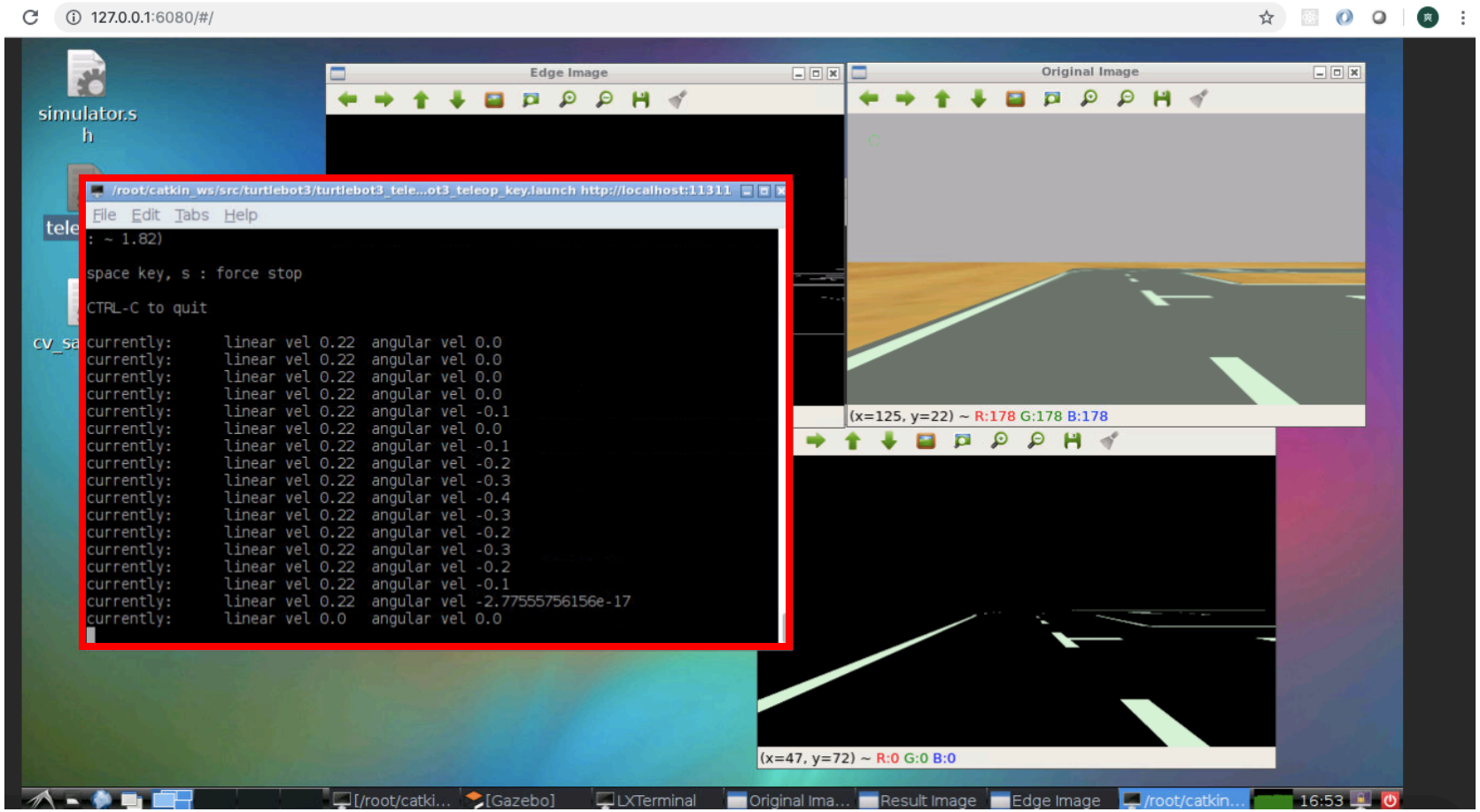
チュートリアル4



⑥teleop.sh
をダブルクリック

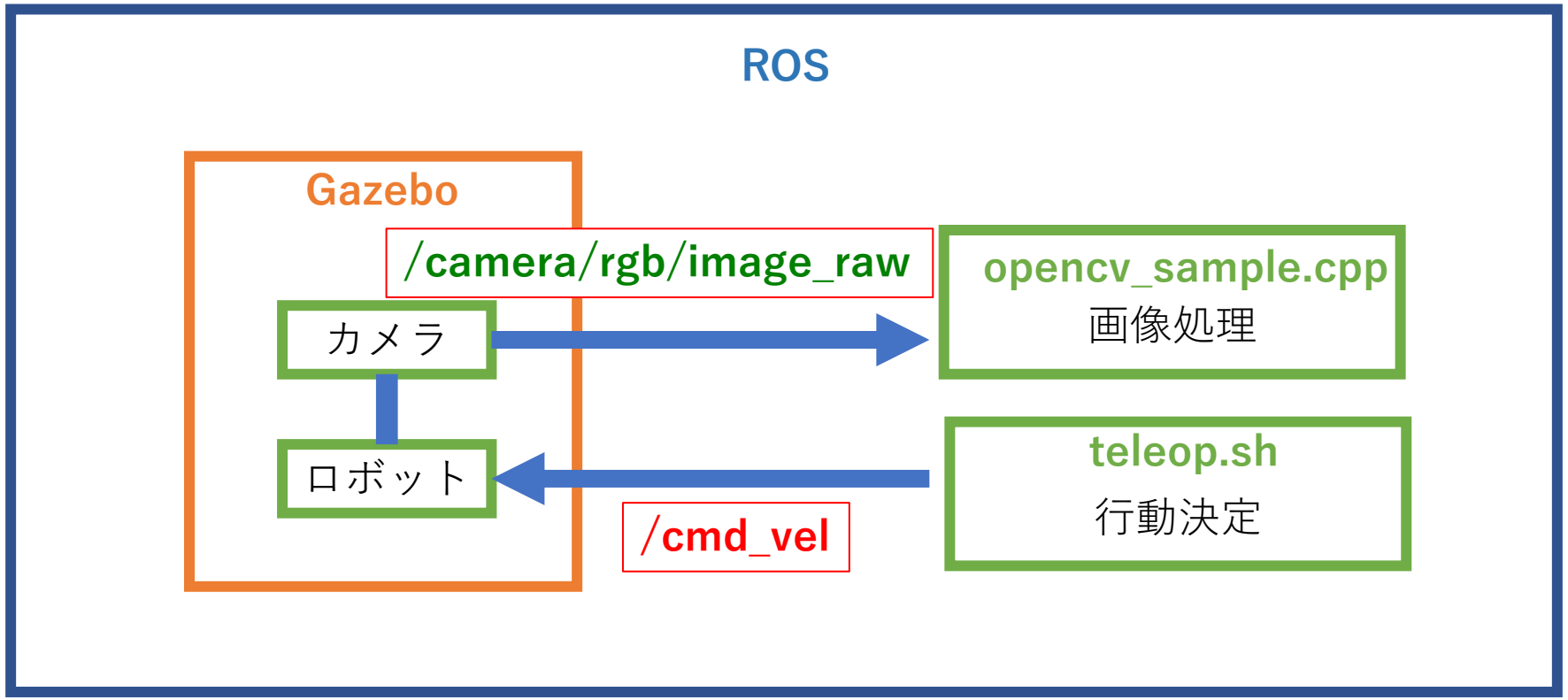
⑦Execute in terminal
をクリック

チュートリアル5



⑥
W
A S D
space
で運転！

仕組み


ROS
Gazebo
/camera/rgb/image_raw
カメラ
ロボット
/cmd_vel
opencv_sample.cpp
画像処理
teleop.sh
行動決定

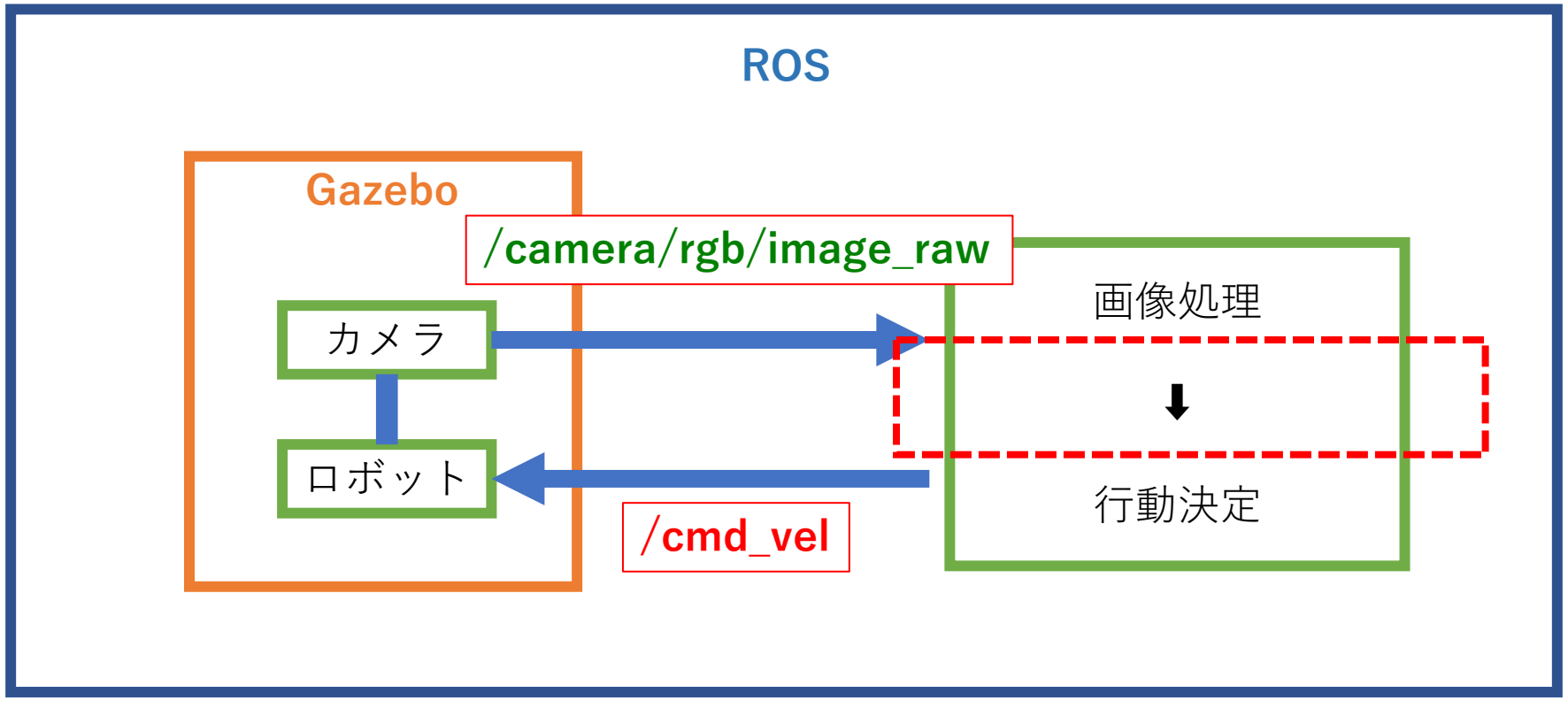
サンプルコード(opencv_sample.cpp)解説

(前略)

```
void imageCb(const sensor_msgs::ImageConstPtr& msg)
{
    (中略)
    cv_ptr    = cv_bridge::toCvCopy(msg,
                 sensor_msgs::image_encodings::BGR8)
    cv::imshow("Original Image", cv_ptr->image);
}
```

画像が送られてくるたび呼び出される

自動運転のために



これだけは必要なROS API

- twist_pub.publish(twist);
  - 速度：twist.linear.x = 0.1;（0.01 ~ 0.22）
  - 角度：twist.angular.z = 0.1; (-2.84 ~ 2.84 右回り)

  のように値をセットして、このAPIを呼び出すことで、ロボットに指令が飛ぶ

autonomous.cppの作り方

```
(前略)
void imageCb(const sensor_msgs::ImageConstPtr& msg)
{
    (中略)
    cv_ptr    = cv_bridge::toCvCopy(msg,
                 sensor_msgs::image_encodings::BGR8)

    twist_pub.publish(twist);
}
```

画像が送られてくるたび呼び出される

cv_ptrを画像処理して、twist. linear.x やtwist.angular.z を決定する処理

ここに任意のアルゴリズムを入れることで、ロボットを自動運転できる！

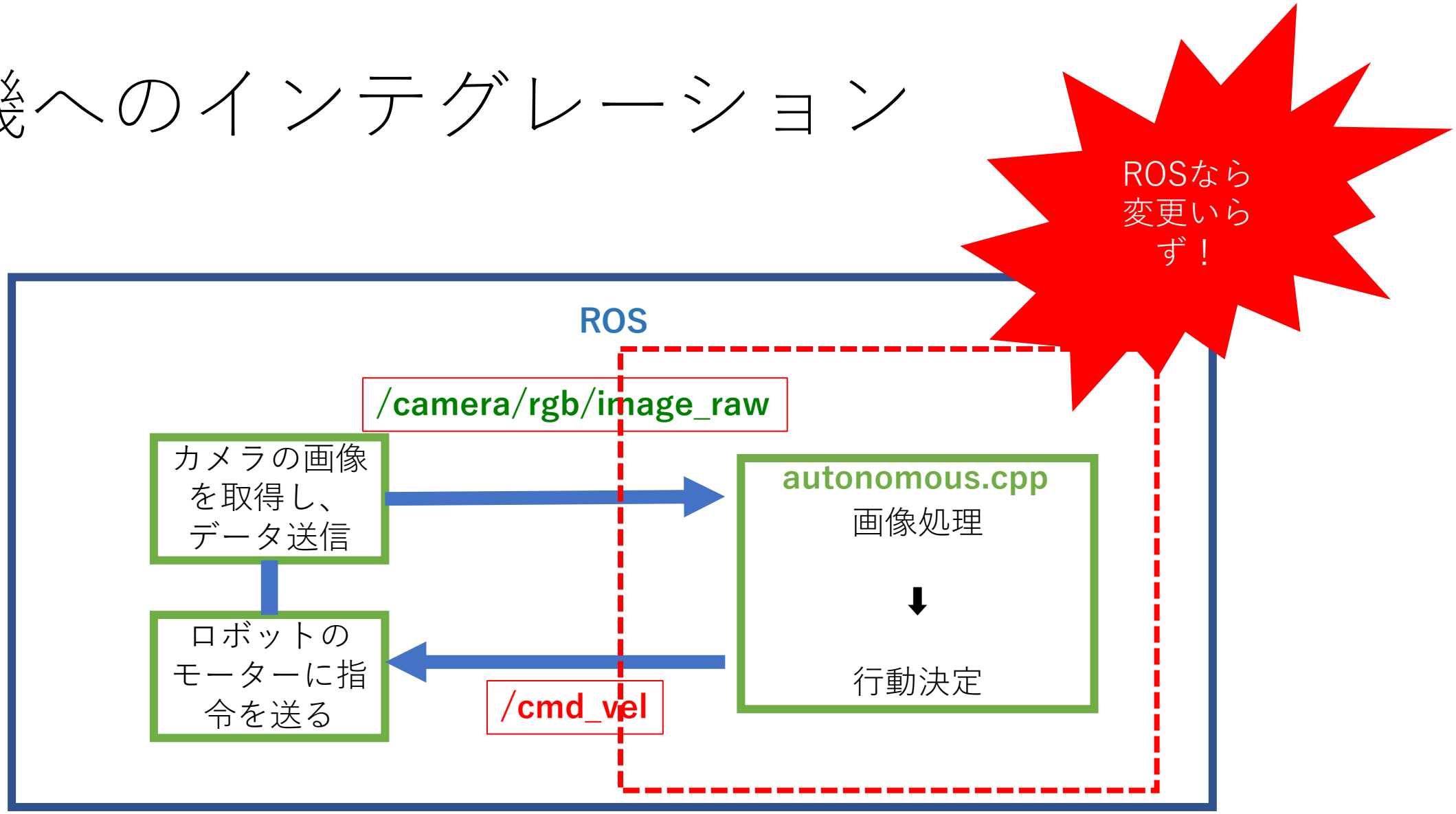
編集するファイル&ビルド

- /root/catkin_ws/src/opencv_sample/src/autonomous.cpp

- ビルド
  $ cd catkin_ws
  $ catkin_make

- 実行
  rosrun opencv_sample autonomous
  もしくはデスクップの autonomous.sh

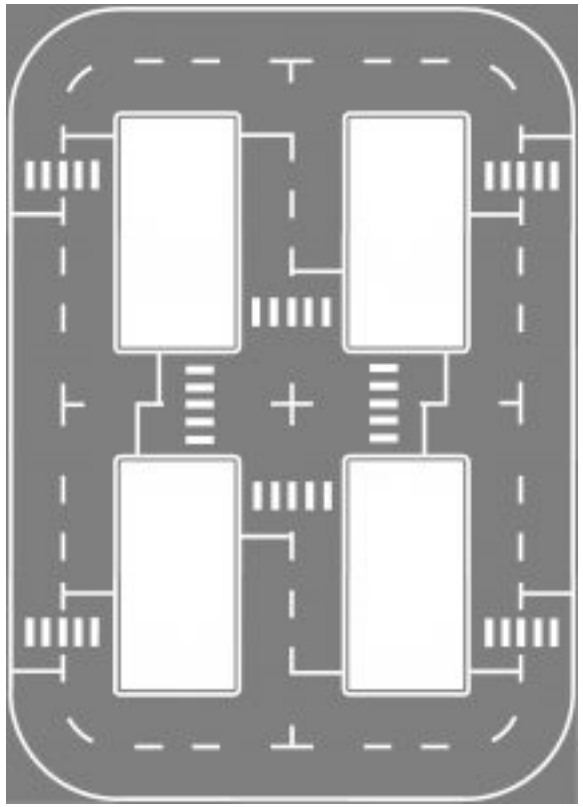
実機へのインテグレーション



ZytleBotで動かしているアルゴリズム

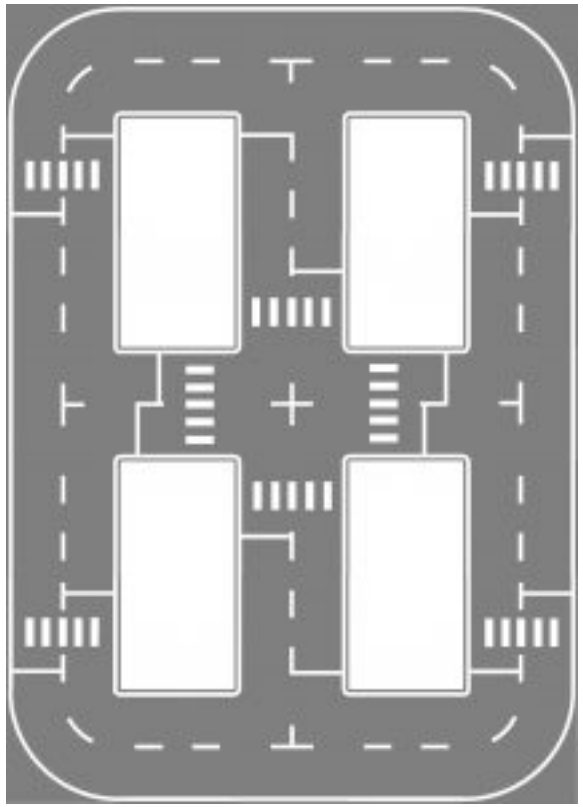
- どこをサボらせるか、こだわるか
  - 地図情報
  - 自己位置推定
  - 物体検出

地図情報



- コースをどう表現する？

地図情報



・・・1

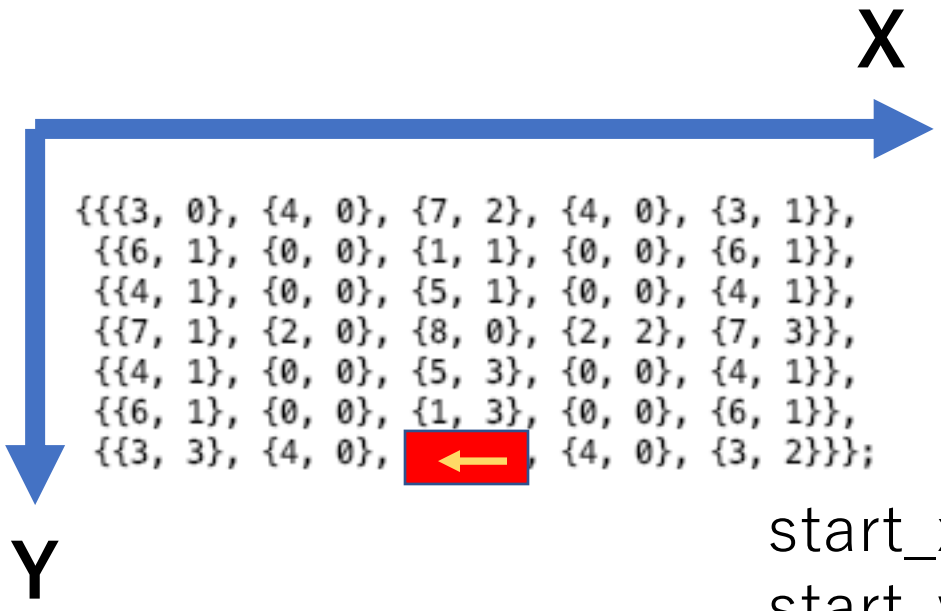
・・・2

・・・3

```
{{{3, 0}, {4, 0}, {7, 2}, {4, 0}, {3, 1}},
 {{6, 1}, {0, 0}, {1, 1}, {0, 0}, {6, 1}},
 {{4, 1}, {0, 0}, {5, 1}, {0, 0}, {4, 1}},
 {{7, 1}, {2, 0}, {8, 0}, {2, 2}, {7, 3}},
 {{4, 1}, {0, 0}, {5, 3}, {0, 0}, {4, 1}},
 {{6, 1}, {0, 0}, {1, 3}, {0, 0}, {6, 1}},
 {{3, 3}, {4, 0}, {7, 0}, {4, 0}, {3, 2}}};
```

- それぞれのタイルに番号を割り振る
- 向きを0~3で表現

地図情報



start_x : 2
start_y : 6
direction: 1

2
1 3
0

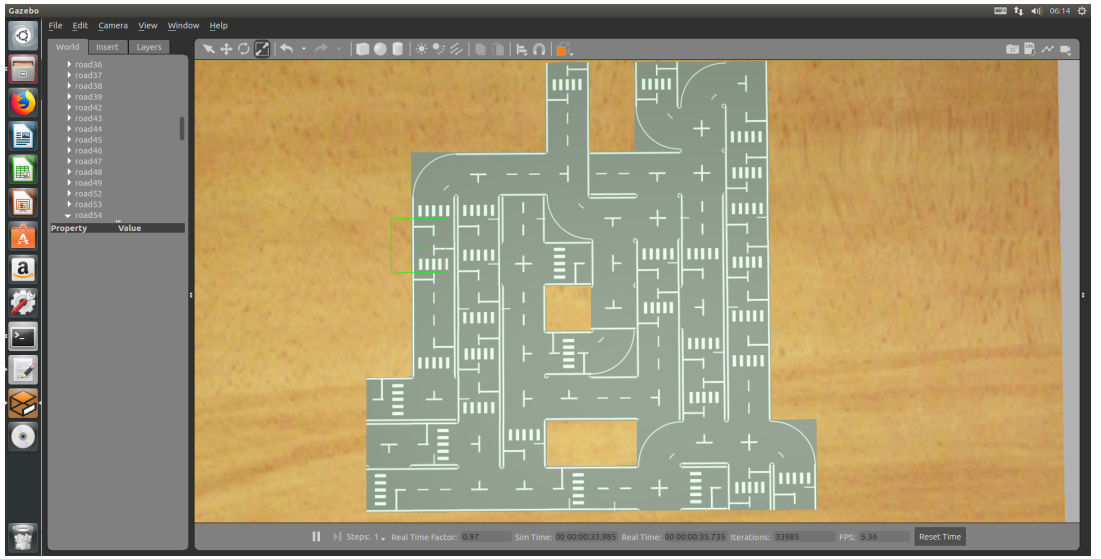
final_dir.txt

2010212301

- 後はスタート位置と向き、交差点ごとの移動方向を教えることで自在に走行可能

ちなみに

```
{{{3, 0}, {4, 0}, {7, 2}, {4, 0}, {3, 1}},
 {{6, 1}, {0, 0}, {1, 1}, {0, 0}, {6, 1}},
 {{4, 1}, {0, 0}, {5, 1}, {0, 0}, {4, 1}},
 {{7, 1}, {2, 0}, {8, 0}, {2, 2}, {7, 3}},
 {{4, 1}, {0, 0}, {5, 3}, {0, 0}, {4, 1}},
 {{6, 1}, {0, 0}, {1, 3}, {0, 0}, {6, 1}},
 {{3, 3}, {4, 0}, {7, 0}, {4, 0}, {3, 2}}};
```



- 配列からシミュレータ用のコースを吐くプログラムを作成
- ランダム経路を作成することも可能
- 急なコース複雑化にも**対応可能！**

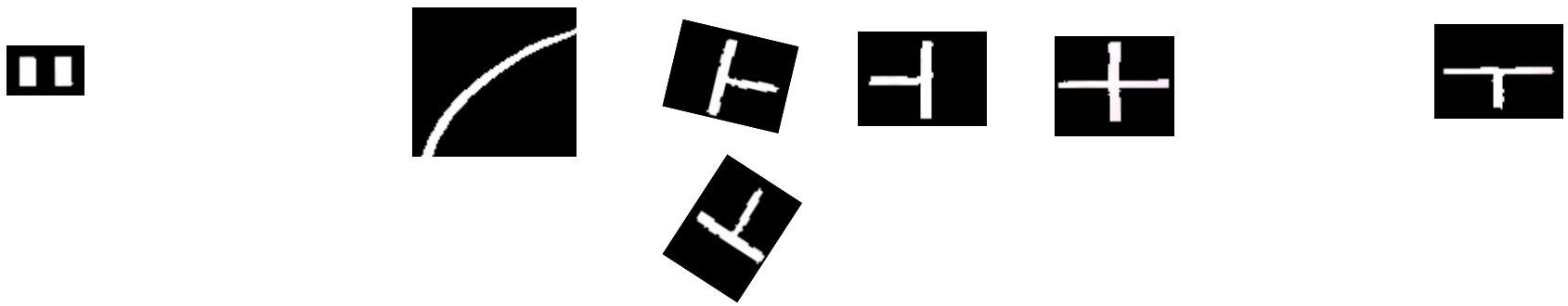
sousou1/gazebo_map_generator

自己位置推定？

- 自分がどのタイルにいるかを把握し続けるためには？

- タイル移動のフラグを監視
  - 例：横断歩道、T字、十字
  - 路面認識

➡テンプレートマッチング



テンプレートマッチングの弱点

- 比較元画像と、路面画像の大きさが一致する必要がある
- **回転に弱い**

- **取得した路面画像を元に、自分の車体と道路の角度のズレを求め、それを元に比較元画像を回転させる**