Project Hecate

Generated by Doxygen 1.8.15

Chapter 1

README

project_hecate

Overview

We propose a self-navigating package delivery robot, capable of finding route between logistic stations and deliver mobile parts like electric circuits, motherboards, screens and similar embedded parts from the manufacturing unit to the assembly line, in large factory units, like the Apple's factory in China. Such autonomous robotic system with inherent artificial intelligence to find it's way in factories and avoid collisions while traversing, has been developed to yield big returns to Acme robotics.

Main features of the product

- Capable of 'learning to find it's way' in a factory/random environment
- Obstacle avoidance
- Spawns at its default location (origin in the gazebo world) and when user commands to deliver a package, it moves to Point A to collect the package. It waits for the factory worker to put the package on it for 5 seconds and then moves towards the Point B, to deliver the package.
- Autonomous navigation

System Design and Algorithm

The architecture involves a turtlebot which has laserscan and odometry to receive the "state" of its environment and actuator control system which allows the turtlebot to take three actions, including move straight , turn right and turn left.

The algorithm implemented is called reinforcement learning (RL). RL is an aspect of Machine learning where an agent learns to behave in an environment, by performing certain actions and observing the rewards/results which it gets from those actions. It's important to note that while RL at its core aims to maximize rewards/gains, implementing a greedy approach, doesn't always lead to successful learning.

Demo Steps

Build Steps
```
cd mkdir -p  /catkin_ws/src
cd  /catkin_ws/
catkin_make
cd src
git clone https://github.com/ToyasDhake/project_hecate.git
cd ..
catkin_make
```

Demo Steps

The user has to specify two points in the gazebo world-

1. Point A- This is the point the turtleboit navigates to, from the origin resting place,in order to receive the load package from the factory worker. The turtlebot waits for the factory worker for about 5 seconds to put on the load. (syntax: xInitial:= X Coordinate of Point A yInitial:= Y Coordinate of Point A)
2. Point B- This is the point the turtlebot navigates to, after picking up the load from Point A, to drop the load at Point B. (syntax: xFinal:= X Coordinate of Point B yFinal:= Y Coordinate of Point B For example, in the commands below, Point A coordinates is (2,2) and Point B coordinates is (0,7). With our experiments we found that this is one of the tough combinations for the RL to predict trajectory of the turtlebot, but our results are pretty good even on these points.

```
source devel/setup.bash
#To load Default RL trained model
roslaunch project_hecate testHecate.launch xInitial:=2 yInitial:=2 xFinal:=0 yFinal:=7
# To train a custom model
roslaunch project_hecate trainHecate.launch path:=<path_to_save>
Note : <path_to_save> should be an absolute path. For example:
"/home/shivam/catkin_ws/src/project_hecate/model.csv"
#To load custom RL model trained by the user
roslaunch project_hecate testHecate.launch xInitial:=2 yInitial:=2 xFinal:=0 yFinal:=7 path:=<path_to_table>
```

Test Steps
```
cd  /catkin_ws/
catkin_make run_tests
```

Doxygen Steps
```
sudo apt install doxygen
cd <project_hecate repo>
doxygen -g
doxygen
cd latex
make
## Creates a pdf containing doxygen documentation. The same can be found in the repository under the
       Documentation folder.
```

Rosbag record and Play

To record:
```
cd  /catkin_ws/
source devel/setup.bash
roslaunch project_hecate testHecate.launch xInitial:=2 yInitial:=2 xFinal:=0 yFinal:=7 rosbagEnable:=true
```

To play: Download the rosbag from the following link: https://drive.google.com/open?id=1mX1fIqaXv8HXWAwkDrOgJz17dvpblmXe and place it in the results folder of this repository
```
Terminal 1:
roscore
Terminal2:
cd  /catkin_ws/src/project_hecate/results
rosbag play hecate.bag
```

Dependencies

ROS Kinetic

TurtleBot v2

ROS Kinetic

Gazebo 7.4 and above

Catkin

Results

The following gif shows the training of the turtlebot to "learn its way" through a floor map. During training as shown below, the turtlebot starts from the origin and then tries to navigate by taking actions of - going straight, take a left or take a right, in each episode. For each of these three actions, the turtlebot receives a reward. An episode involves a set of actions till the turtlebot collides. The episode ends after collision. The priciple during training is to achieve maximum sum of rewards in an episode. With more epochs of training, the turtlebot tries to maximize its rewards in the episode and stores the actions it took for the given states, which led to it earning maximum episode rewards.

During Inference, the turtlebot uses its learnt knowledge during training to decide on what actions to take, given a state, which had earlier fetched it maximum rewards during training. This is illustrated in the gif below, where the turtlebot is given Point A= (2,2) and Point B= (0,7). When the turtlebot starts from the origin to move towards (2,2), it attains maximum rewards for taking the action of turning slightly right towards (2,2) and then move straight towards it. Upon reaching point B, the turtlebot waits for the factory worker to place the weigth on the turtlebot. Again, when the turtlebot starts moving towards (0,7), for each small movement, the turtlebot looks inside the trained model to understand what action it should take based on what action earned it maximum reward when it was in similar state during training. Accordingly, the turtlebot finds it's way to the desired point B.

Assumptions:

1. We assume that the gazebo world is not changed drastically. Although the RL algorithm is capable of performing well in a dynamic world it was not trained on, drastic changes may require hyperparameter tuning of the algorithm.
2. We train the model on the gazebo simulator and assume that it performs well on real world too.
3. Acme Robotics has powerful systems with Ubuntu 16 and Ros kinetics with Gazebo (I7 processor, 16 GB RAM).
4. We assume that the obstacles are stationary.

Known Issues and Limitations

1. The RL algorithm is under active research. The algorithm implemented navigates the robot autonomously and collision free from point A to Point B, but ocassionally the path taken is not highly optimized.
2. The training of the turtlebot is highly compute intensive.
3. The Reinforcement learning algorithm was developed with hyperparametrs optimized for the gazebo world used in the simulation. New worlds may requires training the RL world with hyperparameter tuning and modifications.
4. The user has to define the Point A and Point B within the rectangular walls of the gazebo world. If not done so, the turtlebot would go towards the wall to reach the point, then avoid it and go back again and repeat in a loop. 5 The Cpplint forbids the use of "non-const reference". But passing "const" to ROS function callbacks is not allowed.

Developer Documentation

1. To train the model on a new gazebo world, tune the hyperparamers like the epsilon value, rewards. The developer might have to experiment with the linear and angular velocities for the robot to move take actions slower for the Rl states for better training.

2. Train the model with good number of epochs.

3. Create the gazebo environment as a single model instead of agreegating seprate models and building the environment. This is because when we call reset Environment in the training pipeline, the orientation of objects resets to (0,0,0).

Agile Development Process details

We followed the Pair Programming development procedure which started with both the contributors doing an extensive literature survey. After agreement on the algorithm and the approach, we moved ahead with the first level iteration planning. The week one of the project involved development of stub functions, which was Toyas (driver) while Shivam (navigator) was involved in planning and sanity check as well as a prototype development for the complete product as a proof of concept. The week 2 was started with the review of the potential risks and the remaining backlogs. We switched the roles, and the current driver implemented the Reinforcement learning algorithm, while the navigator did the review and to ensure quality of the product delivery. This development, implementation and testing was continued in the Iteration 3 of the project.

Documentation

Product Backlog and Sprint Schedule

The product backlog file can be accessed `here`

The Sprint planning and review document can be accessed `here`

The presentation is available `here`

License

Contributors

Shivam Akhauri

Former Artificial Intelligence Engineer at Ether Labs. -Former Machine learning Engineer and Project Lead at Tata Elxsi. -Skilled in AI/ML with applications in Computer vision, NLP and Robotics.

Toyas Dhake

Robotics engineer, University of Maryland College Park. -Skilled in embedded system with applications involving Arduino, Raspberry Pi and Jetson Boards.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

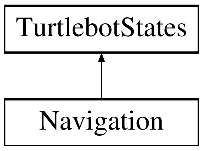
Class Documentation

5.1 Navigation Class Reference

Class Navigation This class contains members to generate linear and angular velocities to the turtulebot based on the depth from the obstacle information received from the depthCalculator.

```
#include <Navigation.hpp>
```

Inheritance diagram for Navigation:



Public Member Functions

- Navigation ()

  constructor Navigation class
- ∼Navigation ()

  destructor Navigation class
- void runRobot (double ix, double fx, double fy, QLearning &qLearning, std::vector< int > state, ros::Rate loop_rate)

  function runRobot
- void runRobot (std::string path, int &highestReward, int &episodeCount, int totalEpisode, int &nextStateIndex, ros::Rate loop_rate, int innerLoopLimit)

  function runRobot
- int getStateIndex (std::vector< int > state)

  function getStateIndex
- void action (int action, bool &colStatus, int &reward, int &nextState)

  function action
- void environmentReset ()

  function environmentReset
- void demoAction (int action)

  function demoAction
- void dom (const nav_msgs::Odometry::ConstPtr &msg)

  function dom

Public Attributes

- double **x**
- double **y**
- double **z**
- double **roll**
- double **pitch** = 0
- double **yaw**
- double **x_goal**
- double **y_goal**

5.1.1 Detailed Description

Class Navigation This class contains members to generate linear and angular velocities to the turtulebot based on the depth from the obstacle information received from the depthCalculator.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Navigation()

```
Navigation::Navigation ( )
```

constructor Navigation class

Parameters

none	

Returns

none initializes the publisher and subsciber initialize the value of odometry initialize the liner and angular speed

5.1.2.2 ∼Navigation()

```
Navigation::∼Navigation ( )
```

destructor Navigation class

Parameters

none	

Returns

none Destructor for the navigation clas

5.1.3 Member Function Documentation

5.1.3.1 action()

```
void Navigation::action (
            int action,
            bool & colStatus,
            int & reward,
            int & nextState )
```

function action

Parameters

int	action
bool	&colStatus
int	&reward
int	&nextState

Returns

none publishes linear and angular velocities to the turtlebot

5.1.3.2 demoAction()

```
void Navigation::demoAction (
            int action )
```

function demoAction

Parameters

int	action

Returns

none publishes linear and angular velocities to the turtlebot

5.1.3.3 dom()

```
void Navigation::dom (
            const nav_msgs::Odometry::ConstPtr & msg )
```

function dom

Parameters

const	nav_msgs::Odometry::ConstPtr

Returns

none callback to read odometry

5.1.3.4 environmentReset()

```
void Navigation::environmentReset ( )
```

function environmentReset

Parameters

none	

Returns

none resets the gazebo environment

5.1.3.5 getStateIndex()

```
int Navigation::getStateIndex (
            std::vector< int > state )
```

function getStateIndex

Parameters

std::vector<int>	state

Returns

int stateIndex mapping the vector to the state in rl table

5.1.3.6 runRobot() [1/2]

```
void Navigation::runRobot (
            double ix,
            double fx,
            double fy,
            QLearning & qLearning,
            std::vector< int > state,
            ros::Rate loop_rate )
```

function runRobot

Parameters

double	ix
double	fx
double	fy
QLearning	&qLearning
std::vector<int>	state
ros::Rate	loop_rate

Returns

none Runs the inferece code the bot uses the trained model to navigate

5.1.3.7 runRobot() [2/2]

```
void Navigation::runRobot (
            std::string path,
            int & highestReward,
            int & episodeCount,
            int totalEpisode,
            int & nextStateIndex,
            ros::Rate loop_rate,
            int innerLoopLimit )
```

function runRobot

Parameters

std::string	path
int	&highestReward
int	&episodeCount
int	totalEpisode, int &nextStateIndex
ros::Rate	loop_rate
int	innerLoopLimit

Returns

none training of the agent by receiving states perform actions in that states and receive rewards

The documentation for this class was generated from the following files:

- include/Navigation.hpp
- src/Navigation.cpp

5.2 QLearning Class Reference

Class Qlearning class to perform reinforcement learning algorithm.

`#include <QLearning.hpp>`

Public Member Functions

- QLearning ()
  constructor Qlearning class
- void setEpsilon (double e)
  function setEpsilon
- double getEpsilon ()
  function getEpsilon
- void setQtable (std::string path)
  function setQtable
- void getQtable (std::string path)
  function getQtable
- void qLearn (int state, int action, int reward, double val)
  function qlearn
- void robotLearn (int si, int act, int rew, int nsi)
  function robotLearn
- void testStoreQ ()
  function testStoreQ
- int demo (int index, bool collision, double angleToGoal)
  function demo
- int chooseAction (int index)
  function chooseAction

5.2.1 Detailed Description

Class Qlearning class to perform reinforcement learning algorithm.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 QLearning()

`QLearning::QLearning ( )`

constructor Qlearning class

Parameters

none	

Returns

none intililizes the reinforcement learning model

5.2.3 Member Function Documentation

5.2.3.1 chooseAction()

```
int QLearning::chooseAction (
            int index )
```

function chooseAction

Parameters

int	index

Returns

int action robots action selection for the state

5.2.3.2 demo()

```
int QLearning::demo (
            int index,
            bool collision,
            double angleToGoal )
```

function demo

Parameters

int	index
bool	collision
double	angleToGoal

Returns

int action use the rl model to decide the best action

5.2.3.3 getEpsilon()

```
double QLearning::getEpsilon ( )
```

function getEpsilon

Parameters

none	

Returns

double epsilon as getter for epsilon

5.2.3.4 getQtable()

```
void QLearning::getQtable (
            std::string path )
```

function getQtable

Parameters

std::string	path

Returns

none loads the pretrained RL model

5.2.3.5 qLearn()

```
void QLearning::qLearn (
            int state,
            int action,
            int reward,
            double val )
```

function qlearn

Parameters

int	state
int	action
int	reward
double	val

Returns

none updates reinforcement learning model

5.2.3.6 robotLearn()

```
void QLearning::robotLearn (
            int si,
            int act,
            int rew,
            int nsi )
```

function robotLearn

Parameters

int	si
int	act
int	rew
int	nsi

Returns

none applies the boltzmann equation to apply RL

5.2.3.7 setEpsilon()

```
void QLearning::setEpsilon (
            double e )
```

function setEpsilon

Parameters

double	e

Returns

none setter for epsilon

5.2.3.8 setQtable()

```
void QLearning::setQtable (
            std::string path )
```

function setQtable

Parameters

std::string	path

Returns

none stores the rl model

5.2.3.9 testStoreQ()

```
void QLearning::testStoreQ ( )
```

function testStoreQ

Parameters

none	

Returns

none function for inference quality test of the rl model

The documentation for this class was generated from the following files:

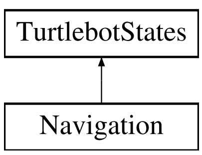
- include/QLearning.hpp
- src/QLearning.cpp

5.3 TurtlebotStates Class Reference

Class depthCalculatio This class contains members to calculate distance for the objects which is obtained from laserscan topic. It also contains members to raise a flag if about to collide.

```
#include <TurtlebotStates.hpp>
```

Inheritance diagram for TurtlebotStates:



Public Member Functions

- TurtlebotStates ()
  constructor TurtlebotStates
- ∼TurtlebotStates ()
  destructor TurtlebotStates
- void findLaserDepth (const sensor_msgs::LaserScan::ConstPtr &msg)
  function findLaserDepth
- bool flagCollision ()
  function flagCollision
- std::vector< int > returnLaserState ()
  function returnLaserState()

5.3.1 Detailed Description

Class depthCalculatio This class contains members to calculate distance for the objects which is obtained from laserscan topic. It also contains members to raise a flag if about to collide.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 TurtlebotStates()

```
TurtlebotStates::TurtlebotStates ( )
```

constructor TurtlebotStates

Parameters

none	

Returns

none initializes the collisionStatus flag

5.3.2.2 ∼TurtlebotStates()

```
TurtlebotStates::∼TurtlebotStates ( )
```

destructor TurtlebotStates

Parameters

none	

Returns

none destroy the TurtlebotStates

5.3.3 Member Function Documentation

5.3.3.1 findLaserDepth()

```
void TurtlebotStates::findLaserDepth (
            const sensor_msgs::LaserScan::ConstPtr & msg )
```

function findLaserDepth

Parameters

msg	type sensor_msgs::LaserScan

Returns

none function to read LaserScan sensor messages and raise flag if distance of the obstacle is less than threshold

5.3.3.2 flagCollision()

```
bool TurtlebotStates::flagCollision ( )
```

function flagCollision

Parameters

none	

Returns

1 if very close to obstacle and 0 if not close Return the current value of collisionStatus

5.3.3.3 returnLaserState()

```
std::vector< int > TurtlebotStates::returnLaserState ( )
```

function returnLaserState()

Parameters

none	

Returns

std::vector<int> return the states for the rl algorithm using the laserscan

The documentation for this class was generated from the following files:

- include/TurtlebotStates.hpp
- src/TurtlebotStates.cpp

Chapter 6

File Documentation

6.1 include/Navigation.hpp File Reference

Header for the robot autonomous of the robot.

```
#include <ros/ros.h>
#include <vector>
#include <string>
#include <iostream>
#include "sensor_msgs/LaserScan.h"
#include "std_srvs/Empty.h"
#include "geometry_msgs/Twist.h"
#include "TurtlebotStates.hpp"
#include "QLearning.hpp"
#include "nav_msgs/Odometry.h"
```

Classes

- class Navigation

  Class Navigation This class contains members to generate linear and angular velocities to the turtulebot based on the depth from the obstacle information received from the depthCalculator.

6.1.1 Detailed Description

Header for the robot autonomous of the robot.

Author

Shivam Akhauri, Toyas Dhake

Date

27 November 2019

Copyright

6.2 include/QLearning.hpp File Reference

Header for the RL algorithm implementation.

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
#include <array>
```

Classes

- class QLearning

  Class Qlearning class to perform reinforcement learning algorithm.

6.2.1 Detailed Description

Header for the RL algorithm implementation.

Author

Shivam Akhauri, Toyas Dhake

Date

27 November 2019

Copyright

BSD 3-clause, 2019 Shivam Akhauri,Toyas Dhake

6.3 include/TurtlebotStates.hpp File Reference

Header for reading the robot current states.

```
#include <vector>
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
#include "sensor_msgs/LaserScan.h"
```

Classes

- class TurtlebotStates

  Class depthCalculatio This class contains members to calculate distance for the objects which is obtained from laserscan topic. It also contains members to raise a flag if about to collide.

6.3.1 Detailed Description

Header for reading the robot current states.

Author

Shivam Akhauri, Toyas Dhake

Date

27 November 2019

Copyright

BSD 3-clause, 2019 Shivam Akhauri,Toyas Dhake

6.4 src/Navigation.cpp File Reference

Code for autonoumous naigation of the robot from a user defined start point to a stop point.

```
#include <tf/transform_listener.h>
#include <cmath>
#include <boost/range/irange.hpp>
#include "Navigation.hpp"
```

6.4.1 Detailed Description

Code for autonoumous naigation of the robot from a user defined start point to a stop point.

Author

Shivam Akhauri, Toyas Dhake

Date

27 November 2019

Copyright

BSD 3-clause, 2019 Shivam Akhauri,Toyas Dhake

6.5 src/QLearning.cpp File Reference

Code to define the reinforcement learning pipeline.

```
#include <time.h>
#include <ros/ros.h>
#include <iostream>
#include <sstream>
#include <fstream>
#include <vector>
#include <string>
#include <cmath>
#include <cstdlib>
#include <random>
#include <boost/range/irange.hpp>
#include "QLearning.hpp"
```

6.5.1 Detailed Description

Code to define the reinforcement learning pipeline.

Author

Shivam Akhauri, Toyas Dhake

Date

27 November 2019

Copyright

BSD 3-clause, 2019 Shivam Akhauri,Toyas Dhake

6.6 src/TurtlebotStates.cpp File Reference

Code to read the current states of the turtlebot.

```
#include <iostream>
#include <cfloat>
#include <cmath>
#include "TurtlebotStates.hpp"
#include <boost/range/irange.hpp>
```

6.6.1 Detailed Description

Code to read the current states of the turtlebot.

Author

Shivam Akhauri, Toyas Dhake

Date

27 November 2019

Copyright

BSD 3-clause, 2019 Shivam Akhauri,Toyas Dhake

6.7 test/NavigationTest.cpp File Reference

Test cases for class Navigation.

```
#include <gtest/gtest.h>
#include <ros/ros.h>
#include <geometry_msgs/Pose.h>
#include <geometry_msgs/Twist.h>
#include <vector>
#include "Navigation.hpp"
#include "QLearning.hpp"
```

Functions

- TEST (TESTNavigation, checkForCorrectStateIndex)

  Tests to verify the correctness of the state indices.
- TEST (TESTNavigation, checkForTestRobot)

  Tests to verify if the turtlebot reaches the desired loaction.
- TEST (TESTNavigation, checkForTrainRobot)

  Tests to verify if the Rl algorithm is training as expected.

6.7.1 Detailed Description

Test cases for class Navigation.

Author

Shivam Akhauri, Toyas Dhake

Copyright

3-clause BSD

6.7.2 Function Documentation

6.7.2.1 TEST() [1/3]

```
TEST (
        TESTNavigation ,
        checkForCorrectStateIndex )
```

Tests to verify the correctness of the state indices.

Parameters

Navigation	gtest framework
checkForCorrectStateIndex	Name of the test

Returns

none

6.7.2.2 TEST() [2/3]

```
TEST (
        TESTNavigation ,
        checkForTestRobot )
```

Tests to verify if the turtlebot reaches the desired loaction.

Parameters

Navigation	gtest framework
checkForTestRobot	Name of the test

Returns

none

6.7.2.3 TEST() [3/3]

```
TEST (
        TESTNavigation ,
        checkForTrainRobot )
```

Tests to verify if the Rl algorithm is training as expected.

Parameters

Navigation	gtest framework
checkForTrainRobot	Name of the test

Returns

none

6.8 test/QlearningTest.cpp File Reference

Test cases for class Qlearning.

```
#include <gtest/gtest.h>
#include <ros/ros.h>
#include "QLearning.hpp"
```

Functions

- TEST (TESTQlearning, testChooseAction)

  Test to verify expected turtlebot action.
- TEST (TestQlearning, testActionfromTheDemoFunctions)

  Test to verify if demo is taking proper decision based on values in table.

6.8.1 Detailed Description

Test cases for class Qlearning.

Author

Shivam Akhauri, Toyas Dhake

Copyright

3-clause BSD

6.8.2 Function Documentation

6.8.2.1 TEST() [1/2]

```
TEST (
            TESTQlearning ,
            testChooseAction )
```

Test to verify expected turtlebot action.

Parameters

TESTQlearning	gtest framework
testChooseAction	Name of the test

Returns

none

6.8.2.2 TEST() [2/2]

```
TEST (
            TestQlearning ,
            testActionfromTheDemoFunctions )
```

Test to verify if demo is taking proper decision based on values in table.

Parameters

TESTQlearning	gtest framework
testActionfromTheDemoFunctions	Name of the test

Returns

none

6.9 test/TurtlebotstatesTest.cpp File Reference

Test cases for class Turtlebotstates.

```
#include <gtest/gtest.h>
#include <ros/ros.h>
#include <sensor_msgs/LaserScan.h>
#include "TurtlebotStates.hpp"
```

Functions

- TEST (TESTTurtlebotState, checkObstacleDetection)

  Test to verify if Obstacle detection is happening properly. Obtain laserscan sensor data and raise a flag if obstacle detected.
- TEST (TESTTurtlebotState, checkDefaultflagCollisionValue)

  check if flag is raised when obstacle distance is very less

6.9.1 Detailed Description

Test cases for class Turtlebotstates.

Author

Shivam Akhauri, Toyas Dhake

Copyright

3-clause BSD

6.9.2 Function Documentation

6.9.2.1 TEST() [1/2]

```
TEST (
        TESTTurtlebotState ,
        checkObstacleDetection )
```

Test to verify if Obstacle detection is happening properly. Obtain laserscan sensor data and raise a flag if obstacle detected.

Parameters

TESTTurtlebotState	gtest framework
checkObstacleDetection	Name of the test

Returns

none

6.9.2.2 TEST() [2/2]

```
TEST (
            TESTTurtlebotState ,
            checkDefaultflagCollisionValue  )
```

check if flag is raised when obstacle distance is very less

Parameters

TESTTurtlebotState	gtest framework
checkDefaultflagCollisionValue	Name of the test

Returns

none

Index