ROS-Dual-CAM

Generated by Doxygen 1.9.4

Chapter 1

ROS-Dual-CAM

Running stereo dual cam using OpenCV and ROS noetic.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 dual-cam Namespace Reference

Variables

- bridge = CvBridge()
- pub_image = rospy.Publisher('/usb_cam/compressed/image_left', CompressedImage, tcp_nodelay=True, queue_size=1)
- pub_image2 = rospy.Publisher('/usb_cam/compressed/image_right', CompressedImage, tcp_nodelay=True, queue_size=1)
- cam1 = cv2.VideoCapture(0, cv2.CAP_V4L)
- cam2 = cv2.VideoCapture(3, cv2.CAP_V4L)
- start = time.time()
- ret1
- frame1 = bridge.cv2_to_compressed_imgmsg(frame1)
- ret2
- frame2 = bridge.cv2_to_compressed_imgmsg(frame2)
- fps = round(1 / (time.time() - start), 1)

6.1.1 Variable Documentation

6.1.1.1 bridge

```
dual-cam.bridge = CvBridge()
```

6.1.1.2 cam1

```
dual-cam.cam1 = cv2.VideoCapture(0, cv2.CAP_V4L)
```

6.1.1.3 cam2

```
dual-cam.cam2 = cv2.VideoCapture(3, cv2.CAP_V4L)
```

6.1.1.4 fps

```
dual-cam.fps = round(1 / (time.time() - start), 1)
```

6.1.1.5 frame1

```
dual-cam.frame1 = bridge.cv2_to_compressed_imgmsg(frame1)
```

6.1.1.6 frame2

```
dual-cam.frame2 = bridge.cv2_to_compressed_imgmsg(frame2)
```

6.1.1.7 pub_image

```
dual-cam.pub_image = rospy.Publisher('/usb_cam/compressed/image_left', CompressedImage, tcp_↩
nodelay=True, queue_size=1)
```

6.1.1.8 pub_image2

```
dual-cam.pub_image2 = rospy.Publisher('/usb_cam/compressed/image_right', CompressedImage,
tcp_nodelay=True, queue_size=1)
```

6.1.1.9 ret1

```
dual-cam.ret1
```

6.1.1.10 ret2

```
dual-cam.ret2
```

6.1.1.11 start

```
dual-cam.start = time.time()
```

6.2 test_local Namespace Reference

Classes

- class TestLocal

Variables

- path = os.path.dirname(os.path.abspath(__file__))
- config = yaml.load(ymlfile, Loader=yaml.FullLoader)
- real_ttb = rf.RealTtb(config, path, output=(640, 640))
- test_local = TestLocal()
- key = cv2.waitKey(1)
- start = time.time()
- val = test_local.step()
- fps = round(1 / (time.time() - start), 1)

6.2.1 Variable Documentation

6.2.1.1 config

```
test_local.config = yaml.load(ymlfile, Loader=yaml.FullLoader)
```

6.2.1.2 fps

```
test_local.fps = round(1 / (time.time() - start), 1)
```

6.2.1.3 key

```
test_local.key = cv2.waitKey(1)
```

6.2.1.4 path

```
test_local.path = os.path.dirname(os.path.abspath(__file__))
```

6.2.1.5 real_ttb

```
test_local.real_ttb = rf.RealTtb(config, path, output=(640, 640))
```

6.2.1.6 start

```
test_local.start = time.time()
```

6.2.1.7 test_local

```
test_local.test_local = TestLocal()
```

6.2.1.8 val

```
test_local.val = test_local.step()
```

6.3 test_real Namespace Reference

Classes

- class LinearDeepQNetwork

Functions

- def updateLidar (lidar)
- def getImage (image)

Variables

- pub_cmd_vel = rospy.Publisher('cmd_vel', Twist, queue_size=1)
- lidar_g = None
- sub_scan = rospy.Subscriber('scan', LaserScan, updateLidar)
- string TURTLE = '003'
- bridge = CvBridge()
- state = None
- font = cv2.FONT_HERSHEY_SIMPLEX
- outfile = TemporaryFile()
- int episodes = 50
- int max_steps = 50000
- list action_low = [-1.5, -0.1]
- list action_high = [1.5, 0.12]
- path = os.path.dirname(os.path.abspath(__file__))
- config = yaml.load(ymlfile, Loader=yaml.FullLoader)
- env = input('Which environment are you running? [1 | 2 | l]:\n')
- real_ttb = rf.RealTtb(config, path, output=(640, 640))
- sub_image = rospy.Subscriber('/usb_cam/compressed/image_right', CompressedImage, getImage, queue↩ _size=1)
- string path_results = path + '/real_results'
- algorithms_sel = np.array(['1', '2', '3', 'e', 'r'])
- string algorithm = ""
- string model = '2' else f"dqn_st{env.upper()}_model_5k.pth"
- string model_fn = f"{path}/{model}"
- int action_size = 5
- int observation_space = 26
- actor = LinearDeepQNetwork(0, action_size, observation_space)
- dictionary det = {0: -1.5, 1: -0.75, 2: 0, 3: 0.75, 4: 1.5}
- int local_episode = 0
- quit = input("Press [Enter] to start the test or press [q] to quit...")
- int episode_reward = 0
- lidar_list = list()
- int num_steps = 0
- ep_start_time = time.time()
- bool done = False
- start = time.time()
- val_state = state
- actions = actor(state)
- action = torch.argmax(actions).item()
- vel_cmd = Twist()
- x
- z
- int reward = 0
- scan = val_state[0:24]
- fps = round(1 / (time.time() - start))
- episode_timing = time.time() - ep_start_time
- list values = [episode_reward, episode_timing, local_episode, num_steps, real_ttb.pts, lidar_list]

6.3.1 Function Documentation

6.3.1.1 getImage()

```
def test_real.getImage (
              image )
```

6.3.1.2 updateLidar()

```
def test_real.updateLidar (
              lidar )
```

6.3.2 Variable Documentation

6.3.2.1 action

```
test_real.action = torch.argmax(actions).item()
```

6.3.2.2 action_high

```
list test_real.action_high = [1.5, 0.12]
```

6.3.2.3 action_low

```
list test_real.action_low = [-1.5, -0.1]
```

6.3.2.4 action_size

```
int test_real.action_size = 5
```

6.3.2.5 actions

```
test_real.actions = actor(state)
```

6.3.2.6 actor

```
test_real.actor = LinearDeepQNetwork(0, action_size, observation_space)
```

6.3.2.7 algorithm

```
test_real.algorithm = ""
```

6.3.2.8 algorithms_sel

```
test_real.algorithms_sel = np.array(['1', '2', '3', 'e', 'r'])
```

6.3.2.9 bridge

```
test_real.bridge = CvBridge()
```

6.3.2.10 config

```
test_real.config = yaml.load(ymlfile, Loader=yaml.FullLoader)
```

6.3.2.11 det

```
dictionary test_real.det = {0:  -1.5, 1:  -0.75, 2:  0, 3:  0.75, 4:  1.5}
```

6.3.2.12 done

```
bool test_real.done = False
```

6.3.2.13 env

```
test_real.env = input('Which environment are you running?  [1 | 2 | l]:\n')
```

6.3.2.14 ep_start_time

```
test_real.ep_start_time = time.time()
```

6.3.2.15 episode_reward

```
int test_real.episode_reward = 0
```

6.3.2.16 episode_timing

```
test_real.episode_timing = time.time() - ep_start_time
```

6.3.2.17 episodes

```
int test_real.episodes = 50
```

6.3.2.18 font

```
test_real.font = cv2.FONT_HERSHEY_SIMPLEX
```

6.3.2.19 fps

```
test_real.fps = round(1 / (time.time() - start))
```

6.3.2.20 lidar_g

```
test_real.lidar_g = None
```

6.3.2.21 lidar_list

```
test_real.lidar_list = list()
```

6.3.2.22 local_episode

```
int test_real.local_episode = 0
```

6.3.2.23 max_steps

```
int test_real.max_steps = 50000
```

6.3.2.24 model

```
string test_real.model = '2' else f"dqn_st{env.upper()}_model_5k.pth"
```

6.3.2.25 model_fn

```
string test_real.model_fn = f"{path}/{model}"
```

6.3.2.26 num_steps

```
int test_real.num_steps = 0
```

6.3.2.27 observation_space

```
int test_real.observation_space = 26
```

6.3.2.28 outfile

```
test_real.outfile = TemporaryFile()
```

6.3.2.29 path

```
test_real.path = os.path.dirname(os.path.abspath(__file__))
```

6.3.2.30 path_results

```
string test_real.path_results = path + '/real_results'
```

6.3.2.31 pub_cmd_vel

```
test_real.pub_cmd_vel = rospy.Publisher('cmd_vel', Twist, queue_size=1)
```

6.3.2.32 quit

```
test_real.quit = input("Press [Enter] to start the test or press [q] to quit...")
```

6.3.2.33 real_ttb

```
test_real.real_ttb = rf.RealTtb(config, path, output=(640, 640))
```

6.3.2.34 reward

```
int test_real.reward = 0
```

6.3.2.35 scan

```
test_real.scan = val_state[0:24]
```

6.3.2.36 start

```
test_real.start = time.time()
```

6.3.2.37 state

```
test_real.state = None
```

6.3.2.38 sub_image

```
test_real.sub_image = rospy.Subscriber('/usb_cam/compressed/image_right', CompressedImage,
getImage, queue_size=1)
```

6.3.2.39 sub_scan

```
test_real.sub_scan = rospy.Subscriber('scan', LaserScan, updateLidar)
```

6.3.2.40 TURTLE

```
string test_real.TURTLE = '003'
```

6.3.2.41 val_state

```
test_real.val_state = state
```

6.3.2.42 values

```
list test_real.values = [episode_reward, episode_timing, local_episode, num_steps, real_ttb.↩
pts, lidar_list]
```

6.3.2.43 vel_cmd

```
test_real.vel_cmd = Twist()
```

6.3.2.44 x

```
test_real.x
```

6.3.2.45 z

```
test_real.z
```

6.4 view_real_results Namespace Reference

Functions

- def antispike (old_list_x, old_list_y)
- def antispike2 (old_list_x, old_list_y)
- def open_test_data (i)

Variables

- path = os.path.dirname(os.path.abspath(__file__))
- list_dir = os.listdir(path + '/real_results/')
- int threshold_x = 10
- int threshold_y = 30
- int threshold = 10
- int STAGE = 1
- int c = 7
- stage = mpimg.imread(path+'/media/stage_{}_real.png'.format(STAGE))
- data = list()
- dictionary color = {0: 'firebrick', 1: 'tomato', 2: 'peru', 3: 'gold', 4: 'dodgerblue', 5: 'springgreen', 6: 'indigo', 7: 'deeppink'}
- size = len(data)
- rewards = list()
- times = list()
- list x = [ ]
- list y = [ ]
- linestyle
- linewidth

6.4.1 Function Documentation

6.4.1.1 antispike()

```
def view_real_results.antispike (
              old_list_x,
              old_list_y )
```

6.4.1.2 antispike2()

```
def view_real_results.antispike2 (
              old_list_x,
              old_list_y )
```

6.4.1.3 open_test_data()

```
def view_real_results.open_test_data (
              i )
```

6.4.2 Variable Documentation

6.4.2.1 c

```
int view_real_results.c = 7
```

6.4.2.2 color

```
view_real_results.color = {0:  'firebrick', 1:  'tomato', 2:  'peru', 3:  'gold', 4:  'dodgerblue',
5:  'springgreen', 6:  'indigo', 7:  'deeppink'}
```

6.4.2.3 data

```
view_real_results.data = list()
```

6.4.2.4 linestyle

```
view_real_results.linestyle
```

6.4.2.5 linewidth

```
view_real_results.linewidth
```

6.4.2.6 list_dir

```
view_real_results.list_dir = os.listdir(path + '/real_results/')
```

6.4.2.7 path

```
view_real_results.path = os.path.dirname(os.path.abspath(__file__))
```

6.4.2.8 rewards

```
view_real_results.rewards = list()
```

6.4.2.9 size

```
view_real_results.size = len(data)
```

6.4.2.10 STAGE

```
int view_real_results.STAGE = 1
```

6.4.2.11 stage

```
view_real_results.stage = mpimg.imread(path+'/media/stage_{}_real.png'.format(STAGE))
```

6.4.2.12 threshold

```
int view_real_results.threshold = 10
```

6.4.2.13 threshold_x

```
int view_real_results.threshold_x = 10
```

6.4.2.14 threshold_y

```
int view_real_results.threshold_y = 30
```

6.4.2.15 times

```
view_real_results.times = list()
```

6.4.2.16 x

```
list view_real_results.x = []
```

6.4.2.17 y

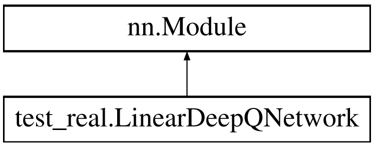
```
list view_real_results.y = []
```

Chapter 7

Class Documentation

7.1 test_real.LinearDeepQNetwork Class Reference

Inheritance diagram for test_real.LinearDeepQNetwork:



Public Member Functions

- def __init__ (self, lr, n_actions, input_dims)
- def forward (self, state)

Public Attributes

- fc1
- fc2
- dropout
- fc3
- optimizer
- loss
- device

7.1.1 Constructor & Destructor Documentation

7.1.1.1 __init__()

```
def test_real.LinearDeepQNetwork.__init__ (
              self,
              lr,
              n_actions,
              input_dims )
```

7.1.2 Member Function Documentation

7.1.2.1 forward()

```
def test_real.LinearDeepQNetwork.forward (
              self,
              state )
```

7.1.3 Member Data Documentation

7.1.3.1 device

```
test_real.LinearDeepQNetwork.device
```

7.1.3.2 dropout

```
test_real.LinearDeepQNetwork.dropout
```

7.1.3.3 fc1

```
test_real.LinearDeepQNetwork.fc1
```

7.1.3.4 fc2

```
test_real.LinearDeepQNetwork.fc2
```

7.1.3.5 fc3

```
test_real.LinearDeepQNetwork.fc3
```

7.1.3.6 loss

```
test_real.LinearDeepQNetwork.loss
```

7.1.3.7 optimizer

```
test_real.LinearDeepQNetwork.optimizer
```

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

- test_real.py

7.2 test_local.TestLocal Class Reference

Public Member Functions

- def __init__ (self)
- def image_right_callback (self, msg)
- def image_left_callback (self, msg)
- def step (self)

Public Attributes

- bridge
- stitcher
- image_right
- image_left
- defisheye1
- defisheye2

7.2.1 Constructor & Destructor Documentation

7.2.1.1 __init__()

```
def test_local.TestLocal.__init__ (
              self )
```

7.2.2 Member Function Documentation

7.2.2.1 image_left_callback()

```
def test_local.TestLocal.image_left_callback (
              self,
              msg )
```

7.2.2.2 image_right_callback()

```
def test_local.TestLocal.image_right_callback (
              self,
              msg )
```

7.2.2.3 step()

```
def test_local.TestLocal.step (
              self )
```

7.2.3 Member Data Documentation

7.2.3.1 bridge

```
test_local.TestLocal.bridge
```

7.2.3.2 defisheye1

```
test_local.TestLocal.defisheye1
```

7.2.3.3 defisheye2

```
test_local.TestLocal.defisheye2
```

7.2.3.4 image_left

```
test_local.TestLocal.image_left
```

7.2.3.5 image_right

```
test_local.TestLocal.image_right
```

7.2.3.6 stitcher

```
test_local.TestLocal.stitcher
```

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

- test_local.py

Chapter 8

File Documentation

8.1 dual-cam.py File Reference

Namespaces

- namespace dual-cam

Variables

- dual-cam.bridge = CvBridge()
- dual-cam.pub_image = rospy.Publisher('/usb_cam/compressed/image_left', CompressedImage, tcp_↩ nodelay=True, queue_size=1)
- dual-cam.pub_image2 = rospy.Publisher('/usb_cam/compressed/image_right', CompressedImage, tcp_↩ nodelay=True, queue_size=1)
- dual-cam.cam1 = cv2.VideoCapture(0, cv2.CAP_V4L)
- dual-cam.cam2 = cv2.VideoCapture(3, cv2.CAP_V4L)
- dual-cam.start = time.time()
- dual-cam.ret1
- dual-cam.frame1 = bridge.cv2_to_compressed_imgmsg(frame1)
- dual-cam.ret2
- dual-cam.frame2 = bridge.cv2_to_compressed_imgmsg(frame2)
- dual-cam.fps = round(1 / (time.time() - start), 1)

8.2 README.md File Reference

8.3 test_local.py File Reference

Classes

- class test_local.TestLocal

Namespaces

- namespace test_local

Variables

- test_local.path = os.path.dirname(os.path.abspath(__file__))
- test_local.config = yaml.load(ymlfile, Loader=yaml.FullLoader)
- test_local.real_ttb = rf.RealTtb(config, path, output=(640, 640))
- test_local.test_local = TestLocal()
- test_local.key = cv2.waitKey(1)
- test_local.start = time.time()
- test_local.val = test_local.step()
- test_local.fps = round(1 / (time.time() - start), 1)

8.4 test_real.py File Reference

Classes

- class test_real.LinearDeepQNetwork

Namespaces

- namespace test_real

Functions

- def test_real.updateLidar (lidar)
- def test_real.getImage (image)

Variables

- test_real.pub_cmd_vel = rospy.Publisher('cmd_vel', Twist, queue_size=1)
- test_real.lidar_g = None
- test_real.sub_scan = rospy.Subscriber('scan', LaserScan, updateLidar)
- string test_real.TURTLE = '003'
- test_real.bridge = CvBridge()
- test_real.state = None
- test_real.font = cv2.FONT_HERSHEY_SIMPLEX
- test_real.outfile = TemporaryFile()
- int test_real.episodes = 50
- int test_real.max_steps = 50000
- list test_real.action_low = [-1.5, -0.1]
- list test_real.action_high = [1.5, 0.12]
- test_real.path = os.path.dirname(os.path.abspath(__file__))
- test_real.config = yaml.load(ymlfile, Loader=yaml.FullLoader)
- test_real.env = input('Which environment are you running? [1 | 2 | l]:\n')
- test_real.real_ttb = rf.RealTtb(config, path, output=(640, 640))
- test_real.sub_image = rospy.Subscriber('/usb_cam/compressed/image_right', CompressedImage, getImage, queue_size=1)
- string test_real.path_results = path + '/real_results'
- test_real.algorithms_sel = np.array(['1', '2', '3', 'e', 'r'])
- string test_real.algorithm = ""
- string test_real.model = '2' else f"dqn_st{env.upper()}_model_5k.pth"

- string test_real.model_fn = f"{path}/{model}"
- int test_real.action_size = 5
- int test_real.observation_space = 26
- test_real.actor = LinearDeepQNetwork(0, action_size, observation_space)
- dictionary test_real.det = {0: -1.5, 1: -0.75, 2: 0, 3: 0.75, 4: 1.5}
- int test_real.local_episode = 0
- test_real.quit = input("Press [Enter] to start the test or press [q] to quit...")
- int test_real.episode_reward = 0
- test_real.lidar_list = list()
- int test_real.num_steps = 0
- test_real.ep_start_time = time.time()
- bool test_real.done = False
- test_real.start = time.time()
- test_real.val_state = state
- test_real.actions = actor(state)
- test_real.action = torch.argmax(actions).item()
- test_real.vel_cmd = Twist()
- test_real.x
- test_real.z
- int test_real.reward = 0
- test_real.scan = val_state[0:24]
- test_real.fps = round(1 / (time.time() - start))
- test_real.episode_timing = time.time() - ep_start_time
- list test_real.values = [episode_reward, episode_timing, local_episode, num_steps, real_ttb.pts, lidar_list]

8.5 view_real_results.py File Reference

Namespaces

- namespace view_real_results

Functions

- def view_real_results.antispike (old_list_x, old_list_y)
- def view_real_results.antispike2 (old_list_x, old_list_y)
- def view_real_results.open_test_data (i)

Variables

- view_real_results.path = os.path.dirname(os.path.abspath(__file__))
- view_real_results.list_dir = os.listdir(path + '/real_results/')
- int view_real_results.threshold_x = 10
- int view_real_results.threshold_y = 30
- int view_real_results.threshold = 10
- int view_real_results.STAGE = 1
- int view_real_results.c = 7
- view_real_results.stage = mpimg.imread(path+'/media/stage_{}_real.png'.format(STAGE))
- view_real_results.data = list()
- dictionary view_real_results.color = {0: 'firebrick', 1: 'tomato', 2: 'peru', 3: 'gold', 4: 'dodgerblue', 5: 'springgreen', 6: 'indigo', 7: 'deeppink'}
- view_real_results.size = len(data)
- view_real_results.rewards = list()
- view_real_results.times = list()
- list view_real_results.x = [ ]
- list view_real_results.y = [ ]
- view_real_results.linestyle
- view_real_results.linewidth

Index