Robotics Project 3
Automatic Sorting Maschine

Jeremy Zid
Sebastian Riemann
Joshua Junt
Fabrice Cheteu
Paul Bölscher

Generated by Doxygen 1.9.1

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

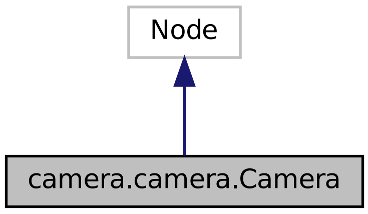
Chapter 3

Class Documentation

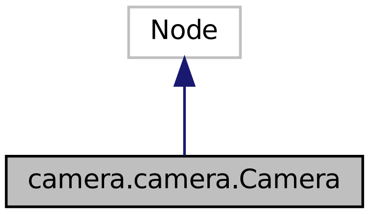
3.1 camera.camera.Camera Class Reference

A Camera instance is used to create a ROS2 node for image capturing.

Inheritance diagram for camera.camera.Camera:



Collaboration diagram for camera.camera.Camera:



Public Member Functions

- def __init__ (self, framerate, queue)

  Initializes a new Camera instance.
- def capture (self)

  Captures an image from a video device, sets its timestamp and publishes it.

3.1.1 Detailed Description

A Camera instance is used to create a ROS2 node for image capturing.

The node captures images from a video device and publishes them on the topic "camera_stream".

3.1.2 Constructor & Destructor Documentation

3.1.2.1 __init__()

```
def camera.camera.Camera.__init__ (
              self,
              framerate,
              queue )
```

Initializes a new Camera instance.

Parameters

framerate	The rate at which images are captured and published.
queue	The queue size of the image publishing.

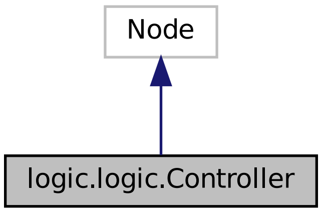
The documentation for this class was generated from the following file:

- /home/seb/ros2_ws/src/camera/camera/camera.py

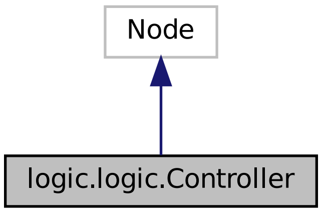
3.2 logic.logic.Controller Class Reference

A logic Controller to control the robot's movements based on received messages and data.

Inheritance diagram for logic.logic.Controller:



Collaboration diagram for logic.logic.Controller:



Public Member Functions

- def __init__ (self)

  Initializes a new Controller instance.

Static Public Member Functions

- def error_callback (self, msg)

  Error callback function to get the error message and set the error variable to true if the function gets called.
- def object_callback (self, msg)

  Object callback function to get the object message and set the velocity in x direction to the negative value of the object velocity.
- def robotPosition_callback (self, msg_in)

  Robot position callback function to get the robot position message and set the robot position in dependence of the current state.

3.2.1 Detailed Description

A logic Controller to control the robot's movements based on received messages and data.

The Logic Controller class represents the main logic for controlling the robot's movements. The node is responsible for controling the robot's behavior, which includes fetching and picking up objects from a conveyor belt, moving to specific positions, and handling error states. It subscribes to several ROS2 messages to receive information about the robot's state, the velocity of the conveyor belt, and the current position of the robot. It also publishes ROS2 messages to command the robot's movements. The logic includes:

- Moving the robot to the initialize position after start.
- Moving the robot to the desired positions.
- Sorting the objects in the boxes.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 __init__()

```
def logic.logic.Controller.__init__ (
              self )
```

Initializes a new Controller instance.

3.2.3 Member Function Documentation

3.2.3.1 error_callback()

```
def logic.logic.Controller.error_callback (
              self,
              msg )  [static]
```

Error callback function to get the error message and set the error variable to true if the function gets called.

This function gets called when an error message gets called and sets the error value to True. With the ROS2 logger the error message gets displayed on the terminal.

Parameters

msg	The error message.

3.2.3.2 object_callback()

```
def logic.logic.Controller.object_callback (
              self,
              msg )  [static]
```

Object callback function to get the object message and set the velocity in x direction to the negative value of the object velocity.

Parameters

msg	The object message.

3.2.3.3 robotPosition_callback()

```
def logic.logic.Controller.robotPosition_callback (
              self,
              msg_in )  [static]
```

Robot position callback function to get the robot position message and set the robot position in dependence of the current state.

Parameters

msg↩ _in	The robot position message.

Cases: The robot moves to different positions in dependence of the current state.

- Case 'init' to move the step motor from the z-axis to the initialize position. The case publishes an updated new robot position that moves the robot for 10 seconds straight to the minimum border of the z-axis. After the 10 seconds, the z position is 0.0, and the next case 'reference_z' is called.
- Case 'reference_z' to move the step motors from the x and y-axis to the initialize position. The case publishes an updated new robot position that moves the robot for 10 seconds straight to the minimum borders of the x and y-axis. After the 10 seconds, the x and y positions are set to 0.0, and a new robot position is published with the defined __wait position, which gets called after that.
- Case 'wait' to move the robot to the defined robot waiting position. Since the wait position is already being published in the case 'reference_z', the robot moves without a new publish to the wait position. The robot waits until a new object is detected, and the vel_x is unequal to zero. After a new object is detected, the y and z values of the robot get updated, and the next case 'fetch_x' gets called.
- Case 'fetch_x' to move the robot along the x-axis to a target position. In the 'fetch_x' case, the robot is controlled to move along the x-axis to reach the object position. The robot trys to chatch the object on the conveyor belt by moving along the x-axis with the object velocity. If the robot can't catch the object, the robot moves to the wait position. If the robot catches the object, the robot moves to the Case 'lifting'.

- Case 'lifting' to pick up the reached object and move the z-axis up. In the 'lifting' case, the robot picks up the reached object by moving the z-axis upwards until the value is greater than or equal to the wait position. After reaching the desired z position, the x position gets updated to the x position of the unicorn or cat box, depending on the lifted object type. The z position gets updated to the value where the y-axis is free to move, and the updated x and z positions get published. After publishing the new positions, the case 'y_enabling' is called.
- Case 'y_enabling' moves the y position of the robot to the y position of the boxes. The y position gets updated after the z position reaches the position where y can move without any danger of crashing or losing the object. After that, the y and z positions are updated and published to the y and z positions of the object box. After the publish, the next case 'xy_moving' is called.
- Case 'xy_moving' moves the robot over the object box and releases the vacuum to sort the object in the box. After the x and y positions are close enough to the desired position, the gripper releases the vacuum, and the current position is published with the updated gripper value. The x and y positions are updated and published to the wait position. The next case 'y_disabling' is called after the position is published.
- Case y_disabling moves the z position to the wait position. After the robot is over the desired x and y position the z position is updated and published. The next case 'wait' gets called after publishing the new z positon.
- Case 'error_z' If there is an error, the gripper set to False and the z axis moves to the z reference position by publishing the updated robot position. After publishing the case 'error_xy' gets called.
- Case 'error_xy' moves the x and y axis to the reference position. After the z axis reaches the reference position the x and y position gets updated and the robot position is published. The next error case 'error' gets called after publishing.
- Case 'error' checks if the x and y position is reached and calls the next case 'dead'
- Casse 'dead' The placeholder pass gets called and after that the programm is in a dead state.

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

- /home/seb/ros2_ws/src/logic/logic/logic.py

3.3 detection_node.tracker.EuclideanDistTracker Class Reference

An EuclideanDistTracker instance is used to track and detected Objects, in an image.

Public Member Functions

- def __init__ (self)

  Initializes a new EuclideanDistTracker instance.
- def update (self, objects_rect)

  Updates the euclideandisttracker to track objects.
- def getSample (self, frame, center_x, id)

  Creates an image of an object if it is in a certain range.
- def detectObject (self, frame)

  Detects objects in a given image.
- def trackObject (self, detections)

  Tracks objects by using update function.
- def find_center_of_mass (self, image, x, y)

  Calculates the center of mass of an image.

3.3.1 Detailed Description

An EuclideanDistTracker instance is used to track and detected Objects, in an image.

To work properly the following conditions must be met.

- The given image must be a binary image.
- Objects need to be big enough.
- Objects need to be bright enough.
- Objects should not contact each other to detect an track them properly

3.3.2 Member Function Documentation

3.3.2.1 detectObject()

```
def detection_node.tracker.EuclideanDistTracker.detectObject (
              self,
              frame )
```

Detects objects in a given image.

Parameters

frame	The input image

Returns

detections A list which contains the x and y coordinates and the width and heigth of the bounding boxes

3.3.2.2 find_center_of_mass()

```
def detection_node.tracker.EuclideanDistTracker.find_center_of_mass (
              self,
              image,
              x,
              y )
```

Calculates the center of mass of an image.

Parameters

image	The image
x	The offset in x direction
y	The offset in y direction

3.3.2.3 getSample()

```
def detection_node.tracker.EuclideanDistTracker.getSample (
              self,
              frame,
              center_x,
              id )
```

Creates an image of an object if it is in a certain range.

Parameters

frame	The image with objects of which samples should be taken
center←↩ _x	The x-coordinate of the object center point
id	The id of the object

Returns

publish A boolean value that is true if a sample was taken and false if not

id_img A ros2 msg which contains the id of the object and the taken sample of the object

3.3.2.4 trackObject()

```
def detection_node.tracker.EuclideanDistTracker.trackObject (
              self,
              detections )
```

Tracks objects by using update function.

Parameters

detections	A list which contains the x and y coordinates and the width and heigth of the bounding boxes

Returns

id The id of the tracked object

cx The x-coordinate of the object center point

cy the y-coordinate of the object center point

3.3.2.5 update()

```
def detection_node.tracker.EuclideanDistTracker.update (
              self,
              objects_rect )
```

Updates the euclideandisttracker to track objects.

Parameters

objects_rect	A list which contains the x and y coordinates and the width and heigth of the bounding boxes

Returns

obcets_bbs_ids A list which contains the ids and the boundings boxes of the tracked objects

Updates the object tracking based on the given object rectangles.

Args: objects_rect (list): A list of rectangles (x, y, w, h) representing the detected objects.

Returns: list: A list of object bounding boxes (x, y, w, h, id) that have been updated.

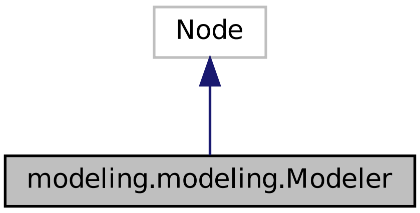
The documentation for this class was generated from the following file:

- /home/seb/ros2_ws/src/detection_node/detection_node/tracker.py

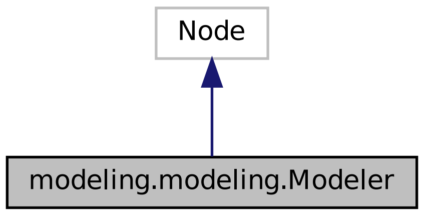
3.4 modeling.modeling.Modeler Class Reference

A Modeler instance is used to create a ROS2 node for movement modeling.

Inheritance diagram for modeling.modeling.Modeler:



Collaboration diagram for modeling.modeling.Modeler:



Public Member Functions

- def __init__ (self)

  Initializes the PositionGrouping node.
- def position_callback (self, msg_in)

  This callback function is called automatically when an IdPosTime message is received.
- def get_positions_by_id (self, id)

  Removes all items with the specified ID from a list and returns them as a separate list.
- def add_position (self, id, timestamp, pos_x, pos_y)

  Adds a newelement to the list , sorts the list by ID and timestamp and Checks if the object is still in the tracking area.

Static Public Member Functions

- def calculate_movement (list)

  Calculates movement data from a list of points.

3.4.1 Detailed Description

A Modeler instance is used to create a ROS2 node for movement modeling.

3.4.2 Member Function Documentation

3.4.2.1 add_position()

```
def modeling.modeling.Modeler.add_position (
              self,
              id,
              timestamp,
              pos_x,
              pos_y )
```

Adds a newelement to the list , sorts the list by ID and timestamp and Checks if the object is still in the tracking area.

Parameters

id	The ID of the new element.
timestamp	The timestamp of the new element.
pos_x	The x-position of the new element.
pos_y	The y-position of the new element.

Returns

None

3.4.2.2 calculate_movement()

```
def modeling.modeling.Modeler.calculate_movement (
              list )  [static]
```

Calculates movement data from a list of points.

Parameters

list	(List): A list of points, where each point is represented as a 3-tuple (x, y, t). Here, x denotes the horizontal position, y denotes the vertical position, and t denotes time.

Returns

Tuple: A 4-tuple containing the calculated movement data: (shifted_horizontal_position, vertical_position, velocity, shifted_time)

3.4.2.3 get_positions_by_id()

```
def modeling.modeling.Modeler.get_positions_by_id (
              self,
              id )
```

Removes all items with the specified ID from a list and returns them as a separate list.

Parameters

list_for_element	(list): The list to remove elements from.
id	The ID of the items to be removed.

Returns

list: A separate list containing the removed items.

3.4.2.4 position_callback()

```
def modeling.modeling.Modeler.position_callback (
              self,
              msg_in )
```

This callback function is called automatically when an IdPosTime message is received.

It processes the received message, adding the position and time data to the internal storage. If the Id is new, it calculates the movement from stored positions and publishes the result.

Parameters

msg↩_in	(IdPosTime): The received IdPosTime message containing the Id, Position, and Time data.

Returns

none

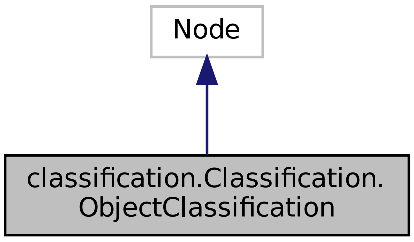
The documentation for this class was generated from the following file:

- /home/seb/ros2_ws/src/modeling/modeling/modeling.py

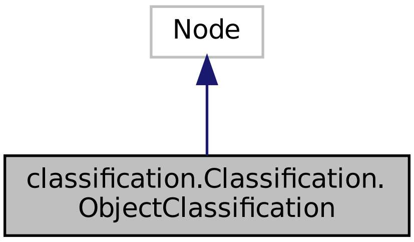
3.5 classification.Classification.ObjectClassification Class Reference

An ObjectClassification instance is used to create a ROS2 node for object classification.

Inheritance diagram for classification.Classification.ObjectClassification:



Collaboration diagram for classification.Classification.ObjectClassification:



Public Member Functions

- def __init__ (self)

  Initializes the object classification.
- def image_callback (self, IdSample)

  Callback function for the input image.
- def find_center_of_mass (self, image)

  Finds the centroid (center of mass) of the largest object in a binary image.
- def find_gripping_point_vector (self, gripping_point, center_point)

  Calculates the vector between a gripping point and a center point.
- def feature_extract (self, image)

  Classifies an image as a cat or unicorn.

3.5.1 Detailed Description

An ObjectClassification instance is used to create a ROS2 node for object classification.

3.5.2 Member Function Documentation

3.5.2.1 feature_extract()

```
def classification.Classification.ObjectClassification.feature_extract (
              self,
              image )
```

Classifies an image as a cat or unicorn.

Parameters

image	(str): Path to the image.

Returns

features (list): List of extracted features of the image.

gripping_point : A tuple containing the gripping point as (x, y) coordinates.

gravity : List containing the gravity_x and the gravity_y

3.5.2.2 find_center_of_mass()

```
def classification.Classification.ObjectClassification.find_center_of_mass (
              self,
              image )
```

Finds the centroid (center of mass) of the largest object in a binary image.

Parameters

image	(ndarray): The binary image containing objects.

Returns

tuple or None: A tuple representing the centroid coordinates (cx, cy) of the largest object. If no objects are found (no contours detected), it returns None.

3.5.2.3 find_gripping_point_vector()

```
def classification.Classification.ObjectClassification.find_gripping_point_vector (
              self,
              gripping_point,
              center_point )
```

Calculates the vector between a gripping point and a center point.

Parameters

gripping_point	(tuple): The coordinates of the gripping point.
center_point	(tuple): The coordinates of the center point.

Returns

A tuple representing the vector from the center point to the gripping point. The tuple contains the x-component and y-component of the vector.

3.5.2.4 image_callback()

```
def classification.Classification.ObjectClassification.image_callback (
              self,
              IdSample )
```

Callback function for the input image.

Parameters

Image	ROS image.

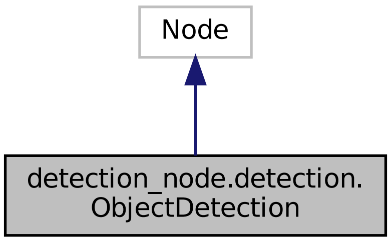
The documentation for this class was generated from the following file:

- /home/seb/ros2_ws/src/classification/classification/Classification.py

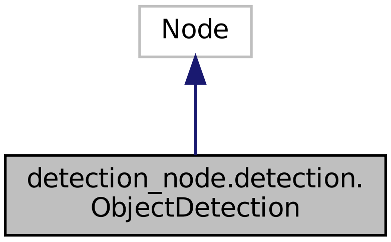
3.6 detection_node.detection.ObjectDetection Class Reference

An ObjectDetection instance is used to create a ROS2 node for image detection.

Inheritance diagram for detection_node.detection.ObjectDetection:



Collaboration diagram for detection_node.detection.ObjectDetection:



Public Member Functions

- def __init__ (self)

  Initializes a new ObjectDetection instance.
- def image_callback (self, msg)

  Callback function for processing image messages.

3.6.1 Detailed Description

An ObjectDetection instance is used to create a ROS2 node for image detection.

The node subscribes the topic "preprocessed_stream", processes incoming images from that topic and publishes the processed images on the topic "id_sample" for classification and publishes the processed images on the topic "id_pos_time" for modelling.

The detection includes:

- Detecting objects in the image.
- Tracking the objects with their positions.
- Creating samples for classification.

3.6.2 Member Function Documentation

3.6.2.1 image_callback()

```
def detection_node.detection.ObjectDetection.image_callback (
              self,
              msg )
```

Callback function for processing image messages.

This function detects and tracks objects in the image. It also creats samples for the classification.

Parameters

msg	The ROS2 message preprocessed_stream containing the binary image of the belt.

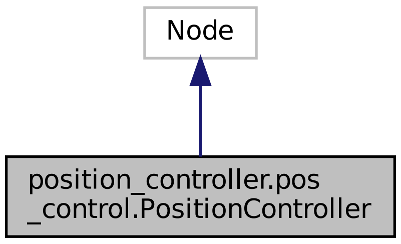
The documentation for this class was generated from the following file:

- /home/seb/ros2_ws/src/detection_node/detection_node/detection.py

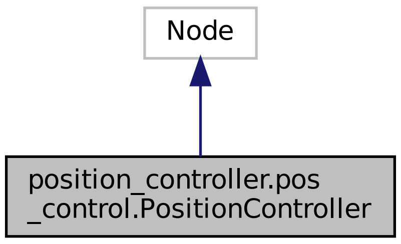
3.7 position_controller.pos_control.PositionController Class Reference

A PositionController instance is used to create a ROS2 node for robot position control.

Inheritance diagram for position_controller.pos_control.PositionController:



Collaboration diagram for position_controller.pos_control.PositionController:



Public Member Functions

- def __init__ (self)

    Initializes a new PositionController instance.

Static Public Member Functions

- def controllerCommand_callback (self, msg)

    Callback function to get the desired X, Y and Z position and the offset for the x velocity.
- def robotPostion_callback (self, msg)

    Callback function to get the current position X, Y and Z and print the velocity in the consol.
- def calculate_control_signal (self)

    Calculates the new velocity for each step motor in dependence of the current position and the desired position.

3.7.1 Detailed Description

A PositionController instance is used to create a ROS2 node for robot position control.

The node subscribes the topic "robot_position" and "controller_command", calculates the new velocity for each step motor in dependence of the current position and the desired position and publishes the calculated velocity on the topic "robot_command".

The pos_control includes:

- Calculation of the new velocity for each step motor in dependence of the current position and the desired position.
- Regulate the velocity to the maximum velocity.
- Regulate the velocity jump to the maximum acceleration.
- Publish the calculated velocity for each step motor and the grip value as a RobotCmd message.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 __init__()

```
def position_controller.pos_control.PositionController.__init__ (
              self )
```

Initializes a new PositionController instance.

Parameters

k	Proportional gain value for the velocity calculation.
max_vel	Array with three values for the maximum velocity of each step motor.
max_acc	Array with thee values for the maximum velocity of each step motor.
desired_pos	Desired position for each step motor.
current_pos	Current position for each step motor.
velocity	Current velocity for each step motor.
vel_x_offset	Offset for the x velocity.
grip	Boolean value to control the robot gripper.
subscriptionCurrent	ROS2 subscription for the current position of the robot from the RobotPos topic.
subscriptionDesired	ROS2 subscription for the desired position of the robot from the ConCmd topic.
publisher	ROS2 publisher for the calculated velocity to the RobotCmd topic.

3.7.3 Member Function Documentation

3.7.3.1 calculate_control_signal()

```
def position_controller.pos_control.PositionController.calculate_control_signal (
              self )  [static]
```

Calculates the new velocity for each step motor in dependence of the current position and the desired position.

For the velocity in x direction the offset has to be added because with this the system is in rest position. The velocity is set to the maximum velocity when the velocity is higher than the maximum velocity. The velocity is set to the maximum acceleration when the velocity jump is higher than the maximum acceleration.

Returns

robot_command Returns the calculated velocity for each step motor and the grip value as a RobotCmd message.

3.7.3.2 controllerCommand_callback()

```
def position_controller.pos_control.PositionController.controllerCommand_callback (
              self,
              msg )  [static]
```

Callback function to get the desired X, Y and Z position and the offset for the x velocity.

The grip value is set to the desired grip value.

Parameters

msg	The ConCmd message containing the desired position and velocity offset.

3.7.3.3 robotPostion_callback()

```
def position_controller.pos_control.PositionController.robotPostion_callback (
              self,
              msg )  [static]
```

Callback function to get the current position X, Y and Z and print the velocity in the consol.

Parameters

msg	The RobotPos message containing the current position.

Returns

robot_command returns the calculated velocity for each step motor and the grip value as a RobotCmd message.

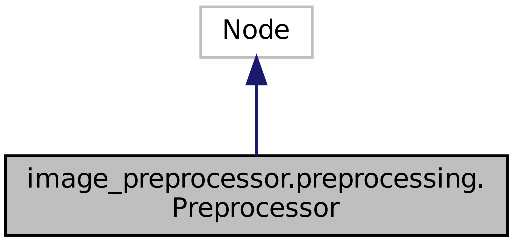
The documentation for this class was generated from the following file:

- /home/seb/ros2_ws/src/position_controller/position_controller/pos_control.py

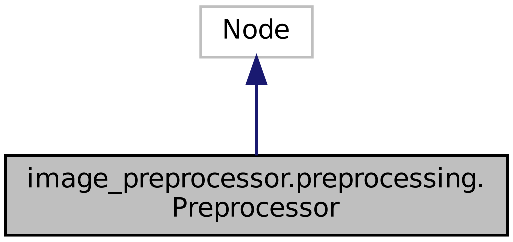
3.8 image_preprocessor.preprocessing.Preprocessor Class Reference

A Preprocessor instance is used to create a ROS2 node for image preprocessing.

Inheritance diagram for image_preprocessor.preprocessing.Preprocessor:



Collaboration diagram for image_preprocessor.preprocessing.Preprocessor:



Public Member Functions

- def __init__ (self, queue, length, m, n)

  Initializes a new Preprocessor instance.
- def preprocess (self, input_msg)

  Preprocesses incoming images from the camera.

3.8.1 Detailed Description

A Preprocessor instance is used to create a ROS2 node for image preprocessing.

The node subscribes the topic "camera_stream", preprocesses incoming images from that topic and publishes the preprocessed images on the topic "preprocessed_stream".

The preprocessing includes:

- Cutting out the belt area.
- Transformation of the perspective.
- Conversion to a binary image.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 __init__()

```
def image_preprocessor.preprocessing.Preprocessor.__init__ (
              self,
              queue,
              length,
              m,
              n )
```

Initializes a new Preprocessor instance.

Parameters

queue	The queue size of the camera subscription and the publishing of preprocessed images.
length	The length (witdh) of the preprocessed images in pixels.
m	Number of scale squares to the end of the belt area of interest, counted from the start of the scale.
n	Number of scale squares to the beginning of the belt area of interest, counted from the start of the scale.

3.8.3 Member Function Documentation

3.8.3.1 preprocess()

```
def image_preprocessor.preprocessing.Preprocessor.preprocess (
              self,
              input_msg )
```

Preprocesses incoming images from the camera.

- Detects the scale once and asks whether the detection was successful. Repeats the detection, if the detected area was declined. For following images the detected area is kept.
- Cuts out the belt area and transforms it to a rectangle.
- Converts the transformed belt area to a binary image.
- Publishes the preprocessed image.

Parameters

input_msg	Image to be preprocessed.

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

- /home/seb/ros2_ws/src/image_preprocessor/image_preprocessor/preprocessing.py

3.9 image_preprocessor.scaleDetection.ScaleDetector Class Reference

A ScaleDetector instance is used to find a scale, consisting of n alternating black and white squares, in an image.

Public Member Functions

- def __init__ (self, delta1, delta2, n, phi1, phi2, sigma1, sigma2, m, count, accuracy)

  Initializes a new ScaleDetector instance.
- def analyseLine (self, ends, anchor)

  Calculates the positions of the edges between black and white in the binary gray values of a line and checks by the distances between them, if a line is likely to go through <count> squares of the scale.
- def scanArea (self, center)

  Checks the nearby area of the current valid line for another valid line, that is assumed to intersect an edge between a black and a white scale square further left in the image.
- def adaptBorders (self, left, right)

  Trys to find the upper and lower border of the scale by moving a valid line in small steps until it gets invalid.
- def detectScale (self, img)

  Trys to find the upper and lower border of a scale in the given image.

Static Public Member Functions

- def calculateLineEnds (shape, anchors, vectors)

  Calculates the intersection points between some lines and the border of an image.
- def calculateLine (ends, anchor)

  Calculates the positions of all pixels on a line between two end points.

3.9.1 Detailed Description

A ScaleDetector instance is used to find a scale, consisting of n alternating black and white squares, in an image.

To work properly the following conditions must be met.

- The beginning of the scale points to the left.
- One square of the scale in the image is wider than one fortieth of the image width.
- More than 20 squares, starting from the beginning of the scale are fully visible, without a gap.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 __init__()

```
def image_preprocessor.scaleDetection.ScaleDetector.__init__ (
              self,
              delta1,
              delta2,
              n,
              phi1,
              phi2,
              sigma1,
              sigma2,
              m,
              count,
              accuracy )
```

Initializes a new ScaleDetector instance.

Parameters

delta1	Radius step width while the whole image is scanned.
delta2	Orthogonal step width while the nearby area of a valid line is scanned.
n	Number of orthogonal steps done in each direction while scanning the nearby area.
phi1	Angle step width while the whole image is scanned.
phi2	Angle step width while the nearby area of a valid line is scanned.
sigma1	Start angle step witdh for the border adaption.
sigma2	Minimal angle step witdh for the border adaption.
m	Number of angle steps done in each direction while scanning the nearby area.
count	Number of gray value edges with minimum absolute distance and low varity in distance ratio in a row, the line must go trough so it is assumed to go through the scale.
accuracy	The maximum value the ratio between the distances from an edge to the previous edge and the next edge can deviate from one.

3.9.3 Member Function Documentation

3.9.3.1 adaptBorders()

```
def image_preprocessor.scaleDetection.ScaleDetector.adaptBorders (
              self,
              left,
              right )
```

Trys to find the upper and lower border of the scale by moving a valid line in small steps until it gets invalid.

Parameters

left	The left point of the valid line.
right	The right point of the valid line.

Returns

Array of shape (2,<count>+1,2) with the positions where it was assumed that the borders intersect the edges between white and black squares.

3.9.3.2 analyseLine()

```
def image_preprocessor.scaleDetection.ScaleDetector.analyseLine (
              self,
              ends,
              anchor )
```

Calculates the positions of the edges between black and white in the binary gray values of a line and checks by the distances between them, if a line is likely to go through <count> squares of the scale.

Parameters

ends	The end points of the line.
anchor	One determined position on the line, to which the output will be aligned.

Returns

If the check is positive, the positions where it was assumed that the line intersects the edges between white and black squares.

3.9.3.3 calculateLine()

```
def image_preprocessor.scaleDetection.ScaleDetector.calculateLine (
              ends,
              anchor )  [static]
```

Calculates the positions of all pixels on a line between two end points.

Parameters

ends	The end points of the line.
anchor	One determined position, that will be in the output.

Returns

Array of pixel positions.

3.9.3.4 calculateLineEnds()

```
def image_preprocessor.scaleDetection.ScaleDetector.calculateLineEnds (
              shape,
              anchors,
              vectors )  [static]
```

Calculates the intersection points between some lines and the border of an image.

Each line is defined by an anchor point on the line and a vector of length one, giving the direction of the line.

Parameters

shape	The shape of the image.
anchors	Array of anchor points.
vectors	Array of line vectors.

Returns

Array of intersection points.

3.9.3.5 detectScale()

```
def image_preprocessor.scaleDetection.ScaleDetector.detectScale (
              self,
              img )
```

Trys to find the upper and lower border of a scale in the given image.

Parameters

img	The image to search in.

Returns

If a scale was detected, an array of shape (2,<count>+1,2) with the positions where it was assumed that the borders intersect the edges between white and black squares.

3.9.3.6 scanArea()

```
def image_preprocessor.scaleDetection.ScaleDetector.scanArea (
              self,
              center )
```

Checks the nearby area of the current valid line for another valid line, that is assumed to intersect an edge between a black and a white scale square further left in the image.

Parameters

The	center point of the current valid line.

Returns

The positions where it was assumed that the line with the most left intersection, intersects the edges between white and black squares.

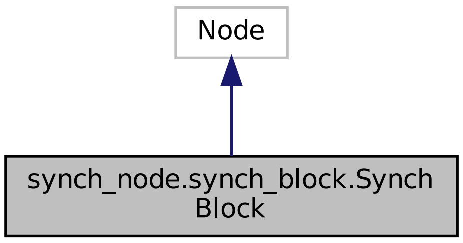
The documentation for this class was generated from the following file:

- /home/seb/ros2_ws/src/image_preprocessor/image_preprocessor/scaleDetection.py

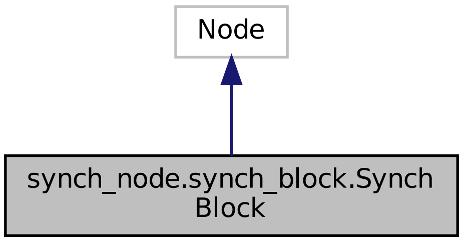
3.10 synch_node.synch_block.SynchBlock Class Reference

A SynchBlock instance is used to create a ROS2 node for synchronisation.

Inheritance diagram for synch_node.synch_block.SynchBlock:



Collaboration diagram for synch_node.synch_block.SynchBlock:



Public Member Functions

- def __init__ (self)

  Initializes a new SynchBlock instance.
- def class_callback (self, msg)

  Callback function for processing classification messages.
- def pos_callback (self, msg)

  Callback function for processing modelling messages.
- def publish_synch (self, pos_x, pos_y, vel, cl, time)

  This function creates and publishes a PosVelClass message with the 4 given integers after converting them to int32.
- def calculatePoint (self, x1, y1, vector_x, vector_y)

  Calculating the gripping point based on the given vector.
- def calculateActualCoordinates (self, x, y)

  Calculates the actual position of the object relativ to the gripper.

3.10.1 Detailed Description

A SynchBlock instance is used to create a ROS2 node for synchronisation.

The node subscribes the topics "id_class_vec" from the classification and "id_pos_vel_time" from the modelling. It processes incoming data from those topics and publishes the synchronised message on the topic "pos_vel_class" for the logic.

The synchronisation includes:

- Data from the classification.
- Data from the modelling.

3.10.2 Member Function Documentation

3.10.2.1 calculateActualCoordinates()

```
def synch_node.synch_block.SynchBlock.calculateActualCoordinates (
              self,
              x,
              y )
```

Calculates the actual position of the object relativ to the gripper.

Parameters

x	The x-position of the object in the image.
y	The y-position of the object in the image.

Returns

actual_pos_x The calculated x_position of the object relativ to the gripper.

actual_pos_y The calculated y_position of the object relativ to the gripper.

3.10.2.2 calculatePoint()

```
def synch_node.synch_block.SynchBlock.calculatePoint (
              self,
              x1,
              y1,
              vector_x,
              vector_y )
```

Calculating the gripping point based on the given vector.

Parameters

x1	The x-positon of the center of mass.
y1	The y-positon of the center of mass.
vector↩ _x	The fist parameter of the vector.
vector↩ _y	The second parameter of the vector.

Returns

new_x The x-postion of the gripping point.

new_y The y-postion of the gripping point.

3.10.2.3 class_callback()

```
def synch_node.synch_block.SynchBlock.class_callback (
              self,
              msg )
```

Callback function for processing classification messages.

Parameters

msg	The ROS2 message IdClass containing the ID and the classification of an object.

3.10.2.4 pos_callback()

```
def synch_node.synch_block.SynchBlock.pos_callback (
              self,
              msg )
```

Callback function for processing modelling messages.

This function checks if the message can be ignoered because its neither a cat or unicorn if this is true it compares the id of the message with the id of the class-message, if they are the same it publishes the new message.

Parameters

msg	The ROS2 message IdPosVel containing the ID, position and the velocity of an object.

3.10.2.5 publish_synch()

```
def synch_node.synch_block.SynchBlock.publish_synch (
              self,
              pos_x,
              pos_y,
              vel,
              cl,
              time )
```

This function creates and publishes a PosVelClass message with the 4 given integers after converting them to int32.

Parameters

pos↵_x	The x-position of an object.
pos↵_y	The y-position of an object.
vel	The velocity of an object.
cl	The classification of an object.
time	The timestemp of the current object.

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

- /home/seb/ros2_ws/src/synch_node/synch_node/synch_block.py