Baxter teleoperation project

1.0

Generated by Doxygen 1.8.19

Chapter 1

SofAr-project

The project's goal was to design and implement a software component for the teleoperation of the Baxter robot simulation in Coppelia. The teleoperation works as follows: the human operator moves its arm keeping a smartphone into its hand, and the sensor data from the smartphone's IMU is sent to the software and used to allow Baxter's end-effector to follow the trajectory and orientation of the human hand.
The project's original goal was actually to not only track the end-effector's configuration, but also to replicate the motion of the human arm into Baxter's, using Mocap technology; this idea, as well as the objective of using the software on the real robot, were later rejected due to the Covid emergence and the consequent impossibility of access the EMARO Lab.
Unfortunately, the elimination of the position measurement with the motion capture do not allow the perfect tracking, but the developed modules try to work inside this limitation.

1.0.1 Prerequisites

In order to run this software, the following prerequisites are needed:

- `ROS kinetic`,
- `CoppeliaSim Edu V4` (which has to be linked with ROS),
- `Ubuntu 16.04`(Other Ubuntu versions may work, but this is the offcialy supported one by ROS kinetic, as well as the one on which all this code was produced.)

Then, it is required to install the app on an Android mobile phone. Unzip *org.ros.android.android_tutorial_camera_imu_1.0.apk* in order to install the *CameraImu* app in your smartphone. Warning: the app works best with Android 8.1 or older; earlier OS versions may cause frequent freezes/crashes
Moreover, launching the software on a virtual machine cause great instability, so, it is strongly adviced against.
In the testing phase, the following hardware characteristic were found to work discretely, which is why they are going to be taken as advised configuration.
Characteristics:

- i5 processors, 3.1GHz, 2 cores
- 8GB DDR4 SDRAM
- 256 GB SSD memory
- 103.5GB dedicated to Ubuntu (in testing 39.2GB of memory were used)

1.0.2 Installing

In order to have a working version of this package running on your computer, you need to:

- Place the package in the src folder of your src foulder of the catkin workspace, and having it named "SofAr-project"
- Have the Coppelia environment foulder in any place under the HOME directory (it is advised to put it on the Desktop or directly in HOME)
- If you don't have the following libraries installed on your system procede with this code
```
python -m pip install numpy
python -m pip install -U matplotlib
sudo apt-get install python-pandas
sudo apt-get install python-scipy
sudo apt-get install ros-kinetic-cmake-modules
```
- Change the current directory to "SofAr-project" and activate the install.sh script
```
cd "Your catkin workspace"/src/SofAr-project
chmod +x install.sh
```
- Install the package via dedicated script
```
./install.sh
```

1.0.3 Running the tests: running the simulation without the smartphone to inspect that the simulation is working

In order to see a test of the working system (without the smartphone application sending signals), proceed with the following commands
```
cd "Your catkin workspace"/src/SofAr-project
./launcher_test.sh
```

When all components are open, find the user interface (the terminal where you have launched the system and the only one asking for the user input).
There you can type "help" and enter to obtain the command list, or you can type "start" and enter to start the animation.
At any moment, you can type "pause" or "stop" to interrupt the simulation and respectively stay in the position or return to the starting one.
When stopped (not paused) you can type "set_default" to set the starting configuration.
Once you are done, you can type "exit" and press enter to close all this project components.
The movement that should be obtained in this test is the increment of the first joint angular position, as can be seen in the animation.

1.0.4 Deployment

Here we are going to present how to use the software to attempt a tracking of the user arm using the data from the smartphone sensor.
Note: when you are asked to take the phone in your hand during the procedures, you are to take it in your right hand, arm stretched, horizontal and at more or less 45 degrees with respect to the plane of the chest, with the smartphone screen vertical, facing you, and the camera in the direction of the thumb. When you give the command "calibration", you can take it as you are most comfortable.
To use the whole system at once, the following action have to take place:

- Start the smartphone application

- Verify the IP address to be inserted in the app

```
hostname -I
```

- Start the software on the computer with the following commands
```
cd "Your catkin workspace"/src/SofAr-project
./launcher.sh
```
- Find the terminal in which you started the software and follow the instructions written on it always checking the app is working properly (you can look at the terminal printing the linear acceleration, orientation and angular velocity)
- Once all the instructions are completed and the Coppelia environment opened, with the smartphone hand still type "calibration" and press enter
- When you are done you can type "start" and procede with the movement

After the command "calibration", you can apply every other string presented in the section "Running test".
When you type start, be sure you are more or less in the same configuration as baxter.

1.0.5 Authors

Marco Demutti, Matteo Dicenzi, Vincenzo Di Pentima, Elena Merlo, Matteo Palmas, Andrea Pitto, Emanuele Rosi, Chiara Saporetti, Giulia Scorza Azzarà, Luca Tarasi, Simone Voto, Gerald Xhaferaj

Chapter 2

imu_calib

This repository contains a ROS package with tools for computing and applying calibration parameters to IMU measurements.

2.1 Usage

The package contains two nodes. The first computes the accelerometer calibration parameters and saves them to a YAML file, and needs to be run only once. After you have run this node to generate the YAML calibration file, the second node uses that file to apply the calibration to an uncalibrated IMU topic to produce a calibrated IMU topic.

2.2 Nodes

2.2.1 do_calib

Computes the accelerometer calibration parameters. It should be run directly with a rosrun in a terminal rather than from a launch file, since it requires keyboard input. After receiving the first IMU message, the node will prompt you to hold the IMU in a certain orientation and then press Enter to record measurements. After all 6 orientations are complete, the node will compute the calibration parameters and write them to the specified YAML file.

The underlying algorithm is a least-squares calibration approach based on and similar to that described in S↩
TMicroeletronics Application Note `AN4508`. Due to the nature of the algorithm, obtaining a good calibration requires fairly accurate positioning of the IMU along each of its axes.

2.2.1.1 Topics

2.2.1.1.1 Subscribed Topics

- `imu` (sensor_msgs/Imu)
  The raw, uncalibrated IMU measurements

2.2.1.2 Parameters

- `~calib_file` (string, default: "imu_calib.yaml")
  The file to which the calibration parameters will be written
- `~measurements` (int, default: 500)
  The number of measurements to collect for each orientation
- `~reference_acceleration` (double, default: 9.80665)
  The expected acceleration due to gravity

2.2.2 apply_calib

Applies the accelerometer calibration parameters computed by the do_calib node. Also optionally (enabled by default) computes the gyro biases at startup and subtracts them off.

2.2.2.1 Topics

2.2.2.1.1 Subscribed Topics

- `raw` (sensor_msgs/Imu)
  The raw, uncalibrated IMU measurements

2.2.2.1.2 Published Topics

- `corrected` (sensor_msgs/Imu)
  The corrected, calibrated IMU measurements

2.2.2.2 Parameters

- `~calib_file` (string, default: "imu_calib.yaml")
  The file from which to read the calibration parameters
- `~calibrate_gyros` (bool, default: true)
  Whether to compute gyro biases at startup and subsequently subtract them off
- `~gyro_calib_samples` (int, default: 100)
  The number of measurements to use for computing the gyro biases

Chapter 3

Namespace Index

3.1 Packages

Here are the packages with brief descriptions (if available):

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 calibration2 Namespace Reference

Functions

- def imu_ee_calibration (data)

  Computes the orientation between 0 and global frame using an initial configuration.
- def simulate_callback (data)

  Waits for the start from handle simulation topic.
- def calibrate_orientation ()

Variables

- pub_rot_matrices
- R0e
- int start = 0

6.1.1 Detailed Description

```
@package pyexample
```

6.1.2 Function Documentation

6.1.2.1 calibrate_orientation()

```
def calibration2.calibrate_orientation ( )
```

Definition at line 73 of file calibration2.py.

```
73 def calibrate_orientation():
74 
75     # In ROS, nodes are uniquely named. If two nodes with the same
76     # name are launched, the previous one is kicked off. The
77     # anonymous=True flag means that rospy will choose a unique
78     # name for our 'listener' node so that multiple listeners can
79     # run simultaneously.
80     rospy.init_node('calibrate_orientation', anonymous=True)
81 
82     rospy.Subscriber("smartphone", Imu, imu_ee_calibration)
83     rospy.Subscriber("handleSimulation", Int8, simulate_callback)
84 
85     rospy.spin()
86 
87 
```

6.1.2.2 imu_ee_calibration()

```
def calibration2.imu_ee_calibration (
                data )
```

Computes the orientation between 0 and global frame using an initial configuration.

Parameters

data	inertial data coming from smartphone. The focus is on the orientation info given by a quaternion.

Definition at line 29 of file calibration2.py.

```
29 def imu_ee_calibration(data):
30     """!
31     Computes the orientation between 0 and global frame using an
32     initial configuration.
33     @param data: inertial data coming from smartphone. The focus is on the
34     orientation info given by a quaternion.
35     """
36 
37     global start
38 
39     if start == 1:
40 
41         orient = [data.orientation.x, data.orientation.y,
42                   data.orientation.z, data.orientation.w]
43 
44         Ttemp = tf.transformations.quaternion_matrix(
45             (orient[0], orient[1], orient[2], orient[3]))
46         Rimu_global = Ttemp[:3, :3]
47 
48         R0_global = np.dot(R0e, Rimu_global)
49 
50         R = util.init_float64_multiarray(9, 1)
51         R0_global = R0_global.reshape(9, 1)
52         R.data = R0_global
53 
54         start = 0
55 
56         rospy.logerr("Setup done!")
57 
58         pub_rot_matrices.publish(R)
59 
60 
```

6.1.2.3 simulate_callback()

```
def calibration2.simulate_callback (
              data )
```

Waits for the start from handle simulation topic.

Parameters

data	integer used to understand which part of the simulation is on.

Definition at line 61 of file calibration2.py.

```
61 def simulate_callback(data):
62     """!
63     Waits for the start from handle simulation topic.
64     @param data: integer used to understand which part of the simulation is on.
65     """
66 
67     global start
68 
69     if data.data == 3:
70         start = 1
71 
72 
```

6.1.3 Variable Documentation

6.1.3.1 pub_rot_matrices

```
calibration2.pub_rot_matrices
```

Initial value:

```
=  rospy.Publisher(
    'rot_matrices', Float64MultiArray, queue_size=10)
```

Definition at line 14 of file calibration2.py.

6.1.3.2 R0e

```
calibration2.R0e
```

Initial value:

```
=  np.array([[0,  0,  1],
             [0,  1,  0],
             [-1,  0,  0]])
```

Definition at line 21 of file calibration2.py.

6.1.3.3 start

```
int calibration2.start = 0
```

Definition at line 26 of file calibration2.py.

6.2 clipping Namespace Reference

Functions

- def dataFileInitializer ()

  Function used initialize the files in which sensor data will be stored; it will generate three files in the 'output' folder.
- def lin_acc_compensate (lin_acc_no_g, threshold)

  Function used to filter unwanted minimal incoming data fluctuations, due to noise as well as human operator shake.
- def storeDataInFiles (fileName, modality, data)

  Function which stores incoming data into .csv files @params fileName name of the generated file @params modality parameter requested by the open() function: 'a' stands for 'open for appending at the end of the file without truncating it' @params data data to be inserted in the .csv file (orientation/linear acceleration/angular velocity)
- def callback (data)

  This is the callback function: it is invoked every time there is incoming data and has the duty of calling all the previously mentioned functions as well as eulerAnglesToRotationMatrix().
- def listener ()

  The listener is used to instantiate the homonymous node and to subscribe to the android/imu_corected topic, from which it receives incoming data of smarthpone accelerometers.
- def talker (msg)

  The talker is used to publish the refined data on the so called 'smartphone' topic.

Variables

- script_dir = os.path.dirname(__file__)
- string rel_path1 = "../output/lin_acc.csv"
- abs_file_path1 = os.path.join(script_dir, rel_path1)
- string rel_path2 = "../output/orientation.csv"
- abs_file_path2 = os.path.join(script_dir, rel_path2)
- string rel_path3 = "../output/angVel.csv"
- abs_file_path3 = os.path.join(script_dir, rel_path3)
- string rel_path_gravity = "../calibration/gravity.csv"
- abs_file_gravity = os.path.join(script_dir, rel_path_gravity)
- int flagWriteData = 1
- int index = 1
- float delta = 0.05
- list lin_acc_no_g = [0, 0, 0]
- list angular_velocity = [0, 0, 0]
- list orientation = [0, 0, 0, 0]
- list g = [0 ,0 ,0]
- list max_lin_acc = [0, 0, 0]
- float safety_coeff = 1.1
- int counter = 0
- int maxIteration = 50
- temp = pd.read_csv(abs_file_gravity, names=['X', 'Y', 'Z'], header=0, decimal=',')

6.2.1 Detailed Description

```
Documentation for the clipping tool.
This piece of code receives data from the imu sensor and removes noise from
the incoming linear acceleration through a clipping algorithm.
```

6.2.2 Function Documentation

6.2.2.1 callback()

```
def clipping.callback (
              data )
```

This is the callback function: it is invoked every time there is incoming data and has the duty of calling all the previously mentioned functions as well as eulerAnglesToRotationMatrix().

After these calls, it finally invokes the talker(). In order to achieve all of this, it translates incoming quaternions into euler angles beforehand

Parameters

data	data incoming from the imu sensor

Definition at line 97 of file clipping.py.

```
97 def callback(data):
98     """!
99     This is the callback function: it is invoked every time there is incoming data and
100     has the duty of calling all the previously mentioned functions as well as
       eulerAnglesToRotationMatrix().
101     After these calls, it finally invokes the talker(). In order to achieve all of this,
102     it translates incoming quaternions into euler angles beforehand
103     @param data data incoming from the imu sensor
104     """
105     global index, lin_acc_no_g, angular_velocity, orientation, counter, max_lin_acc, delta
106 
107     # get data
108     orientation = [data.orientation.x, data.orientation.y,
109                    data.orientation.z, data.orientation.w]
110 
111     # transform quaternion to euler angles
112     tempAngles = tf.transformations.euler_from_quaternion(orientation, "sxyz")
113 
114     rot_matrix = eulerAnglesToRotationMatrix(tempAngles)
115 
116     angular_velocity = [data.angular_velocity.x,
117                         data.angular_velocity.y, data.angular_velocity.z]
118     linear_acceleration = [data.linear_acceleration.x,
119                            data.linear_acceleration.y, data.linear_acceleration.z]
120 
121     lin_acc_no_g = removeGravity(linear_acceleration, rot_matrix, g)
122 
123     if counter < maxIteration:
124         #compute root of gravity
125         g_root = math.sqrt(
126             pow(lin_acc_no_g[0], 2) + pow(lin_acc_no_g[1], 2) + pow(lin_acc_no_g[2], 2))
127 
128         max_root = math.sqrt(
129             pow(max_lin_acc[0], 2) + pow(max_lin_acc[1], 2) + pow(max_lin_acc[2], 2))
130 
131         if g_root > max_root:  # if true, the 'maximum linear acceleration vector' has to be updated
132             max_lin_acc = lin_acc_no_g
133 
134     elif counter == maxIteration:
135         #increase 'max_lin_acc' for safety reason
```

```
        max_lin_acc = [safety_coeff * i for i in max_lin_acc]
        max_root = math.sqrt(
            pow(max_lin_acc[0], 2) + pow(max_lin_acc[1], 2) + pow(max_lin_acc[2], 2))

        delta_root = math.sqrt(3 * (pow(delta, 2)))

        print("Calibration terminated")
        # if true, the 'maximum linear acceleration vector' is very small (which is good): lower delta accordingly
        if max_root > delta_root:
            delta = max_root

    else:
        #reduce noise on incoming linear acceleration data
        lin_acc_no_g = lin_acc_compensate(lin_acc_no_g, delta)

        if flagWriteData == 1:
            # store data into .csv files in order to analyse them offline
            storeDataInFiles(abs_file_path1, 'a', lin_acc_no_g)

            angles_in_deg = [(tempAngles[0]*180) / math.pi,
                             (tempAngles[1]*180) / math.pi, (tempAngles[2]*180) / math.pi]
            storeDataInFiles(abs_file_path2, 'a', angles_in_deg)
            storeDataInFiles(abs_file_path3, 'a', angular_velocity)

            index += 1  # update index

        # modify Imu message to be sent
        data.orientation.x = orientation[0]
        data.orientation.y = orientation[1]
        data.orientation.z = orientation[2]

        data.angular_velocity.x = angular_velocity[0]
        data.angular_velocity.y = angular_velocity[1]
        data.angular_velocity.z = angular_velocity[2]
        data.linear_acceleration.x = lin_acc_no_g[0]
        data.linear_acceleration.y = lin_acc_no_g[1]
        data.linear_acceleration.z = lin_acc_no_g[2]

        # publish message
        try:
            talker(data)
        except rospy.ROSInterruptException:
            pass

    counter +=1

```

6.2.2.2 dataFileInitializer()

```
def clipping.dataFileInitializer ( )
```

Function used initialize the files in which sensor data will be stored; it will generate three files in the 'output' folder.

Definition at line 50 of file clipping.py.

```
50 def dataFileInitializer():
51     """!
52     Function used initialize the files in which sensor data will be stored; it will generate three files in the 'output' folder
53     """
54     # initialize files to store data
55     with open(abs_file_path1, 'w') as file:
56         writer = csv.writer(file)
57         writer.writerow(["", "X", "Y", "Z"])
58
59     with open(abs_file_path2, 'w') as file:
60         writer = csv.writer(file)
61         writer.writerow(["", "X", "Y", "Z"])
62
63     with open(abs_file_path3, 'w') as file:
64         writer = csv.writer(file)
65         writer.writerow(["", "X", "Y", "Z"])
66
```

6.2.2.3 lin_acc_compensate()

```
def clipping.lin_acc_compensate (
              lin_acc_no_g,
              threshold )
```

Function used to filter unwanted minimal incoming data fluctuations, due to noise as well as human operator shake.

Parameters

lin_acc_no←_g	linear acceleration along X, Y, Z axes
threshold	a threshold computed in the calibration phase to remove unwanted vibrations ----------------------------—

Returns

returns a filtered version (if necessary) of the input linear acceleration without gravity

Definition at line 67 of file clipping.py.

```
67 def lin_acc_compensate(lin_acc_no_g, threshold):
68     """!
69     Function used to filter unwanted minimal incoming data fluctuations,
70     due to noise as well as human operator shake
71     @param lin_acc_no_g linear acceleration along X, Y, Z axes
72     @param threshold a threshold computed in the calibration phase to remove unwanted vibrations
       -----------------------------
73     @returns returns a filtered version (if necessary) of the input linear acceleration without gravity
74     """
75     # initialization
76     res = [0, 0, 0]
77 
78     result = math.sqrt(
79         pow(lin_acc_no_g[0], 2) + pow(lin_acc_no_g[1], 2) + pow(lin_acc_no_g[2], 2))
80 
81     if result >= threshold:
82         res = lin_acc_no_g
83 
84     return res
85 
```

6.2.2.4 listener()

```
def clipping.listener ( )
```

The listener is used to instantiate the homonymous node and to subscribe to the android/imu_corected topic, from which it receives incoming data of smarthpone accelerometers.

Definition at line 182 of file clipping.py.

```
182 def listener():
183     """!
184     The listener is used to instantiate the homonymous node and to subscribe to the android/imu_corected
       topic,
185     from which it receives incoming data of smarthpone accelerometers
186     """
187 
188     # In ROS, nodes are uniquely named. If two nodes with the same
189     # name are launched, the previous one is kicked off. The
190     # anonymous = True flag means that rospy will choose a unique
191     # name for our 'listener' node so that multiple listeners can
192     # run simultaneously.
193     rospy.init_node('clipping', anonymous=True)
194 
195     # This declares that your node subscribes to the android/imu topic,
```

```
196     # which is of type sensor_msgs.msg.Imu. When new data is received,
197     # callback is invoked with that data as argument.
198     rospy.Subscriber("/android/imu_corrected", Imu, callback)
199
200     # spin() simply keeps python from exiting until this node is stopped
201     rospy.spin()
202
203
```

6.2.2.5 storeDataInFiles()

```
def clipping.storeDataInFiles (
        fileName,
        modality,
        data )
```

Function which stores incoming data into .csv files @params fileName name of the generated file @params modality parameter requested by the open() function: 'a' stands for 'open for appending at the end of the file without truncating it' @params data data to be inserted in the .csv file (orientation/linear acceleration/angular velocity)

Definition at line 86 of file clipping.py.

```
86 def storeDataInFiles(fileName, modality, data):
87     """!
88     Function which stores incoming data into .csv files
89     @params fileName name of the generated file
90     @params modality parameter requested by the open() function: 'a' stands for 'open for appending at
       the end of the file without truncating it'
91     @params data data to be inserted in the .csv file (orientation/linear acceleration/angular velocity)
92     """
93     with open(fileName, modality) as file:
94         writer = csv.writer(file)
95         writer.writerow([index, data[0], data[1], data[2]])
96
```

6.2.2.6 talker()

```
def clipping.talker (
        msg )
```

The talker is used to publish the refined data on the so called 'smartphone' topic.

Definition at line 204 of file clipping.py.

```
204 def talker(msg):
205     """!
206     The talker is used to publish the refined data on the so called 'smartphone' topic
207     """
208     pub = rospy.Publisher('smartphone', Imu, queue_size=10)
209     pub.publish(msg)
210
211
```

6.2.3 Variable Documentation

6.2.3.1 abs_file_gravity

```
clipping.abs_file_gravity = os.path.join(script_dir, rel_path_gravity)
```

Definition at line 32 of file clipping.py.

6.2.3.2 abs_file_path1

```
clipping.abs_file_path1 = os.path.join(script_dir, rel_path1)
```

Definition at line 24 of file clipping.py.

6.2.3.3 abs_file_path2

```
clipping.abs_file_path2 = os.path.join(script_dir, rel_path2)
```

Definition at line 26 of file clipping.py.

6.2.3.4 abs_file_path3

```
clipping.abs_file_path3 = os.path.join(script_dir, rel_path3)
```

Definition at line 28 of file clipping.py.

6.2.3.5 angular_velocity

```
list clipping.angular_velocity = [0, 0, 0]
```

Definition at line 42 of file clipping.py.

6.2.3.6 counter

```
int clipping.counter = 0
```

Definition at line 47 of file clipping.py.

6.2.3.7 delta

```
float clipping.delta = 0.05
```

Definition at line 40 of file clipping.py.

6.2.3.8 flagWriteData

```
int clipping.flagWriteData = 1
```

Definition at line 35 of file clipping.py.

6.2.3.9 g

```
list clipping.g = [0 ,0 ,0]
```

Definition at line 44 of file clipping.py.

6.2.3.10 index

```
int clipping.index = 1
```

Definition at line 37 of file clipping.py.

6.2.3.11 lin_acc_no_g

```
list clipping.lin_acc_no_g = [0, 0, 0]
```

Definition at line 41 of file clipping.py.

6.2.3.12 max_lin_acc

```
list clipping.max_lin_acc = [0, 0, 0]
```

Definition at line 45 of file clipping.py.

6.2.3.13 maxIteration

```
int clipping.maxIteration = 50
```

Definition at line 48 of file clipping.py.

6.2.3.14 orientation

```
list clipping.orientation = [0, 0, 0, 0]
```

Definition at line 43 of file clipping.py.

6.2.3.15 rel_path1

```
string clipping.rel_path1 = "../output/lin_acc.csv"
```

Definition at line 23 of file clipping.py.

6.2.3.16 rel_path2

```
string clipping.rel_path2 = "../output/orientation.csv"
```

Definition at line 25 of file clipping.py.

6.2.3.17 rel_path3

```
string clipping.rel_path3 = "../output/angVel.csv"
```

Definition at line 27 of file clipping.py.

6.2.3.18 rel_path_gravity

```
string clipping.rel_path_gravity = "../calibration/gravity.csv"
```

Definition at line 31 of file clipping.py.

6.2.3.19 safety_coeff

```
float clipping.safety_coeff = 1.1
```

Definition at line 46 of file clipping.py.

6.2.3.20 script_dir

```
clipping.script_dir = os.path.dirname(__file__)
```

Definition at line 22 of file clipping.py.

6.2.3.21 temp

```
clipping.temp = pd.read_csv(abs_file_gravity, names=['X', 'Y', 'Z'], header=0, decimal=',')
```

Definition at line 218 of file clipping.py.

6.3 computeGravity Namespace Reference

Functions

- def dataFileInitializer ()

  Function used initialize the files in which data will be stored; it will be stored in the calibration folder.
- def callback (data)

  This is the callback function: it is invoked every time there is incoming data and has the duty of calling all the previously mentioned functions as well as eulerAnglesToRotationMatrix().
- def listener ()

  The listener is used to instantiate the homonymous node and to subscribe to the android/imu_corrected topic, from which it receives incoming data of smarthpone accelerometers.

Variables

- script_dir = os.path.dirname(__file__)
- string rel_path_gravity = "../calibration/gravity.csv"
- abs_file_path_gravity = os.path.join(script_dir, rel_path_gravity)
- int counter = 0
- float sum_x = 0.0
- float sum_y = 0.0
- float sum_z = 0.0
- float delta = 0.05
- float safety_coeff = 1.1
- list max_lin_acc = [0, 0, 0]
- list lin_acc_no_g = [0, 0, 0]
- list orientation = [0, 0, 0, 0]
- list g = [0, 0, 0]
- int maxIteration = 50

6.3.1 Detailed Description

```
Documentation for the compute gravity tool.
This code analyzes the first ’maxIteration’ samples and computes an estimate of the gravity vector.
```

6.3.2 Function Documentation

6.3.2.1 callback()

```
def computeGravity.callback (
              data )
```

This is the callback function: it is invoked every time there is incoming data and has the duty of calling all the previously mentioned functions as well as eulerAnglesToRotationMatrix().

After these calls, it finally invokes the talker(). In order to achieve all of this, it translates incoming quaternions into euler angles beforehand

Parameters

data	data incoming from the imu sensor

Definition at line 48 of file computeGravity.py.

```
48 def callback(data):
49     """!
50     This is the callback function: it is invoked every time there is incoming data and
51     has the duty of calling all the previously mentioned functions as well as
       eulerAnglesToRotationMatrix().
52     After these calls, it finally invokes the talker(). In order to achieve all of this,
53     it translates incoming quaternions into euler angles beforehand
54     @param data data incoming from the imu sensor
55     """
56     global counter, sum_x, sum_y, sum_z, delta, lin_acc_no_g, orientation, max_lin_acc, g
57 
58     # get data
59     orientation = [data.orientation.x, data.orientation.y,
60                    data.orientation.z, data.orientation.w]
61 
62     # transform quaternion to euler angles
63     tempAngles = tf.transformations.euler_from_quaternion(orientation, "sxyz")
64 
65     #compute the rotation matrix
66     rot_matrix = eulerAnglesToRotationMatrix(tempAngles)
67 
68     #define the linear acceleration vector
69     linear_acceleration = [data.linear_acceleration.x,
70                            data.linear_acceleration.y, data.linear_acceleration.z]
71 
72     # calibration phase for gravity removal
73     if counter < maxIteration:
74 
75         g_frame_i = np.dot(rot_matrix.transpose(), linear_acceleration)
76 
77         sum_x += g_frame_i[0]
78         sum_y += g_frame_i[1]
79         sum_z += g_frame_i[2]
80 
81     elif counter == maxIteration:
82         #estimate gravity vector
83         g = [sum_x/maxIteration, sum_y/maxIteration, sum_z/maxIteration]
84 
85         #store gravity vector in .csv file
86         with open(abs_file_path_gravity, 'a') as file:
87             writer = csv.writer(file)
88             writer.writerow([1, g[0], g[1], g[2]])
```

```
            print("File ok")
            rospy.signal_shutdown("")

    counter += 1  #update


```

6.3.2.2 dataFileInitializer()

`def computeGravity.dataFileInitializer ( )`

Function used initialize the files in which data will be stored; it will be stored in the calibration folder.

Definition at line 39 of file computeGravity.py.

```
39 def dataFileInitializer():
40     """!
41     Function used initialize the files in which data will be stored; it will be stored in the calibration
       folder
42     """
43     # initialize files to store data
44     with open(abs_file_path_gravity, 'w') as file:
45         writer = csv.writer(file)
46         writer.writerow(["", "X", "Y", "Z"])
47
```

6.3.2.3 listener()

`def computeGravity.listener ( )`

The listener is used to instantiate the homonymous node and to subscribe to the android/imu_corrected topic, from which it receives incoming data of smarthpone accelerometers.

Definition at line 95 of file computeGravity.py.

```
95 def listener():
96     """!
97     The listener is used to instantiate the homonymous node and to subscribe to the android/imu_corrected
       topic,
98     from which it receives incoming data of smarthpone accelerometers
99     """
100
101     # In ROS, nodes are uniquely named. If two nodes with the same
102     # name are launched, the previous one is kicked off. The
103     # anonymous = True flag means that rospy will choose a unique
104     # name for our 'listener' node so that multiple listeners can
105     # run simultaneously.
106     rospy.init_node('computeGravity', anonymous=True)
107
108     # This declares that your node subscribes to the android/imu topic,
109     # which is of type sensor_msgs.msg.Imu. When new data is received,
110     # callback is invoked with that data as argument.
111     rospy.Subscriber("/android/imu_corrected", Imu, callback)
112
113     rospy.spin() #simply keeps python from exiting until this node is stopped
114
```

6.3.3 Variable Documentation

6.3.3.1 abs_file_path_gravity

```
computeGravity.abs_file_path_gravity = os.path.join(script_dir, rel_path_gravity)
```

Definition at line 24 of file computeGravity.py.

6.3.3.2 counter

```
int computeGravity.counter = 0
```

Definition at line 26 of file computeGravity.py.

6.3.3.3 delta

```
float computeGravity.delta = 0.05
```

Definition at line 31 of file computeGravity.py.

6.3.3.4 g

```
list computeGravity.g = [0, 0, 0]
```

Definition at line 36 of file computeGravity.py.

6.3.3.5 lin_acc_no_g

```
list computeGravity.lin_acc_no_g = [0, 0, 0]
```

Definition at line 34 of file computeGravity.py.

6.3.3.6 max_lin_acc

```
list computeGravity.max_lin_acc = [0, 0, 0]
```

Definition at line 33 of file computeGravity.py.

6.3.3.7 maxIteration

```
int computeGravity.maxIteration = 50
```

Definition at line 37 of file computeGravity.py.

6.3.3.8 orientation

```
list computeGravity.orientation = [0, 0, 0, 0]
```

Definition at line 35 of file computeGravity.py.

6.3.3.9 rel_path_gravity

```
string computeGravity.rel_path_gravity = "../calibration/gravity.csv"
```

Definition at line 23 of file computeGravity.py.

6.3.3.10 safety_coeff

```
float computeGravity.safety_coeff = 1.1
```

Definition at line 32 of file computeGravity.py.

6.3.3.11 script_dir

```
computeGravity.script_dir = os.path.dirname(__file__)
```

Definition at line 22 of file computeGravity.py.

6.3.3.12 sum_x

```
float computeGravity.sum_x = 0.0
```

Definition at line 27 of file computeGravity.py.

6.3.3.13 sum_y

```
float computeGravity.sum_y = 0.0
```

Definition at line 28 of file computeGravity.py.

6.3.3.14 sum_z

```
float computeGravity.sum_z = 0.0
```

Definition at line 29 of file computeGravity.py.

6.4 Errors Namespace Reference

Functions

- def ang_mis (Rg, Re)
  Computes the angular misalignment between goal frame and e.e.
- def errors (data)
  Computes the errors needed by the inverse kinematics algorithms.
- def errors_node ()

Variables

- pub = rospy.Publisher('errors', Float64MultiArray, queue_size=10)
- Rg = np.zeros((3,3))
- Re = np.zeros((3,3))
- xg = np.zeros((3,1))
- xe = np.zeros((3,1))
- vg = np.zeros((3,1))
- ve = np.zeros((3,1))

6.4.1 Function Documentation

6.4.1.1 ang_mis()

```
def Errors.ang_mis (
              Rg,
              Re )
```

Computes the angular misalignment between goal frame and e.e.

frame.

Parameters

Rg	goal orientation matrix.
Re	end effector orientation matrix.

Returns

rho: misalignment vector.

Definition at line 22 of file Errors.py.

```
22 def ang_mis(Rg, Re):
23     """!
24     Computes the angular misalignment between goal frame and e.e. frame.
25     @param Rg: goal orientation matrix.
26     @param Re: end effector orientation matrix.
27     @return rho: misalignment vector.
28     """
29
30     i = np.transpose(np.array([[1, 0, 0]]))
31     j = np.transpose(np.array([[0, 1, 0]]))
32     k = np.transpose(np.array([[0, 0, 1]]))
33
34     skew_i = np.array([[0, 0, 0],
35                        [0, 0, -1],
36                        [0, 1, 0]])
37
38     skew_j = np.array([[0, 0, 1],
39                        [0, 0, 0],
40                        [-1, 0, 0]])
41
42     skew_k = np.array([[0, -1, 0],
43                        [1, 0, 0],
44                        [0, 0, 0]])
45
46     # this term equals 1 + 2cos(theta), with theta the misalignement angle
47     summ = np.dot(np.transpose(np.dot(Re, i)), np.dot(Rg, i)) + np.dot(np.transpose(np.dot(Re, j)),
       np.dot(Rg, j)) + np.dot(np.transpose(np.dot(Re, k)), np.dot(Rg, k))
48
49     Re_ = np.transpose(Re)
50
51     # this term equals 2*v*sin(theta), with v the angular misalignement vector
52     prod = np.dot(np.dot(Re, np.dot(skew_i, Re_)), np.dot(Rg, i)) + np.dot(np.dot(Re, np.dot(skew_j,
       Re_)), np.dot(Rg, j)) + np.dot(np.dot(Re, np.dot(skew_k, Re_)), np.dot(Rg, k))
53
54     delta = (summ - 1)/2 # = cos(theta)
55
56     if np.absolute(delta) < 1:
57         theta = np.arccos(delta)
58         v = prod/(2*np.sin(theta))
59         v = v/(np.linalg.norm(v))
60         rho = theta*v
61     elif delta == 1:
62         # theta multiple of 2*k*pi
63         theta = 0
64         rho = np.array([[0, 0, 0]]).transpose()
65     else:
66         theta = np.pi
67         summ1 = np.dot(Re, i) + np.dot(Re, j) + np.dot(Re, k) + np.dot(Rg, i) + np.dot(Rg, j) +
       np.dot(Rg, k)
68         v0 = summ1/(np.linalg.norm(summ1))
69         rho = theta*v0
70
71     return rho
72
```

6.4.1.2 errors()

```
def Errors.errors (
                data )
```

Computes the errors needed by the inverse kinematics algorithms.

Specifically it calculates the misaligned vector between the goal frame and e.e. frame, the error due to the position of the two frames and also the error due to velocity of the two frames.

Parameters

data	vector coming from forward kinematics node. Constains rotation matrices, vector positions and velocities.

Definition at line 73 of file Errors.py.

```
73 def errors(data):
74     """!
75     Computes the errors needed by the inverse kinematics algorithms. Specifically
76     it calculates the misaligned vector between the goal frame and e.e. frame, the
77     error due to the position of the two frames and also the error due to velocity
78     of the two frames.
79     @param data: vector coming from forward kinematics node. Constains rotation matrices,
80     vector positions and velocities.
81     """
82
83     Data = data.data
84     global Rg, Re, xg, xe, vg, ve
85
86     # Moves along Data vector.
87     k = 0
88
89     for i in range(3):
90        for j in range(3):
91           Rg[i][j] = Data[k+j]
92        k = k + 3
93
94
95     for i in range(3):
96        for j in range(3):
97            Re[i][j] = Data[k+j]
98        k = k + 3
99
100     for i in range(3):
101        xg[i][0] = Data[k+i]
102
103     k = k + 3
104
105     for i in range(3):
106        xe[i][0] = Data[k+i]
107
108     k = k + 3
109
110     for i in range(3):
111        vg[i][0] = Data[k+i]
112
113     k = k + 3
114
115     for i in range(3):
116        ve[i][0] = Data[k+i]
117
118     # angular misalignment
119     rho = ang_mis(Rg, Re)
120
121     # position error
122     eta = xg - xe
123
124     # velocity error
125     ni = vg - ve
126
127
130
131     # Send rho, eta, ni
132     err = np.array([rho[0], rho[1], rho[2], eta[0], eta[1], eta[2], ni[0], ni[1], ni[2]],
        dtype=np.float_)
133     errors = util.init_float64_multiarray(9, 1)
134     errors.data = err
135     pub.publish(errors)
136
137
```

6.4.1.3 errors_node()

```
def Errors.errors_node ( )
```

Definition at line 138 of file Errors.py.

```
def errors_node():



    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('errors_node', anonymous=True)

    # errors is the callback functions
    rospy.Subscriber("Data_for_errors", Float64MultiArray, errors)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()
```

6.4.2 Variable Documentation

6.4.2.1 pub

```
Errors.pub = rospy.Publisher('errors', Float64MultiArray, queue_size=10)
```

Definition at line 9 of file Errors.py.

6.4.2.2 Re

```
Errors.Re = np.zeros((3,3))
```

Definition at line 13 of file Errors.py.

6.4.2.3 Rg

```
Errors.Rg = np.zeros((3,3))
```

Definition at line 12 of file Errors.py.

6.4.2.4 ve

```
Errors.ve = np.zeros((3,1))
```

Definition at line 20 of file Errors.py.

6.4.2.5 vg

```
Errors.vg = np.zeros((3,1))
```

Definition at line 19 of file Errors.py.

6.4.2.6 xe

```
Errors.xe = np.zeros((3,1))
```

Definition at line 17 of file Errors.py.

6.4.2.7 xg

```
Errors.xg = np.zeros((3,1))
```

Definition at line 16 of file Errors.py.

6.5 Forward_Kine2 Namespace Reference

Functions

- def main_callback ()

  Publish to error node and inverse kinematics nodes the data, in particular the jacobian matrix, the tracking vectors and the data needed to compute the errors.
- def baxter_callback (data)

  Computes the configuration of baxter's arm whenever the data are available, then extracts the rotation matrix from 0 to e.e and the position of the e.e.
- def dot_callback (data)

  Reads the q_dots provided by the weighter.
- def smart_callback (data)

  Computes the linear acceleration, angular velocity projected on 0 given the data.
- def calib_callback (data)

  Receives the orientation matrices from the calibration node.
- def simulate_callback (data)

  Handles changes in the simulation.
- def FK ()

Variables

- pub_err = rospy.Publisher('Data_for_errors', Float64MultiArray, queue_size=10)
- pub_track = rospy.Publisher('tracking', Float64MultiArray, queue_size=10)
- pub_jac = rospy.Publisher('jacobian', Float64MultiArray, queue_size=10)
- p = np.pi
- int n_joints = 7
- float L0 = 0.27035
- float L1 = 0.06900
- float L2 = 0.36435
- float L3 = 0.06900
- float L4 = 0.37429
- float L5 = 0.01000
- float L6 = 0.36830
- DH
- T_dh = t.DH_to_T(DH)
- Jkmin1 = np.zeros((6, 7))
- info = np.array([1, 1, 1, 1, 1, 1, 1])
- int ini_bax = 0
- int ini_dot = 0
- int ini_smart = 0
- int key_bax = 0
- int key_smart = 0
- int key_dot = 0
- int key = -1
- int calib_ok = 0
- int sequence = 0
- int steps = 10
- int index = 1
- float dt = 0.01
- x_0e_kmin1 = np.zeros((3, 1))
- x_0e_k = x_0e_kmin1
- x_0e_kmin1B = np.zeros((3, 1))
- v_0e_kmin1 = np.zeros((3, 1))
- v_0e_k = v_0e_kmin1
- v_0e_kmin1B = np.zeros((3, 1))
- q = np.zeros(7)
- q_dot = np.zeros((7, 1))
- R0e_ini = np.zeros((3, 3))
- R0global = np.zeros((3, 3))
- Re_imu = np.zeros((3, 3))
- Rimu_global_k = np.zeros((3, 3))
- R0e_kmin1 = np.zeros((3, 3))
- R0e_k = np.zeros((3, 3))
- omega_imu_global = np.zeros((3, 1))
- a_imu_global = np.zeros((3, 1))
- omega_0e = np.zeros((3, 1))
- a_0e = np.zeros((3, 1))

6.5.1 Function Documentation

6.5.1.1 baxter_callback()

```
def Forward_Kine2.baxter_callback (
                data )
```

Computes the configuration of baxter's arm whenever the data are available, then extracts the rotation matrix from 0 to e.e and the position of the e.e.

with respect to 0. It also computes the jacobian matrix. In case all the other callbacks have been called then it computes the velocity of the e.e. with respect to 0 and the end it calls the main_callback.

Parameters

data	coming from baxter node which provides a JointState message.

Definition at line 156 of file Forward_Kine2.py.

```
156 def baxter_callback(data):
157     """!
158     Computes the configuration of baxter's arm whenever the data are available, then extracts the rotation
159     matrix from 0 to e.e and the position of the e.e. with respect to 0. It also computes the jacobian matrix.
160     In case all the other callbacks have been called then it computes the velocity of the
161     e.e. with respect to 0 and the end it calls the main_callback.
162     @param data: coming from baxter node which provides a JointState message.
163     """
164 
165     global ini_bax, q, R0e_ini, R0e_kmin1, Jkmin1, x_0e_kmin1B, x_0e_kmin1, v_0e_kmin1B, key_bax, key_dot, key_smart
166 
167     if (key == 1 or ini_bax == 0):
168 
169 
172         if ini_bax != 0:
173             # configuration at time kmin1
174             q = np.array(data.position)
175 
176         # relative T's with the configuration passed.
177         T_rel_kmin1 = t.transformations(T_dh, q, info)
178         # absolute T's
179         T_abs_kmin1 = t.abs_trans(T_rel_kmin1)
180 
181         # geometric vectors needed to compute jacobian.
182         geom = j.geometric_vectors(T_abs_kmin1)
183 
184         # jacobian computation
185         Jkmin1 = j.jacob(geom[0], geom[1], n_joints, info)
186 
187         # Transformation matrix from 0 to end effector at time k
188         T0e_kmin1 = T_abs_kmin1[7]
189 
190         # end effector position of baxter at time k
191         for i in range(3):
192             x_0e_kmin1B[i] = T0e_kmin1[i][3]
193 
194         # end effector orientation of baxter at time k. At time 0 i have the
195         # orientation of zero with respect of inertial frame also.
196         for i in range(3):
197             for k in range(3):
198                 R0e_kmin1[i][k] = T0e_kmin1[i][k]
199 
200         if ini_bax == 0:
201             # Initial conditions.
202             R0e_ini = R0e_kmin1
203             x_0e_kmin1 = x_0e_kmin1B
204             ini_bax = ini_bax + 1
205 
206         key_bax = key_bax + 1
207 
208         if (key_bax >= 1 and key_dot >= 1):
209             x_dot = np.dot(Jkmin1, q_dot)
210             for i in range(3):
211                 v_0e_kmin1B[i][0] = x_dot[i][0]
212 
213             if(key_smart >= 1):
214                 key_bax = 0
215                 key_dot = 0
```

```
                key_smart = 0

                main_callback()


```

6.5.1.2 calib_callback()

```
def Forward_Kine2.calib_callback (
                data )
```

Receives the orientation matrices from the calibration node.

Parameters

data	vector containing 2 matrices, R0global and Reimu

Definition at line 354 of file Forward_Kine2.py.

```
354 def calib_callback(data):
355     """!
356     Receives the orientation matrices from the calibration node.
357     @param data: vector containing 2 matrices, R0global and Reimu
358     """
359
360     global calib_ok, R0global
361
362     R = np.array(data.data)
363
364     R0global = R.reshape(3, 3)
365
366     calib_ok = 1
367
368
```

6.5.1.3 dot_callback()

```
def Forward_Kine2.dot_callback (
                data )
```

Reads the q_dots provided by the weighter.

In case all the other callbacks have been called then it computes the velocity of the e.e. with respect to 0 and the end it calls the main_callback.

Parameters

data	coming from weighter node which provides a JointState message.

Definition at line 221 of file Forward_Kine2.py.

```
221 def dot_callback(data):
222     """!
223     Reads the q_dots provided by the weighter. In case all the other callbacks have been called then it
       computes the velocity of the
224     e.e. with respect to 0 and the end it calls the main_callback.
225     @param data: coming from weighter node which provides a JointState message.
226     """
```

```
    global ini_dot, q_dot, Jkmin1, v_0e_kmin1B, key_bax, key_dot, key_smart

    if (key == 1 or ini_dot == 0):


        if ini_dot != 0:
            q_dot = np.transpose(np.array([data.velocity]))

        key_dot = key_dot + 1

        if ini_dot == 0:
            ini_dot = ini_dot + 1

        if (key_bax >= 1 and key_dot >= 1):
            x_dot = np.dot(Jkmin1, q_dot)
            for i in range(3):
                v_0e_kmin1B[i][0] = x_dot[i][0]

            if(key_smart >= 1):
                key_bax = 0
                key_dot = 0
                key_smart = 0

                main_callback()
```

6.5.1.4 FK()

`def Forward_Kine2.FK ( )`

Definition at line 408 of file Forward_Kine2.py.

```
408 def FK():
409
410     # In ROS, nodes are uniquely named. If two nodes with the same
411     # name are launched, the previous one is kicked off. The
412     # anonymous=True flag means that rospy will choose a unique
413     # name for our 'listener' node so that multiple listeners can
414     # run simultaneously.
415     rospy.init_node('FK', anonymous=True)
416
417     # Receive data from smartphone, baxter, weighter and coppelia.
418     rospy.Subscriber("smartphone", Imu, smart_callback)
419     rospy.Subscriber("logtopic", JointState, baxter_callback)
420     rospy.Subscriber("cmdtopic", JointState, dot_callback)
421     rospy.Subscriber("handleSimulation", Int8, simulate_callback)
422     rospy.Subscriber("rot_matrices", Float64MultiArray, calib_callback)
423
424     # spin() simply keeps python from exiting until this node is stopped
425     rospy.spin()
426
427
```

6.5.1.5 main_callback()

`def Forward_Kine2.main_callback ( )`

Publish to error node and inverse kinematics nodes the data, in particular the jacobian matrix, the tracking vectors and the data needed to compute the errors.

Definition at line 116 of file Forward_Kine2.py.

```
116 def main_callback():
117     """!
118     Publish to error node and inverse kinematics nodes the data, in particular the
119     jacobian matrix, the tracking vectors and the data needed to compute the errors.
120     """
121
122
```

```
    global x_0e_kmin1B, x_0e_k, v_0e_kmin1B, v_0e_k, a_0e, omega_0e, R0e_kmin1, R0e_k, Jkmin1


    J = util.init_float64_multiarray(6*7, 1)
    J.data = Jkmin1.reshape(6*7, 1)
    pub_jac.publish(J)



    # Send R0e_k, R0e_kmin1; x_0e_k, x_0e_kmin1B; v_0e_k, v_0e_kmin1B;
    Rg_Re_xg_xe_vg_ve = np.array([R0e_k[0][0], R0e_k[0][1], R0e_k[0][2], R0e_k[1][0], R0e_k[1][1],
   R0e_k[1][2], R0e_k[2][0], R0e_k[2][1], R0e_k[2][2], R0e_kmin1[0][0], R0e_kmin1[0][1],
   R0e_kmin1[0][2], R0e_kmin1[1][0], R0e_kmin1[1][1], R0e_kmin1[1][2], R0e_kmin1[2]
                                  [0], R0e_kmin1[2][1], R0e_kmin1[2][2], x_0e_k[0][0], x_0e_k[1][0],
   x_0e_k[2][0], x_0e_kmin1B[0][0], x_0e_kmin1B[1][0], x_0e_kmin1B[2][0], v_0e_k[0][0], v_0e_k[1][0],
   v_0e_k[2][0], v_0e_kmin1B[0][0], v_0e_kmin1B[1][0], v_0e_kmin1B[2][0]], dtype=np.float_)
    R_x_v = util.init_float64_multiarray(30, 1)
    R_x_v.data = Rg_Re_xg_xe_vg_ve
    pub_err.publish(R_x_v)



    # send v_0e_k, omega_0e, a_0e
    vg_omega_a = np.array([v_0e_k[0][0], v_0e_k[1][0], v_0e_k[2][0], omega_0e[0][0],
                           omega_0e[1][0], omega_0e[2][0], a_0e[0][0], a_0e[1][0], a_0e[2][0]],
   dtype=np.float_)
    v_w_a = util.init_float64_multiarray(9, 1)
    v_w_a.data = vg_omega_a
    pub_track.publish(v_w_a)
```

6.5.1.6 simulate_callback()

```
def Forward_Kine2.simulate_callback (
                data )
```

Handles changes in the simulation.

If data = 0, then the initial conditions must be resetted. If data = 1, then the algorithm moves on. If data = 2, then the algorithm pauses.

Parameters

data	coming from coppelia_sim.

Definition at line 369 of file Forward_Kine2.py.

```
369 def simulate_callback(data):
370     """!
371     Handles changes in the simulation. If data = 0, then the initial conditions must be resetted.
372     If data = 1, then the algorithm moves on. If data = 2, then the algorithm pauses.
373     @param data: coming from coppelia_sim.
374     """
375 
376     global ini_bax, ini_dot, ini_smart, q, q_dot, v_0e_kmin1, key_bax, key_smart, key_dot, key
377 
378     key = data.data
379     rospy.logerr("------------------------------------------k: ")
380     rospy.logerr(key)
381     if key == 0:
382         rospy.logerr("Resetting")
383         # reset of the initial conditions.
384 
385         # Need this variable to store initial rotation matrix, because it's equal to
386         # rotation matrix from 0 to inertial frame.
387         ini_bax = 0
388         # needed to differentiate from initial condition to computed qdots.
389         ini_dot = 0
390         ini_smart = 0
391 
392         # Sync variables
```

```
        key_bax = 0
        key_smart = 0
        key_dot = 0

        # Initial velocity of end effector w.r.t. zero
        v_0e_kmin1 = np.zeros((3, 1))  # starting velocity

        # Define the q's and q dots
        q = np.zeros(7)
        q_dot = np.zeros((7, 1))

        baxter_callback(0)  # to set the initial conditions.
        dot_callback(0)  # to set the initial conditions.


```

6.5.1.7 smart_callback()

```
def Forward_Kine2.smart_callback (
                data )
```

Computes the linear acceleration, angular velocity projected on 0 given the data.

In case all the other callbacks have been called then it computes the velocity of the e.e. with respect to 0 and the end it calls the main_callback.

Parameters

data	coming from smartphone which provides a Imu() message.

Definition at line 257 of file Forward_Kine2.py.

```
257 def smart_callback(data):
258     """!
259     Computes the linear acceleration, angular velocity projected on 0 given the data. In case all the
    other callbacks have been called then it computes the velocity of the
260     e.e. with respect to 0 and the end it calls the main_callback.
261     @param data: coming from smartphone which provides a Imu() message.
262     """
263
264     if (key == 1 and calib_ok == 1):
265
266         global index, sequence, ini_smart, omega_imu_global, a_imu_global, Re_imu, Rimu_global_k, R0e_k,
    x_0e_kmin1, x_0e_k, v_0e_kmin1, v_0e_k, a_0e, omega_0e, key_bax, key_dot, key_smart
267
271
272         # Get orientation
273         orien = [data.orientation.x, data.orientation.y,
274                  data.orientation.z, data.orientation.w]
275
276         # transform quaternion to euler angles
277         Ttemp = tf.transformations.quaternion_matrix(
278             (orien[0], orien[1], orien[2], orien[3]))
279         Rimu_global_k = Ttemp[:3, :3]
280
281         Rglobal_imu_k = np.transpose(Rimu_global_k)
282
283         if ini_smart == 0:
284             Re_imu = np.dot(
285                 np.dot(R0e_ini.transpose(), R0global), Rglobal_imu_k)
286             rospy.logerr("You can start moving")
287             ini_smart = ini_smart + 1
288
289         # angular velocity of imu (end effector) w.r.t. inertial frame projected on imu frame
290         omega_imu_global[0][0] = data.angular_velocity.x
291         omega_imu_global[1][0] = data.angular_velocity.y
292         omega_imu_global[2][0] = data.angular_velocity.z
293
294         # linear acceleration of imu (end effector) w.r.t. inertial frame projected on imu frame
295         a_imu_global[0][0] = data.linear_acceleration.x
296         a_imu_global[1][0] = data.linear_acceleration.y
297         a_imu_global[2][0] = data.linear_acceleration.z
```

```
        # imu frame at time k is superimposed to e.e. frame at time k. Innertial and zero
        # are not moving and since the inertial is placed where the e.e. was at its initial conditions,
        # i can compute R0e_k
        R0imu_k = np.dot(R0global, Rglobal_imu_k)
        R0e_k = np.dot(R0imu_k, Re_imu.transpose())

        # Since inertial is not moving, the angular velocity and linear acceleration are the same
        # if calculated w.r.t. 0, however i need to project them in zero.
        omega_0e = np.dot(R0imu_k, omega_imu_global)
        a_0e = np.dot(R0imu_k, a_imu_global)



        # clipped accelerations should satisfy this condition.
        if (np.linalg.norm(a_0e) < 0.01):

            # counts how many clipped acc. are received in a row.
            sequence = sequence + 1

            if (sequence >= steps):
                # let the velocity approach zero.
                v_0e_k = v_0e_kmin1*np.exp(-index)

                index = index + 1

            else:
                # let the velocity be constant.
                v_0e_k = v_0e_kmin1
        else:
            # reset the variables that handle the velocity inside the clipping region.
            sequence = 0
            index = 1

            # Target velocity.
            v_0e_k = v_0e_kmin1 + a_0e*dt

        # Target position.
        x_0e_k = x_0e_kmin1 + v_0e_kmin1*dt + 0.5*a_0e*dt*dt

        key_smart = key_smart + 1

        if (key_bax >= 1 and key_dot >= 1 and key_smart >= 1):
            key_bax = 0
            key_dot = 0
            key_smart = 0
            main_callback()

        # Update this vectors to compute integration at next steps.

        x_0e_kmin1 = x_0e_k
        v_0e_kmin1 = v_0e_k
```

6.5.2 Variable Documentation

6.5.2.1 a_0e

`Forward_Kine2.a_0e = np.zeros((3, 1))`

Definition at line 113 of file Forward_Kine2.py.

6.5.2.2 a_imu_global

`Forward_Kine2.a_imu_global = np.zeros((3, 1))`

Definition at line 109 of file Forward_Kine2.py.

6.5.2.3 calib_ok

```
int Forward_Kine2.calib_ok = 0
```

Definition at line 63 of file Forward_Kine2.py.

6.5.2.4 DH

```
Forward_Kine2.DH
```

Initial value:

```
=  np.array([[0, 0, L0, 0],
             [-p/2, L1, 0, p/2],
             [p/2, 0, L2, 0],
             [-p/2, L3, 0, 0],
             [p/2, 0, L4, 0],
             [-p/2, L5, 0, 0],
             [p/2, 0, 0, 0],
             [0, 0, L6, 0]])
```

Definition at line 32 of file Forward_Kine2.py.

6.5.2.5 dt

```
float Forward_Kine2.dt = 0.01
```

Definition at line 73 of file Forward_Kine2.py.

6.5.2.6 index

```
int Forward_Kine2.index = 1
```

Definition at line 70 of file Forward_Kine2.py.

6.5.2.7 info

```
Forward_Kine2.info = np.array([1, 1, 1, 1, 1, 1, 1])
```

Definition at line 48 of file Forward_Kine2.py.

6.5.2.8 ini_bax

```
int Forward_Kine2.ini_bax = 0
```

Definition at line 51 of file Forward_Kine2.py.

6.5.2.9 ini_dot

```
int Forward_Kine2.ini_dot = 0
```

Definition at line 53 of file Forward_Kine2.py.

6.5.2.10 ini_smart

```
int Forward_Kine2.ini_smart = 0
```

Definition at line 54 of file Forward_Kine2.py.

6.5.2.11 Jkmin1

```
Forward_Kine2.Jkmin1 = np.zeros((6, 7))
```

Definition at line 45 of file Forward_Kine2.py.

6.5.2.12 key

```
int Forward_Kine2.key = -1
```

Definition at line 60 of file Forward_Kine2.py.

6.5.2.13 key_bax

```
int Forward_Kine2.key_bax = 0
```

Definition at line 57 of file Forward_Kine2.py.

6.5.2.14 key_dot

```
int Forward_Kine2.key_dot = 0
```

Definition at line 59 of file Forward_Kine2.py.

6.5.2.15 key_smart

```
int Forward_Kine2.key_smart = 0
```

Definition at line 58 of file Forward_Kine2.py.

6.5.2.16 L0

```
float Forward_Kine2.L0 = 0.27035
```

Definition at line 22 of file Forward_Kine2.py.

6.5.2.17 L1

```
float Forward_Kine2.L1 = 0.06900
```

Definition at line 23 of file Forward_Kine2.py.

6.5.2.18 L2

```
float Forward_Kine2.L2 = 0.36435
```

Definition at line 24 of file Forward_Kine2.py.

6.5.2.19 L3

```
float Forward_Kine2.L3 = 0.06900
```

Definition at line 25 of file Forward_Kine2.py.

6.5.2.20 L4

```
float Forward_Kine2.L4 = 0.37429
```

Definition at line 26 of file Forward_Kine2.py.

6.5.2.21 L5

```
float Forward_Kine2.L5 = 0.01000
```

Definition at line 27 of file Forward_Kine2.py.

6.5.2.22 L6

```
float Forward_Kine2.L6 = 0.36830
```

Definition at line 28 of file Forward_Kine2.py.

6.5.2.23 n_joints

```
int Forward_Kine2.n_joints = 7
```

Definition at line 19 of file Forward_Kine2.py.

6.5.2.24 omega_0e

```
Forward_Kine2.omega_0e = np.zeros((3, 1))
```

Definition at line 112 of file Forward_Kine2.py.

6.5.2.25 omega_imu_global

```
Forward_Kine2.omega_imu_global = np.zeros((3, 1))
```

Definition at line 108 of file Forward_Kine2.py.

6.5.2.26 p

```
Forward_Kine2.p = np.pi
```

Definition at line 18 of file Forward_Kine2.py.

6.5.2.27 pub_err

```
Forward_Kine2.pub_err = rospy.Publisher('Data_for_errors', Float64MultiArray, queue_size=10)
```

Definition at line 14 of file Forward_Kine2.py.

6.5.2.28 pub_jac

```
Forward_Kine2.pub_jac = rospy.Publisher('jacobian', Float64MultiArray, queue_size=10)
```

Definition at line 16 of file Forward_Kine2.py.

6.5.2.29 pub_track

```
Forward_Kine2.pub_track = rospy.Publisher('tracking', Float64MultiArray, queue_size=10)
```

Definition at line 15 of file Forward_Kine2.py.

6.5.2.30 q

```
Forward_Kine2.q = np.zeros(7)
```

Definition at line 86 of file Forward_Kine2.py.

6.5.2.31 q_dot

```
Forward_Kine2.q_dot = np.zeros((7, 1))
```

Definition at line 87 of file Forward_Kine2.py.

6.5.2.32 R0e_ini

```
Forward_Kine2.R0e_ini = np.zeros((3, 3))
```

Definition at line 92 of file Forward_Kine2.py.

6.5.2.33 R0e_k

```
Forward_Kine2.R0e_k = np.zeros((3, 3))
```

Definition at line 105 of file Forward_Kine2.py.

6.5.2.34 R0e_kmin1

```
Forward_Kine2.R0e_kmin1 = np.zeros((3, 3))
```

Definition at line 104 of file Forward_Kine2.py.

6.5.2.35 R0global

```
Forward_Kine2.R0global = np.zeros((3, 3))
```

Definition at line 95 of file Forward_Kine2.py.

6.5.2.36 Re_imu

```
Forward_Kine2.Re_imu = np.zeros((3, 3))
```

Definition at line 98 of file Forward_Kine2.py.

6.5.2.37 Rimu_global_k

```
Forward_Kine2.Rimu_global_k = np.zeros((3, 3))
```

Definition at line 101 of file Forward_Kine2.py.

6.5.2.38 sequence

```
int Forward_Kine2.sequence = 0
```

Definition at line 67 of file Forward_Kine2.py.

6.5.2.39 steps

```
int Forward_Kine2.steps = 10
```

Definition at line 69 of file Forward_Kine2.py.

6.5.2.40 T_dh

```
Forward_Kine2.T_dh = t.DH_to_T(DH)
```

Definition at line 42 of file Forward_Kine2.py.

6.5.2.41 v_0e_k

```
Forward_Kine2.v_0e_k = v_0e_kmin1
```

Definition at line 82 of file Forward_Kine2.py.

6.5.2.42 v_0e_kmin1

```
Forward_Kine2.v_0e_kmin1 = np.zeros((3, 1))
```

Definition at line 81 of file Forward_Kine2.py.

6.5.2.43 v_0e_kmin1B

```
Forward_Kine2.v_0e_kmin1B = np.zeros((3, 1))
```

Definition at line 83 of file Forward_Kine2.py.

6.5.2.44 x_0e_k

```
Forward_Kine2.x_0e_k = x_0e_kmin1
```

Definition at line 77 of file Forward_Kine2.py.

6.5.2.45 x_0e_kmin1

```
Forward_Kine2.x_0e_kmin1 = np.zeros((3, 1))
```

Definition at line 76 of file Forward_Kine2.py.

6.5.2.46 x_0e_kmin1B

```
Forward_Kine2.x_0e_kmin1B = np.zeros((3, 1))
```

Definition at line 78 of file Forward_Kine2.py.

6.6 Forward_Kine_halfcircle Namespace Reference

Functions

- def main_callback ()

  Publish to error node and inverse kinematics nodes the data, in particular the jacobian matrix, the tracking vectors and the data needed to compute the errors.
- def baxter_callback (data)

  Computes the configuration of baxter's arm whenever the data are available, then extracts the rotation matrix from 0 to e.e and the position of the e.e.
- def dot_callback (data)

  Reads the q_dots provided by the weighter.
- def smart_callback (data)

  Computes the linear acceleration, angular velocity projected on 0 given the data.
- def simulate_callback (data)

  Handles changes in the simulation.
- def FK ()

Variables

- pub_err = rospy.Publisher('Data_for_errors', Float64MultiArray, queue_size=10)
- pub_track = rospy.Publisher('tracking', Float64MultiArray, queue_size=10)
- pub_jac = rospy.Publisher('jacobian', Float64MultiArray, queue_size=10)
- p = np.pi
- int n_joints = 7
- float L0 = 0.27035
- float L1 = 0.06900
- float L2 = 0.36435
- float L3 = 0.06900
- float L4 = 0.37429
- float L5 = 0.01000
- float L6 = 0.36830
- int xeflag = 0
- DH
- T_dh = t.DH_to_T(DH)
- Jkmin1 = np.zeros((6,7))
- info = np.array([1, 1, 1, 1, 1, 1, 1])
- int ini_bax = 0
- int ini_dot = 0
- int ini_smart = 0
- int key_bax = 0
- int key_smart = 0
- int key_dot = 0
- int key = 1
- int flag_bax = 0
- int flag_dot = 0
- float dt = 0.01
- x_0e_kmin1 = np.zeros((3,1))
- x_0e_k = x_0e_kmin1
- x_0e_kmin1B = np.zeros((3,1))
- v_0e_kmin1 = np.zeros((3,1))
- v_0e_k = v_0e_kmin1
- v_0e_kmin1B = np.zeros((3,1))
- q = np.zeros(7)
- q_dot = np.zeros((7,1))
- R0inert = np.zeros((3,3))
- R0e_ini = np.zeros((3,3))
- Rimu_inert_k = np.zeros((3,3))
- R0e_kmin1 = np.zeros((3,3))
- R0e_k = np.zeros((3,3))
- omega_imu_inert = np.zeros((3,1))
- a_imu_inert = np.zeros((3,1))
- omega_0e = np.zeros((3,1))
- a_0e = np.zeros((3,1))

6.6.1 Function Documentation

6.6.1.1 baxter_callback()

```
def Forward_Kine_halfcircle.baxter_callback (
              data )
```

Computes the configuration of baxter's arm whenever the data are available, then extracts the rotation matrix from 0 to e.e and the position of the e.e.

with respect to 0. It also computes the jacobian matrix. In case all the other callbacks have been called then it computes the velocity of the e.e. with respect to 0 and the end it calls the main_callback.

Parameters

data	coming from baxter node which provides a JointState message.

Definition at line 140 of file Forward_Kine_halfcircle.py.

```
140 def baxter_callback(data):
141     """!
142     Computes the configuration of baxter's arm whenever the data are available, then extracts the rotation
143     matrix from 0 to e.e and the position of the e.e. with respect to 0. It also computes the jacobian matrix.
144     In case all the other callbacks have been called then it computes the velocity of the
145     e.e. with respect to 0 and the end it calls the main_callback.
146     @param data: coming from baxter node which provides a JointState message.
147     """
148 
149     global ini_bax, q, R0e_kmin1, R0e_ini, Jkmin1, x_0e_kmin1B, x_0e_kmin1, v_0e_kmin1B, key_bax, key_dot, key_smart, flag_bax, flag_dot
150 
151     if (key == 1 or ini_bax == 0):
152 
153 
156         if ini_bax != 0:
157             flag_bax = data.effort[0]
158             # configuration at time kmin1
159             q = np.array(data.position)
160 
161         if int(flag_bax) == int(flag_dot):
162 
163 
164             # relative T's with the configuration passed.
165             T_rel_kmin1 = t.transformations(T_dh, q, info)
166 
167             # absolute T's
168             T_abs_kmin1 = t.abs_trans(T_rel_kmin1)
169 
170             # geometric vectors needed to compute jacobian.
171             geom = j.geometric_vectors(T_abs_kmin1)
172 
173             # jacobian computation
174             Jkmin1 = j.jacob(geom[0], geom[1], n_joints, info)
175 
176             # Transformation matrix from 0 to end effector at time k
177             T0e_kmin1 = T_abs_kmin1[7]
178 
179             # end effector position of baxter at time k
180             for i in range(3):
181                 x_0e_kmin1B[i] = T0e_kmin1[i][3]
182 
183             # end effector orientation of baxter at time k. At time 0 i have the
184             # orientation of zero with respect of inertial frame also.
185             for i in range(3):
186                 for k in range(3):
187                     R0e_kmin1[i][k] = T0e_kmin1[i][k]
188 
189             if ini_bax == 0:
190                 #R0inert = R0e_kmin1 # Constant in time.
191                 R0e_ini = R0e_kmin1 # equal at starting configuration
192                 #x_0e_kmin1 =  x_0e_kmin1B # Initially they are equal
193                 x_0e_kmin1 = np.array([[ 1.1759, -4.3562e-06, 0.1913]]).transpose() # Initially they are equa
194                 ini_bax = ini_bax + 1
195 
196             key_bax = key_bax + 1
197 
198             if (key_bax >= 1 and key_dot >= 1):
```

```
199                 x_dot = np.dot(Jkmin1, q_dot)
200                 for i in range(3):
201                     v_0e_kmin1B[i][0] = x_dot[i][0]
202 
203                 if(key_smart >= 1):
204                     key_bax = 0
205                     key_dot = 0
206                     key_smart = 0
207 
208                     main_callback()
209 
210 
```

6.6.1.2 dot_callback()

```
def Forward_Kine_halfcircle.dot_callback (
                data )
```

Reads the q_dots provided by the weighter.

In case all the other callbacks have been called then it computes the velocity of the e.e. with respect to 0 and the end it calls the main_callback.

Parameters

data	coming from weighter node which provides a JointState message.

Definition at line 211 of file Forward_Kine_halfcircle.py.

```
211 def dot_callback(data):
212     """!
213     Reads the q_dots provided by the weighter. In case all the other callbacks have been called then it computes the velocity of the
214     e.e. with respect to 0 and the end it calls the main_callback.
215     @param data: coming from weighter node which provides a JointState message.
216     """
217 
218     global ini_dot, q_dot, Jkmin1, v_0e_kmin1B, key_bax, key_dot, key_smart, flag_bax, flag_dot
219 
220     if (key == 1 or ini_dot == 0):
221 
222 
226         if ini_dot != 0:
227             flag_dot = data.effort[0]
228             q_dot = np.transpose(np.array([data.velocity]))
229 
230         key_dot = key_dot + 1
231 
232         if ini_dot == 0:
233             ini_dot = ini_dot + 1
234 
235         if (key_bax >= 1 and key_dot >= 1):
236             x_dot = np.dot(Jkmin1, q_dot)
237             for i in range(3):
238                 v_0e_kmin1B[i][0] = x_dot[i][0]
239 
240             if(key_smart >= 1):
241                 key_bax = 0
242                 key_dot = 0
243                 key_smart = 0
244 
245                 main_callback()
246 
247 
```

6.6.1.3 FK()

```
def Forward_Kine_halfcircle.FK ( )
```

Definition at line 375 of file Forward_Kine_halfcircle.py.

```
375 def FK():
376
377     # In ROS, nodes are uniquely named. If two nodes with the same
378     # name are launched, the previous one is kicked off. The
379     # anonymous=True flag means that rospy will choose a unique
380     # name for our 'listener' node so that multiple listeners can
381     # run simultaneously.
382
383
384     rospy.init_node('FK', anonymous=True)
385
386     # Receive data from smartphone, baxter, weighter and coppelia.
387     rospy.Subscriber("smartphone", Imu, smart_callback)
388     rospy.Subscriber("logtopic", JointState, baxter_callback)
389     rospy.Subscriber("cmdtopic", JointState, dot_callback)
390     rospy.Subscriber("handleSimulation", Int8, simulate_callback)
391
392
393     # spin() simply keeps python from exiting until this node is stopped
394     rospy.spin()
395
396
```

6.6.1.4 main_callback()

```
def Forward_Kine_halfcircle.main_callback ( )
```

Publish to error node and inverse kinematics nodes the data, in particular the jacobian matrix, the tracking vectors and the data needed to compute the errors.

Definition at line 97 of file Forward_Kine_halfcircle.py.

```
97 def main_callback():
98     """!
99     Publish to error node and inverse kinematics nodes the data, in particular the
100     jacobian matrix, the tracking vectors and the data needed to compute the errors.
101     """
102
103
105     global x_0e_kmin1B, x_0e_k, v_0e_kmin1B, v_0e_k, a_0e, omega_0e, R0e_kmin1, R0e_k, Jkmin1
106
107
110     J = util.init_float64_multiarray(6*7, 1)
111     J.data = Jkmin1.reshape(6*7, 1)
112     pub_jac.publish(J)
113
114
117
118     # Send R0e_k, R0e_kmin1; x_0e_k, x_0e_kmin1B; v_0e_k, v_0e_kmin1B;
119     Rg_Re_xg_xe_vg_ve = np.array([R0e_k[0][0], R0e_k[0][1], R0e_k[0][2], R0e_k[1][0], R0e_k[1][1],
        R0e_k[1][2], R0e_k[2][0], R0e_k[2][1], R0e_k[2][2], R0e_kmin1[0][0], R0e_kmin1[0][1],
        R0e_kmin1[0][2], R0e_kmin1[1][0], R0e_kmin1[1][1], R0e_kmin1[1][2], R0e_kmin1[2][0], R0e_kmin1[2][1],

120     R0e_kmin1[2][2], x_0e_k[0][0], x_0e_k[1][0], x_0e_k[2][0], x_0e_kmin1B[0][0], x_0e_kmin1B[1][0],
        x_0e_kmin1B[2][0], v_0e_k[0][0], v_0e_k[1][0], v_0e_k[2][0], v_0e_kmin1B[0][0], v_0e_kmin1B[1][0],
        v_0e_kmin1B[2][0]], dtype=np.float_)
121
122     R_x_v = util.init_float64_multiarray(30, 1)
123     R_x_v.data = Rg_Re_xg_xe_vg_ve
124     pub_err.publish(R_x_v)
125
126
129
130     # send v_0e_k, omega_0e, a_0e
131     vg_omega_a = np.array([v_0e_k[0][0], v_0e_k[1][0], v_0e_k[2][0],
132     omega_0e[0][0], omega_0e[1][0], omega_0e[2][0],
133     a_0e[0][0], a_0e[1][0], a_0e[2][0]], dtype=np.float_)
134     v_w_a = util.init_float64_multiarray(9, 1)
135     v_w_a.data = vg_omega_a
136     pub_track.publish(v_w_a)
137
138     #print("Published")
139
```

6.6.1.5 simulate_callback()

```
def Forward_Kine_halfcircle.simulate_callback (
            data )
```

Handles changes in the simulation.

If data = 0, then the initial conditions must be resetted. If data = 1, then the algorithm moves on. If data = 2, then the algorithm pauses.

Parameters

data	coming from coppelia_sim.

Definition at line 334 of file Forward_Kine_halfcircle.py.

```
334 def simulate_callback(data):
335     """!
336     Handles changes in the simulation. If data = 0, then the initial conditions must be resetted.
337     If data = 1, then the algorithm moves on. If data = 2, then the algorithm pauses.
338     @param data: coming from coppelia_sim.
339     """
340
341     global ini_bax, ini_dot, ini_smart, q, q_dot, v_0e_kmin1, key_bax, key_smart, key_dot, key,
        flag_bax, flag_dot
342
343     key = data.data
344
345     if key == 0:
346         print("Resetting")
347         # reset of the initial conditions.
348
349         # Need this variable to store initial rotation matrix, because it's equal to
350         # rotation matrix from 0 to inertial frame.
351         ini_bax = 0
352         ini_dot = 0 # needed to differentiate from initial condition to computed qdots.
353         ini_smart = 0
354
355         # Sync variables.
356         key_bax = 0
357         key_smart = 0
358         key_dot = 0
359
360         # flag variables.
361         flag_bax = 0
362         flag_dot = 0
363
364         # Initial velocity of end effector w.r.t. zero
365         v_0e_kmin1 = np.zeros((3,1)) # starting velocity
366
367         # Define the q's and q dots
368         q = np.zeros(7)
369         q_dot = np.zeros((7,1))
370
371         baxter_callback(0) # to set the initial conditions.
372         dot_callback(0) # to set the initial conditions.
373
374
```

6.6.1.6 smart_callback()

```
def Forward_Kine_halfcircle.smart_callback (
            data )
```

Computes the linear acceleration, angular velocity projected on 0 given the data.

In case all the other callbacks have been called then it computes the velocity of the e.e. with respect to 0 and the end it calls the main_callback.

Parameters

data	coming from smartphone which provides a Imu() message.

Definition at line 248 of file Forward_Kine_halfcircle.py.

```
248 def smart_callback(data):
249     """!
250     Computes the linear acceleration, angular velocity projected on 0 given the data. In case all the other callbacks have been called then it computes the velocity of the
251     e.e. with respect to 0 and the end it calls the main_callback.
252     @param data: coming from smartphone which provides a Imu() message.
253     """
254 
255     if key == 1:
256 
257         global ini_smart, omega_imu_inert, a_imu_inert, R0inert, Rimu_inert_k, R0e_k, x_0e_kmin1, x_0e_k, v_0e_kmin1, v_0e_k, a_0e, omega_0e, key_bax, key_dot, key_smart, xeflag
258 
262 
263 
264         #global bigFilex,bigFiley
265         # Get orientation
266         orientation = [ data.orientation.x, data.orientation.y, data.orientation.z, data.orientation.w]
267 
268         # transform quaternion to euler angles
269         tempAngles = tf.transformations.euler_from_quaternion(orientation, "sxyz")
270 
271         angles = util.anglesCompensate(tempAngles)
272 
273         # Goal orientation matrix
274         Rimu_inert_k = util.eulerAnglesToRotationMatrix(angles)
275 
276         if ini_smart == 0:
277             R0inert = np.identity(3)
278             ini_smart = ini_smart + 1
279 
280         Rinert_imu_k = np.transpose(Rimu_inert_k)
281 
282         # angular velocity of imu (end effector) w.r.t. inertial frame projected on imu frame
283         omega_imu_inert[0][0] = data.angular_velocity.x
284         omega_imu_inert[1][0] = data.angular_velocity.y
285         omega_imu_inert[2][0] = data.angular_velocity.z
286 
287         # linear acceleration of imu (end effector) w.r.t. inertial frame projected on imu frame
288         a_imu_inert[0][0] = data.linear_acceleration.x
289         a_imu_inert[1][0] = data.linear_acceleration.y
290         a_imu_inert[2][0] = data.linear_acceleration.z
291 
292         # imu frame at time k is superimposed to e.e. frame at time k. Innertial and zero
293         # are not moving and since the inertial is placed where the e.e. was at its initial conditions,
294         # i can compute R0e_k
295         R0e_k = np.dot(R0inert, Rinert_imu_k) # R0e_k it is equal to Rinert_imu_k, since R0inert = id
296 
297         # Since inertial is not moving, the angular velocity and linear acceleration are the same
298         # if calculated w.r.t. 0, however i need to project them in zero.
299         omega_0e = omega_imu_inert
300 
301         a_0e = a_imu_inert
302 
303 
306 
307         # Target velocity.
308         if (xeflag == 0):
309             v_0e_k = v_0e_kmin1
310         else:
311             v_0e_k = v_0e_kmin1 + a_0e*dt
312 
313         # Target position.
314         if (xeflag == 0):
315             x_0e_k = x_0e_kmin1
316         else:
317             x_0e_k = x_0e_kmin1 + v_0e_kmin1*dt + 0.5*a_0e*dt*dt
318 
319         key_smart = key_smart + 1
320 
321         if (key_bax >= 1 and key_dot >= 1 and key_smart >= 1):
322             key_bax = 0
323             key_dot = 0
324             key_smart = 0
325             if (xeflag == 0): xeflag = 1
326             main_callback()
327 
328         # Update this vectors to compute integration at next steps.
```

```

        x_0e_kmin1 = x_0e_k
        v_0e_kmin1 = v_0e_k


```

6.6.2 Variable Documentation

6.6.2.1 a_0e

```
Forward_Kine_halfcircle.a_0e = np.zeros((3,1))
```

Definition at line 95 of file Forward_Kine_halfcircle.py.

6.6.2.2 a_imu_inert

```
Forward_Kine_halfcircle.a_imu_inert = np.zeros((3,1))
```

Definition at line 92 of file Forward_Kine_halfcircle.py.

6.6.2.3 DH

```
Forward_Kine_halfcircle.DH
```

Initial value:

```
=  np.array([[0, 0, L0, 0],
                [-p/2, L1, 0, p/2],
                [p/2, 0, L2, 0],
                [-p/2, L3, 0, 0],
                [p/2, 0, L4, 0],
                [-p/2, L5, 0, 0],
                [p/2, 0, 0, 0],
                [0, 0, L6, 0]])
```

Definition at line 33 of file Forward_Kine_halfcircle.py.

6.6.2.4 dt

```
float Forward_Kine_halfcircle.dt = 0.01
```

Definition at line 68 of file Forward_Kine_halfcircle.py.

6.6.2.5 flag_bax

```
int Forward_Kine_halfcircle.flag_bax = 0
```

Definition at line 64 of file Forward_Kine_halfcircle.py.

6.6.2.6 flag_dot

```
int Forward_Kine_halfcircle.flag_dot = 0
```

Definition at line 65 of file Forward_Kine_halfcircle.py.

6.6.2.7 info

```
Forward_Kine_halfcircle.info = np.array([1, 1, 1, 1, 1, 1, 1])
```

Definition at line 49 of file Forward_Kine_halfcircle.py.

6.6.2.8 ini_bax

```
int Forward_Kine_halfcircle.ini_bax = 0
```

Definition at line 53 of file Forward_Kine_halfcircle.py.

6.6.2.9 ini_dot

```
int Forward_Kine_halfcircle.ini_dot = 0
```

Definition at line 54 of file Forward_Kine_halfcircle.py.

6.6.2.10 ini_smart

```
int Forward_Kine_halfcircle.ini_smart = 0
```

Definition at line 55 of file Forward_Kine_halfcircle.py.

6.6.2.11 Jkmin1

```
Forward_Kine_halfcircle.Jkmin1 = np.zeros((6,7))
```

Definition at line 46 of file Forward_Kine_halfcircle.py.

6.6.2.12 key

```
int Forward_Kine_halfcircle.key = 1
```

Definition at line 61 of file Forward_Kine_halfcircle.py.

6.6.2.13 key_bax

```
int Forward_Kine_halfcircle.key_bax = 0
```

Definition at line 58 of file Forward_Kine_halfcircle.py.

6.6.2.14 key_dot

```
int Forward_Kine_halfcircle.key_dot = 0
```

Definition at line 60 of file Forward_Kine_halfcircle.py.

6.6.2.15 key_smart

```
int Forward_Kine_halfcircle.key_smart = 0
```

Definition at line 59 of file Forward_Kine_halfcircle.py.

6.6.2.16 L0

```
float Forward_Kine_halfcircle.L0 = 0.27035
```

Definition at line 22 of file Forward_Kine_halfcircle.py.

6.6.2.17 L1

```
float Forward_Kine_halfcircle.L1 = 0.06900
```

Definition at line 23 of file Forward_Kine_halfcircle.py.

6.6.2.18 L2

```
float Forward_Kine_halfcircle.L2 = 0.36435
```

Definition at line 24 of file Forward_Kine_halfcircle.py.

6.6.2.19 L3

```
float Forward_Kine_halfcircle.L3 = 0.06900
```

Definition at line 25 of file Forward_Kine_halfcircle.py.

6.6.2.20 L4

```
float Forward_Kine_halfcircle.L4 = 0.37429
```

Definition at line 26 of file Forward_Kine_halfcircle.py.

6.6.2.21 L5

```
float Forward_Kine_halfcircle.L5 = 0.01000
```

Definition at line 27 of file Forward_Kine_halfcircle.py.

6.6.2.22 L6

```
float Forward_Kine_halfcircle.L6 = 0.36830
```

Definition at line 28 of file Forward_Kine_halfcircle.py.

6.6.2.23 n_joints

```
int Forward_Kine_halfcircle.n_joints = 7
```

Definition at line 19 of file Forward_Kine_halfcircle.py.

6.6.2.24 omega_0e

```
Forward_Kine_halfcircle.omega_0e = np.zeros((3,1))
```

Definition at line 94 of file Forward_Kine_halfcircle.py.

6.6.2.25 omega_imu_inert

```
Forward_Kine_halfcircle.omega_imu_inert = np.zeros((3,1))
```

Definition at line 91 of file Forward_Kine_halfcircle.py.

6.6.2.26 p

```
Forward_Kine_halfcircle.p = np.pi
```

Definition at line 18 of file Forward_Kine_halfcircle.py.

6.6.2.27 pub_err

```
Forward_Kine_halfcircle.pub_err = rospy.Publisher('Data_for_errors', Float64MultiArray, queue←
_size=10)
```

Definition at line 14 of file Forward_Kine_halfcircle.py.

6.6.2.28 pub_jac

```
Forward_Kine_halfcircle.pub_jac = rospy.Publisher('jacobian', Float64MultiArray, queue_←
size=10)
```

Definition at line 16 of file Forward_Kine_halfcircle.py.

6.6.2.29 pub_track

```
Forward_Kine_halfcircle.pub_track = rospy.Publisher('tracking', Float64MultiArray, queue_↩
size=10)
```

Definition at line 15 of file Forward_Kine_halfcircle.py.

6.6.2.30 q

```
Forward_Kine_halfcircle.q = np.zeros(7)
```

Definition at line 81 of file Forward_Kine_halfcircle.py.

6.6.2.31 q_dot

```
Forward_Kine_halfcircle.q_dot = np.zeros((7,1))
```

Definition at line 82 of file Forward_Kine_halfcircle.py.

6.6.2.32 R0e_ini

```
Forward_Kine_halfcircle.R0e_ini = np.zeros((3,3))
```

Definition at line 86 of file Forward_Kine_halfcircle.py.

6.6.2.33 R0e_k

```
Forward_Kine_halfcircle.R0e_k = np.zeros((3,3))
```

Definition at line 89 of file Forward_Kine_halfcircle.py.

6.6.2.34 R0e_kmin1

```
Forward_Kine_halfcircle.R0e_kmin1 = np.zeros((3,3))
```

Definition at line 88 of file Forward_Kine_halfcircle.py.

6.6.2.35 R0inert

```
Forward_Kine_halfcircle.R0inert = np.zeros((3,3))
```

Definition at line 85 of file Forward_Kine_halfcircle.py.

6.6.2.36 Rimu_inert_k

```
Forward_Kine_halfcircle.Rimu_inert_k = np.zeros((3,3))
```

Definition at line 87 of file Forward_Kine_halfcircle.py.

6.6.2.37 T_dh

```
Forward_Kine_halfcircle.T_dh = t.DH_to_T(DH)
```

Definition at line 43 of file Forward_Kine_halfcircle.py.

6.6.2.38 v_0e_k

```
Forward_Kine_halfcircle.v_0e_k = v_0e_kmin1
```

Definition at line 77 of file Forward_Kine_halfcircle.py.

6.6.2.39 v_0e_kmin1

```
Forward_Kine_halfcircle.v_0e_kmin1 = np.zeros((3,1))
```

Definition at line 76 of file Forward_Kine_halfcircle.py.

6.6.2.40 v_0e_kmin1B

```
Forward_Kine_halfcircle.v_0e_kmin1B = np.zeros((3,1))
```

Definition at line 78 of file Forward_Kine_halfcircle.py.

6.6.2.41 x_0e_k

```
Forward_Kine_halfcircle.x_0e_k = x_0e_kmin1
```

Definition at line 72 of file Forward_Kine_halfcircle.py.

6.6.2.42 x_0e_kmin1

```
Forward_Kine_halfcircle.x_0e_kmin1 = np.zeros((3,1))
```

Definition at line 71 of file Forward_Kine_halfcircle.py.

6.6.2.43 x_0e_kmin1B

```
Forward_Kine_halfcircle.x_0e_kmin1B = np.zeros((3,1))
```

Definition at line 73 of file Forward_Kine_halfcircle.py.

6.6.2.44 xeflag

```
int Forward_Kine_halfcircle.xeflag = 0
```

Definition at line 29 of file Forward_Kine_halfcircle.py.

6.7 Forward_Kine_JT Namespace Reference

Functions

- def main_callback ()

  Publish to error node and inverse kinematics nodes the data, in particular the jacobian matrix, the tracking vectors and the data needed to compute the errors.
- def baxter_callback (data)

  Computes the configuration of baxter's arm whenever the data are available, then extracts the rotation matrix from 0 to e.e and the position of the e.e.
- def dot_callback (data)

  Reads the q_dots provided by the weighter.
- def smart_callback (data)

  Computes the linear acceleration, angular velocity projected on 0 given the data.
- def simulate_callback (data)

  Handles changes in the simulation.
- def subs ()

Variables

- pub_err = rospy.Publisher('Data_for_errors', Float64MultiArray, queue_size=10)
- pub_track = rospy.Publisher('tracking', Float64MultiArray, queue_size=10)
- pub_jac = rospy.Publisher('jacobian', Float64MultiArray, queue_size=10)
- pub_axes = rospy.Publisher('axes', Float64MultiArray, queue_size=10)

    New part.
- axis_vect = np.zeros((9,3))
- p = np.pi
- int n_joints = 7
- float L0 = 0.27035
- float L1 = 0.06900
- float L2 = 0.36435
- float L3 = 0.06900
- float L4 = 0.37429
- float L5 = 0.01000
- float L6 = 0.36830
- int xeflag = 0
- DH
- T_dh = t.DH_to_T(DH)
- Jkmin1 = np.zeros((6,7))
- info = np.array([1, 1, 1, 1, 1, 1, 1])
- int ini_bax = 0
- int ini_dot = 0
- int ini_smart = 0
- int key_bax = 0
- int key_smart = 0
- int key_dot = 0
- int key = -1
- int flag_bax = 0
- int flag_dot = 0
- float dt = 0.01
- x_0e_kmin1 = np.zeros((3,1))
- x_0e_k = x_0e_kmin1
- x_0e_kmin1B = np.zeros((3,1))
- v_0e_kmin1 = np.zeros((3,1))
- v_0e_k = v_0e_kmin1
- v_0e_kmin1B = np.zeros((3,1))
- q = np.zeros(7)
- q_dot = np.zeros((7,1))
- R0inert = np.zeros((3,3))
- R0e_ini = np.zeros((3,3))
- Reimu_ini
- Rimu_inert_k = np.zeros((3,3))
- R0e_kmin1 = np.zeros((3,3))
- R0e_k = np.zeros((3,3))
- omega_imu_inert = np.zeros((3,1))
- a_imu_inert = np.zeros((3,1))
- omega_0e = np.zeros((3,1))
- a_0e = np.zeros((3,1))

6.7.1 Function Documentation

6.7.1.1 baxter_callback()

```
def Forward_Kine_JT.baxter_callback (
              data )
```

Computes the configuration of baxter's arm whenever the data are available, then extracts the rotation matrix from 0 to e.e and the position of the e.e.

with respect to 0. It also computes the jacobian matrix. In case all the other callbacks have been called then it computes the velocity of the e.e. with respect to 0 and the end it calls the main_callback.

Parameters

data	coming from baxter node which provides a JointState message.

Definition at line 161 of file Forward_Kine_JT.py.

```
161 def baxter_callback(data):
162     """!
163     Computes the configuration of baxter’s arm whenever the data are available, then extracts the rotation
164     matrix from 0 to e.e and the position of the e.e. with respect to 0. It also computes the jacobian matrix.
165     In case all the other callbacks have been called then it computes the velocity of the
166     e.e. with respect to 0 and the end it calls the main_callback.
167     @param data: coming from baxter node which provides a JointState message.
168     """
169 
170     global axis_vect, ini_bax, q, R0e_kmin1, R0e_ini, Jkmin1, x_0e_kmin1B, x_0e_kmin1, v_0e_kmin1B, key_bax, key_dot, key_smart, flag_bax, flag_dot
171 
172     if (key == 1 or ini_bax == 0):
173         #start = time.time()
174 
175 
178         if ini_bax != 0:
179             flag_bax = data.effort[0]
180             # configuration at time kmin1
181             q = np.array(data.velocity)
182             #print("~~~~~")
183             #print(int(flag_bax))
184             #print(int(flag_dot))
185             #print(int(flag_bax) == int(flag_dot))
186             #print("~~~~~")
187 
188         if int(flag_bax) == int(flag_dot):
189             # relative T’s with the configuration passed.
190             T_rel_kmin1 = t.transformations(T_dh, q, info)
191             #print(T_rel_kmin1)
192             # absolute T’s
193             T_abs_kmin1 = t.abs_trans(T_rel_kmin1)
194 
195             # geometric vectors needed to compute jacobian.
196             geom = j.geometric_vectors(T_abs_kmin1)
197 
198 
200 
201             # axes of joints projected on zero
202             i_j = j.i_j(T_abs_kmin1)
203 
204             # vector containing some axes.
205             axis_vect = j.axis_vector(i_j[0], i_j[1], geom[0])
206 
207 
208             # jacobian computation
209             Jkmin1 = j.jacob(geom[0], geom[1], n_joints, info)
210 
211             # Transformation matrix from 0 to end effector at time k
212             T0e_kmin1 = T_abs_kmin1[7]
213 
215 
216             # end effector position of baxter at time k
217             for i in range(3):
218                 x_0e_kmin1B[i] = T0e_kmin1[i][3]
219 
220             # end effector orientation of baxter at time k. At time 0 i have the
221             # orientation of zero with respect of inertial frame also.
222             for i in range(3):
```

```
                for k in range(3):
                    R0e_kmin1[i][k] = T0e_kmin1[i][k]

            #print("----")
            #print("first  T0e:")
            #print(T0e_kmin1)
            if ini_bax == 0:
                #print("Init bax")
                #R0inert = R0e_kmin1 # Constant in time.
                #print(R0e_kmin1)
                R0e_ini = R0e_kmin1 # equal at starting configuration
                #x_0e_kmin1 =  x_0e_kmin1B # Initially they are equal
                x_0e_kmin1 = np.array([[ 1.1759, -4.3562e-06, 0.1913]]).transpose() # Initially they are equa
                ini_bax = ini_bax + 1

            key_bax = key_bax + 1

            if (key_bax >= 1 and key_dot >= 1):
                x_dot = np.dot(Jkmin1, q_dot)
                for i in range(3):
                    v_0e_kmin1B[i][0] = x_dot[i][0]

                if(key_smart >= 1):
                    key_bax = 0
                    key_dot = 0
                    key_smart = 0

                    main_callback()

        #end = time.time()
        #print("Bax Frequency: " + str(1/(end-start)))
```

6.7.1.2 dot_callback()

def Forward_Kine_JT.dot_callback (
data)

Reads the q_dots provided by the weighter.

In case all the other callbacks have been called then it computes the velocity of the e.e. with respect to 0 and the end it calls the main_callback.

Parameters

data	coming from weighter node which provides a JointState message.

Definition at line 255 of file Forward_Kine_JT.py.

```
255 def dot_callback(data):
256     """!
257     Reads the q_dots provided by the weighter. In case all the other callbacks have been called then it computes the velocity of the
258     e.e. with respect to 0 and the end it calls the main_callback.
259     @param data: coming from weighter node which provides a JointState message.
260     """
261 
262     global ini_dot, q_dot, Jkmin1, v_0e_kmin1B, key_bax, key_dot, key_smart, flag_bax, flag_dot
263 
264     if (key == 1 or ini_dot == 0):
265 
266 
270         if ini_dot != 0:
271             flag_dot = data.effort[0]
272             q_dot = np.transpose(np.array([data.velocity]))
273 
274         key_dot = key_dot + 1
275 
276         if ini_dot == 0:
277             #print("Init dot")
278             ini_dot = ini_dot + 1
```

```

        if (key_bax >= 1 and key_dot >= 1):
            x_dot = np.dot(Jkmin1, q_dot)
            for i in range(3):
                v_0e_kmin1B[i][0] = x_dot[i][0]

            if(key_smart >= 1):
                key_bax = 0
                key_dot = 0
                key_smart = 0

                main_callback()


```

6.7.1.3 main_callback()

```
def Forward_Kine_JT.main_callback ( )
```

Publish to error node and inverse kinematics nodes the data, in particular the jacobian matrix, the tracking vectors and the data needed to compute the errors.

Definition at line 111 of file Forward_Kine_JT.py.

```
111 def main_callback():
112     """!
113     Publish to error node and inverse kinematics nodes the data, in particular the
114     jacobian matrix, the tracking vectors and the data needed to compute the errors.
115     """
116
117
119     global axis_vect, x_0e_kmin1B, x_0e_k, v_0e_kmin1B, v_0e_k, a_0e, omega_0e, R0e_kmin1, R0e_k, Jkmin1
120
121
124     J = util.init_float64_multiarray(6*7, 1)
125     J.data = Jkmin1.reshape(6*7, 1)
126     pub_jac.publish(J)
127
128
131
132     # Send R0e_k, R0e_kmin1; x_0e_k, x_0e_kmin1B; v_0e_k, v_0e_kmin1B;
133     Rg_Re_xg_xe_vg_ve = np.array([R0e_k[0][0], R0e_k[0][1], R0e_k[0][2], R0e_k[1][0], R0e_k[1][1],
       R0e_k[1][2], R0e_k[2][0], R0e_k[2][1], R0e_k[2][2], R0e_kmin1[0][0], R0e_kmin1[0][1],
       R0e_kmin1[0][2], R0e_kmin1[1][0], R0e_kmin1[1][1], R0e_kmin1[1][2], R0e_kmin1[2][0], R0e_kmin1[2][1],

134     R0e_kmin1[2][2], x_0e_k[0][0], x_0e_k[1][0], x_0e_k[2][0], x_0e_kmin1B[0][0], x_0e_kmin1B[1][0],
       x_0e_kmin1B[2][0], v_0e_k[0][0], v_0e_k[1][0], v_0e_k[2][0], v_0e_kmin1B[0][0], v_0e_kmin1B[1][0],
       v_0e_kmin1B[2][0]], dtype=np.float_)
135
136     R_x_v = util.init_float64_multiarray(30, 1)
137     R_x_v.data = Rg_Re_xg_xe_vg_ve
138     pub_err.publish(R_x_v)
139
140
143
144     # send v_0e_k, omega_0e, a_0e
145     vg_omega_a = np.array([v_0e_k[0][0], v_0e_k[1][0], v_0e_k[2][0],
146     omega_0e[0][0], omega_0e[1][0], omega_0e[2][0],
147     a_0e[0][0], a_0e[1][0], a_0e[2][0]], dtype=np.float_)
148     v_w_a = util.init_float64_multiarray(9, 1)
149     v_w_a.data = vg_omega_a
150     pub_track.publish(v_w_a)
151
152
154     ax = util.init_float64_multiarray(27, 1)
155     ax.data = axis_vect.reshape(27, 1)
156     pub_axes.publish(ax)
157
158
159     #print("Published")
160
```

6.7.1.4 simulate_callback()

```
def Forward_Kine_JT.simulate_callback (
              data )
```

Handles changes in the simulation.

If data = 0, then the initial conditions must be resetted. If data = 1, then the algorithm moves on. If data = 2, then the algorithm pauses.

Parameters

data	coming from coppelia_sim.

Definition at line 401 of file Forward_Kine_JT.py.

```
def simulate_callback(data):
    """!
    Handles changes in the simulation. If data = 0, then the initial conditions must be resetted.
    If data = 1, then the algorithm moves on. If data = 2, then the algorithm pauses.
    @param data: coming from coppelia_sim.
    """

    global ini_bax, ini_dot, ini_smart, q, q_dot, v_0e_kmin1, key_bax, key_smart, key_dot, key,
   flag_bax, flag_dot

    key = data.data

    if key == 0:
        print("Resetting")
        # reset of the initial conditions.

        # Need this variable to store initial rotation matrix, because it's equal to
        # rotation matrix from 0 to inertial frame.
        ini_bax = 0
        ini_dot = 0 # needed to differentiate from initial condition to computed qdots.
        ini_smart = 0

        # Sync variables.
        key_bax = 0
        key_smart = 0
        key_dot = 0

        # flag variables.
        flag_bax = 0
        flag_dot = 0

        # Initial velocity of end effector w.r.t. zero
        v_0e_kmin1 = np.zeros((3,1)) # starting velocity

        # Define the q's and q dots
        q = np.zeros(7)
        q_dot = np.zeros((7,1))

        baxter_callback(0) # to set the initial conditions.
        dot_callback(0) # to set the initial conditions.


```

6.7.1.5 smart_callback()

```
def Forward_Kine_JT.smart_callback (
              data )
```

Computes the linear acceleration, angular velocity projected on 0 given the data.

In case all the other callbacks have been called then it computes the velocity of the e.e. with respect to 0 and the end it calls the main_callback.

Parameters

data	coming from smartphone which provides a Imu() message.

Definition at line 293 of file Forward_Kine_JT.py.

```
293 def smart_callback(data):
294     """!
295     Computes the linear acceleration, angular velocity projected on 0 given the data. In case all the
    other callbacks have been called then it computes the velocity of the
296     e.e. with respect to 0 and the end it calls the main_callback.
297     @param data: coming from smartphone which provides a Imu() message.
298     """
299 
300     if key == 1:
301         #print("Starting")
302         #start = time.time()
303 
304         global ini_smart, omega_imu_inert, a_imu_inert, R0inert, Rimu_inert_k, R0e_k, x_0e_kmin1,
    x_0e_k, v_0e_kmin1, v_0e_k, a_0e, omega_0e, key_bax, key_dot, key_smart, xeflag
305 
309 
310 
311         #global bigFilex,bigFiley
312         # Get orientation
313         orientation = [ data.orientation.x, data.orientation.y, data.orientation.z, data.orientation.w]
314         #print("orient")
315         #print(orientation)
316         #print("----")
317 
318         # transform quaternion to euler angles
319         tempAngles = tf.transformations.euler_from_quaternion(orientation, "sxyz")
320 
321         angles = util.anglesCompensate(tempAngles)
322 
323         Rimu_inert_k = util.eulerAnglesToRotationMatrix(angles)
329         if ini_smart == 0:
330             R0inert = np.dot(np.dot(R0e_ini, Reimu_ini), Rimu_inert_k) # constant in the overall
    simulation.
331             R0inert = np.identity(3)
334             ini_smart = ini_smart + 1
335 
336         Rinert_imu_k = np.transpose(Rimu_inert_k)
337         #Rinert_imu_k = Rimu_inert_k    #### FOR TESTING
338 
339         # angular velocity of imu (end effector) w.r.t. inertial frame projected on imu frame
340         omega_imu_inert[0][0] = data.angular_velocity.x
341         omega_imu_inert[1][0] = data.angular_velocity.y
342         omega_imu_inert[2][0] = data.angular_velocity.z
343 
345 
346         # linear acceleration of imu (end effector) w.r.t. inertial frame projected on imu frame
347         a_imu_inert[0][0] = data.linear_acceleration.x
348         a_imu_inert[1][0] = data.linear_acceleration.y
349         a_imu_inert[2][0] = data.linear_acceleration.z
350 
352 
353         # imu frame at time k is superimposed to e.e. frame at time k. Innertial and zero
354         # are not moving and since the inertial is placed where the e.e. was at its initial conditions,
355         # i can compute R0e_k
356         R0e_k = np.dot(R0inert, Rinert_imu_k)
359 
360         # Since inertial is not moving, the angular velocity and linear acceleration are the same
361         # if calculated w.r.t. 0, however i need to project them in zero.
362         omega_0e = omega_imu_inert
364         a_0e = a_imu_inert
365 
366 
369 
370         # Target velocity.
371         if (xeflag == 0):
372             v_0e_k = v_0e_kmin1
373         else:
374             v_0e_k = v_0e_kmin1 + a_0e*dt
375         #print("v_0e_k: ")
376         #print(v_0e_k)
377 
378         # Target position.
379         if (xeflag == 0):
380             x_0e_k = x_0e_kmin1
381         else:
382             x_0e_k = x_0e_kmin1 + v_0e_kmin1*dt + 0.5*a_0e*dt*dt
383 
384         key_smart = key_smart + 1
```

```python

        if (key_bax >= 1 and key_dot >= 1 and key_smart >= 1):
            key_bax = 0
            key_dot = 0
            key_smart = 0
            if (xeflag == 0): xeflag = 1
            main_callback()

        # Update this vectors to compute integration at next steps.

        x_0e_kmin1 = x_0e_k
        v_0e_kmin1 = v_0e_k

       #end = time.time()
       #print("Smart Frequency: " + str(1/(end-start)))

```

6.7.1.6 subs()

def Forward_Kine_JT.subs ( )

Definition at line 442 of file Forward_Kine_JT.py.

```python
def subs():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.


    rospy.init_node('subs', anonymous=True)

    # Receive data from smartphone, baxter, weighter and coppelia.
    rospy.Subscriber("smartphone", Imu, smart_callback)
    rospy.Subscriber("logtopic", JointState, baxter_callback)
    rospy.Subscriber("cmdtopic", JointState, dot_callback)
    rospy.Subscriber("handleSimulation", Int8, simulate_callback)


    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()


```

6.7.2 Variable Documentation

6.7.2.1 a_0e

Forward_Kine_JT.a_0e = np.zeros((3,1))

Definition at line 109 of file Forward_Kine_JT.py.

6.7.2.2 a_imu_inert

Forward_Kine_JT.a_imu_inert = np.zeros((3,1))

Definition at line 106 of file Forward_Kine_JT.py.

6.7.2.3 axis_vect

```
Forward_Kine_JT.axis_vect = np.zeros((9,3))
```

Definition at line 24 of file Forward_Kine_JT.py.

6.7.2.4 DH

```
Forward_Kine_JT.DH
```

Initial value:

```
1 =  np.array([[0, 0, L0, 0],
2                [-p/2, L1, 0, p/2],
3                [p/2, 0, L2, 0],
4                [-p/2, L3, 0, 0],
5                [p/2, 0, L4, 0],
6                [-p/2, L5, 0, 0],
7                [p/2, 0, 0, 0],
8                [0, 0, L6, 0]])
```

Definition at line 42 of file Forward_Kine_JT.py.

6.7.2.5 dt

```
float Forward_Kine_JT.dt = 0.01
```

Definition at line 79 of file Forward_Kine_JT.py.

6.7.2.6 flag_bax

```
int Forward_Kine_JT.flag_bax = 0
```

Definition at line 75 of file Forward_Kine_JT.py.

6.7.2.7 flag_dot

```
int Forward_Kine_JT.flag_dot = 0
```

Definition at line 76 of file Forward_Kine_JT.py.

6.7.2.8 info

```
Forward_Kine_JT.info = np.array([1, 1, 1, 1, 1, 1, 1])
```

Definition at line 60 of file Forward_Kine_JT.py.

6.7.2.9 ini_bax

```
int Forward_Kine_JT.ini_bax = 0
```

Definition at line 64 of file Forward_Kine_JT.py.

6.7.2.10 ini_dot

```
int Forward_Kine_JT.ini_dot = 0
```

Definition at line 65 of file Forward_Kine_JT.py.

6.7.2.11 ini_smart

```
int Forward_Kine_JT.ini_smart = 0
```

Definition at line 66 of file Forward_Kine_JT.py.

6.7.2.12 Jkmin1

```
Forward_Kine_JT.Jkmin1 = np.zeros((6,7))
```

Definition at line 57 of file Forward_Kine_JT.py.

6.7.2.13 key

```
int Forward_Kine_JT.key = -1
```

Definition at line 72 of file Forward_Kine_JT.py.

6.7.2.14 key_bax

```
int Forward_Kine_JT.key_bax = 0
```

Definition at line 69 of file Forward_Kine_JT.py.

6.7.2.15 key_dot

```
int Forward_Kine_JT.key_dot = 0
```

Definition at line 71 of file Forward_Kine_JT.py.

6.7.2.16 key_smart

```
int Forward_Kine_JT.key_smart = 0
```

Definition at line 70 of file Forward_Kine_JT.py.

6.7.2.17 L0

```
float Forward_Kine_JT.L0 = 0.27035
```

Definition at line 31 of file Forward_Kine_JT.py.

6.7.2.18 L1

```
float Forward_Kine_JT.L1 = 0.06900
```

Definition at line 32 of file Forward_Kine_JT.py.

6.7.2.19 L2

```
float Forward_Kine_JT.L2 = 0.36435
```

Definition at line 33 of file Forward_Kine_JT.py.

6.7.2.20 L3

```
float Forward_Kine_JT.L3 = 0.06900
```

Definition at line 34 of file Forward_Kine_JT.py.

6.7.2.21 L4

```
float Forward_Kine_JT.L4 = 0.37429
```

Definition at line 35 of file Forward_Kine_JT.py.

6.7.2.22 L5

```
float Forward_Kine_JT.L5 = 0.01000
```

Definition at line 36 of file Forward_Kine_JT.py.

6.7.2.23 L6

```
float Forward_Kine_JT.L6 = 0.36830
```

Definition at line 37 of file Forward_Kine_JT.py.

6.7.2.24 n_joints

```
int Forward_Kine_JT.n_joints = 7
```

Definition at line 28 of file Forward_Kine_JT.py.

6.7.2.25 omega_0e

```
Forward_Kine_JT.omega_0e = np.zeros((3,1))
```

Definition at line 108 of file Forward_Kine_JT.py.

6.7.2.26 omega_imu_inert

```
Forward_Kine_JT.omega_imu_inert = np.zeros((3,1))
```

Definition at line 105 of file Forward_Kine_JT.py.

6.7.2.27 p

```
Forward_Kine_JT.p = np.pi
```

Definition at line 27 of file Forward_Kine_JT.py.

6.7.2.28 pub_axes

```
Forward_Kine_JT.pub_axes = rospy.Publisher('axes', Float64MultiArray, queue_size=10)
```

New part.

Definition at line 22 of file Forward_Kine_JT.py.

6.7.2.29 pub_err

```
Forward_Kine_JT.pub_err = rospy.Publisher('Data_for_errors', Float64MultiArray, queue_size=10)
```

Definition at line 15 of file Forward_Kine_JT.py.

6.7.2.30 pub_jac

```
Forward_Kine_JT.pub_jac = rospy.Publisher('jacobian', Float64MultiArray, queue_size=10)
```

Definition at line 17 of file Forward_Kine_JT.py.

6.7.2.31 pub_track

```
Forward_Kine_JT.pub_track = rospy.Publisher('tracking', Float64MultiArray, queue_size=10)
```

Definition at line 16 of file Forward_Kine_JT.py.

6.7.2.32 q

```
Forward_Kine_JT.q = np.zeros(7)
```

Definition at line 92 of file Forward_Kine_JT.py.

6.7.2.33 q_dot

```
Forward_Kine_JT.q_dot = np.zeros((7,1))
```

Definition at line 93 of file Forward_Kine_JT.py.

6.7.2.34 R0e_ini

```
Forward_Kine_JT.R0e_ini = np.zeros((3,3))
```

Definition at line 97 of file Forward_Kine_JT.py.

6.7.2.35 R0e_k

```
Forward_Kine_JT.R0e_k = np.zeros((3,3))
```

Definition at line 103 of file Forward_Kine_JT.py.

6.7.2.36 R0e_kmin1

```
Forward_Kine_JT.R0e_kmin1 = np.zeros((3,3))
```

Definition at line 102 of file Forward_Kine_JT.py.

6.7.2.37 R0inert

```
Forward_Kine_JT.R0inert = np.zeros((3,3))
```

Definition at line 96 of file Forward_Kine_JT.py.

6.7.2.38 Reimu_ini

```
Forward_Kine_JT.Reimu_ini
```

Initial value:

```
1 =  np.array([[-1, 0, 0],
2                    [0, -1, 0],
3                    [0, 0, 1]])
```

Definition at line 98 of file Forward_Kine_JT.py.

6.7.2.39 Rimu_inert_k

```
Forward_Kine_JT.Rimu_inert_k = np.zeros((3,3))
```

Definition at line 101 of file Forward_Kine_JT.py.

6.7.2.40 T_dh

```
Forward_Kine_JT.T_dh = t.DH_to_T(DH)
```

Definition at line 52 of file Forward_Kine_JT.py.

6.7.2.41 v_0e_k

```
Forward_Kine_JT.v_0e_k = v_0e_kmin1
```

Definition at line 88 of file Forward_Kine_JT.py.

6.7.2.42 v_0e_kmin1

```
Forward_Kine_JT.v_0e_kmin1 = np.zeros((3,1))
```

Definition at line 87 of file Forward_Kine_JT.py.

6.7.2.43 v_0e_kmin1B

```
Forward_Kine_JT.v_0e_kmin1B = np.zeros((3,1))
```

Definition at line 89 of file Forward_Kine_JT.py.

6.7.2.44 x_0e_k

```
Forward_Kine_JT.x_0e_k = x_0e_kmin1
```

Definition at line 83 of file Forward_Kine_JT.py.

6.7.2.45 x_0e_kmin1

```
Forward_Kine_JT.x_0e_kmin1 = np.zeros((3,1))
```

Definition at line 82 of file Forward_Kine_JT.py.

6.7.2.46 x_0e_kmin1B

```
Forward_Kine_JT.x_0e_kmin1B = np.zeros((3,1))
```

Definition at line 84 of file Forward_Kine_JT.py.

6.7.2.47 xeflag

```
int Forward_Kine_JT.xeflag = 0
```

Definition at line 38 of file Forward_Kine_JT.py.

6.8 imu_calib Namespace Reference

Classes

- class AccelCalib
- class ApplyCalib
- class DoCalib

6.9 integrator Namespace Reference

Functions

- def sat (x, xmin, xmax)
- def qdot_callback (qdot_data)

  Receives qdot vector and sents q vector obtained by integration.
- def simulate_callback (data)

  Handles changes in the simulation.
- def integr ()

  Integrator function.

Variables

- q = np.zeros((7,1))
- qdot = np.zeros((7,1))
- qdotprev = None
- qdotprevprev = None
- bool qdotppnone = True
- bool qdotpnone = True
- qold = np.zeros((7,1))
- int eff = 0
- int key = 0
- float DT = 0.01
- list qmin = [-1.6817,-2.1268,-3.0343,-0.3,-3.0396,-1.5508,-3.0396]
- list qmax = [1.6817,1.0272,3.0343,2.5829,3.0378,2.0744,3.0378]
- anonymous
- pub = rospy.Publisher('logtopic', JointState, queue_size=10)

6.9.1 Function Documentation

6.9.1.1 integr()

```
def integrator.integr ( )
```

Integrator function.

Definition at line 75 of file integrator.py.

```
75 def integr():
76     """!
77     Integrator function
78     """
79     global q,qdot,key
80     rospy.Subscriber("cmdtopic", JointState, qdot_callback) # subscribe to weighter
81     rospy.Subscriber("handleSimulation", Int8, simulate_callback) # subscribe to simulation messages
82 
83     # spin() simply keeps python from exiting until this node is stopped
84     rospy.spin()
85 
```

6.9.1.2 qdot_callback()

```
def integrator.qdot_callback (
              qdot_data )
```

Receives qdot vector and sents q vector obtained by integration.

@params qdot_data: qdot message received from weighter.

Definition at line 29 of file integrator.py.

```
29 def qdot_callback (qdot_data):
30     """!
31     Receives qdot vector and sents q vector obtained by integration.
32     @params qdot_data: qdot message received from weighter.
33     """
34     global q,qdot,eff,key,pub,qdotprev,qdotprevprev,qold,qdotpnone,qdotppnone,qmin,qmax
```

```
    qdot = np.array([qdot_data.velocity]).transpose() # store received vector in global variable
    eff = qdot_data.effort[0] # store new seq number
    tosend = JointState() # joint state object to be sent
    qtmp = q # store old q in qtmp

    # Integration
    if not qdotppnone: # can use Simpson integration
        q = qold + DT * (qdot + qdotprevprev + 4 * qdotprev) / 3
    elif not qdotpnone: # can use trap integration
        q = q + DT * (qdot + qdotprev) * .5
        #qdotppnone = False
    else: # can use rectangular integration
        q = q + DT * qdot
        qdotpnone = False

    for i in range(7):
        q[i] = sat(q[i],qmin[i],qmax[i])

    # Update past values
    qdotprev = qdot
    qdotprevprev = qdotprev
    qold = qtmp

    # Fill in and send object tosend
    tosend.position = q
    tosend.effort = [eff]
    tosend.header.stamp = rospy.Time.now()
    pub.publish(tosend)
```

6.9.1.3 sat()

```
def integrator.sat (
        x,
        xmin,
        xmax )
```

Definition at line 23 of file integrator.py.

```
23 def sat(x,xmin,xmax):
24     if x > xmax: return xmax
25     if x < xmin: return xmin
26     return x
27
28
```

6.9.1.4 simulate_callback()

```
def integrator.simulate_callback (
        data )
```

Handles changes in the simulation.

If data = 0, then the initial conditions must be resetted.

Parameters

data	coming from coppelia_sim.

Definition at line 65 of file integrator.py.

```
65 def simulate_callback(data):
```

```
    """!
    Handles changes in the simulation. If data = 0, then the initial conditions must be resetted.
    @param data: coming from coppelia_sim.
    """
    global q,key
    if data.data == 0:
        q = np.zeros((7,1));key = 0
```

6.9.2 Variable Documentation

6.9.2.1 anonymous

`integrator.anonymous`

Definition at line 88 of file integrator.py.

6.9.2.2 DT

`float integrator.DT = 0.01`

Definition at line 18 of file integrator.py.

6.9.2.3 eff

`int integrator.eff = 0`

Definition at line 16 of file integrator.py.

6.9.2.4 key

`int integrator.key = 0`

Definition at line 17 of file integrator.py.

6.9.2.5 pub

`integrator.pub = rospy.Publisher('logtopic', JointState, queue_size=10)`

Definition at line 89 of file integrator.py.

6.9.2.6 q

```
integrator.q = np.zeros((7,1))
```

Definition at line 9 of file integrator.py.

6.9.2.7 qdot

```
integrator.qdot = np.zeros((7,1))
```

Definition at line 10 of file integrator.py.

6.9.2.8 qdotpnone

```
bool integrator.qdotpnone = True
```

Definition at line 14 of file integrator.py.

6.9.2.9 qdotppnone

```
bool integrator.qdotppnone = True
```

Definition at line 13 of file integrator.py.

6.9.2.10 qdotprev

```
integrator.qdotprev = None
```

Definition at line 11 of file integrator.py.

6.9.2.11 qdotprevprev

```
integrator.qdotprevprev = None
```

Definition at line 12 of file integrator.py.

6.9.2.12 qmax

```
list integrator.qmax = [1.6817,1.0272,3.0343,2.5829,3.0378,2.0744,3.0378]
```

Definition at line 20 of file integrator.py.

6.9.2.13 qmin

```
list integrator.qmin = [-1.6817,-2.1268,-3.0343,-0.3,-3.0396,-1.5508,-3.0396]
```

Definition at line 19 of file integrator.py.

6.9.2.14 qold

```
integrator.qold = np.zeros((7,1))
```

Definition at line 15 of file integrator.py.

6.10 J_computations Namespace Reference

Functions

- def geometric_vectors (T_abs)

  Computes the vectors needed to compute geometric jacobian.
- def i_j (T_abs)

  New part.
- def axis_vector (i, j, k)

  Computes the vector needed for Jtransp optimisation.
- def jacob (k, r, n_joints, info)

  Computes the jacobian matrix given the geometric vectors, number of joints and info.

Variables

- J = np.concatenate((Jl, Ja), axis = 0)

  else: zero = np.array([[0], [0], [0]]) Ja = np.concatenate((Ja, zero), axis = 1) Jl = np.concatenate((Jl, k[i]), axis = 1)

6.10.1 Function Documentation

6.10.1.1 axis_vector()

```
def J_computations.axis_vector (
              i,
              j,
              k )
```

Computes the vector needed for Jtransp optimisation.

Parameters

i,j,k	joint axes projected on zero.

Returns

v: vector containing some of this axes.

Definition at line 60 of file J_computations.py.

```
60 def axis_vector(i, j, k):
61     """!
62     Computes the vector needed for Jtransp optimisation.
63     @param i, j, k: joint axes projected on zero.
64     @return v: vector containing some of this axes.
65     """
66 
67     v = np.zeros((9,3))
68 
69     v[0] = np.transpose(i[0])
70     v[1] = np.transpose(j[1])
71     v[2] = np.transpose(k[2])
72     v[3] = np.transpose(j[3])
73     v[4] = np.transpose(k[4])
74     v[5] = np.transpose(j[5])
75     v[6] = np.transpose(k[1])
76     v[7] = np.transpose(k[3])
77     v[8] = np.transpose(k[5])
78 
79     return v
80 
```

6.10.1.2 geometric_vectors()

```
def J_computations.geometric_vectors (
                T_abs )
```

Computes the vectors needed to compute geometric jacobian.

Parameters

Tabs	the transformation matrices from joint to 0 frame in current configuration

Returns

geom_v: geometric vectors exctracted from Tabs that allow to compute the jacobian.

Definition at line 3 of file J_computations.py.

```
3 def geometric_vectors(T_abs):
4     """!
5     Computes the vectors needed to compute geometric jacobian.
6     @param Tabs: the transformation matrices from joint to 0 frame in current configuration
7     @return geom_v: geometric vectors exctracted from Tabs that allow to compute the jacobian.
8     """
9     r = []
10    k = []
11    geom_v = []
12
13    n_matrices = len(T_abs)
14
15    for i in range(n_matrices-1):
16        tmp_k = np.array([[T_abs[i][0][2], T_abs[i][1][2], T_abs[i][2][2]]])
17        tmp_k = np.transpose(tmp_k)
```

```
        k.append(tmp_k)

        tmp_r = np.array([[T_abs[n_matrices-1][0][3] - T_abs[i][0][3], T_abs[n_matrices-1][1][3] -
        T_abs[i][1][3], T_abs[n_matrices-1][2][3] - T_abs[i][2][3]]])
        tmp_r = np.transpose(tmp_r)
        r.append(tmp_r)

    geom_v.append(k)
    geom_v.append(r)

    return geom_v

```

6.10.1.3 i_j()

```
def J_computations.i_j (
                T_abs )
```

New part.

Computes the vectors needed to optimize Jtransp

Parameters

Tabs	the transformation matrices from joint to 0 frame in current configuration

Returns

i_j: axes of joints projected on zero.

Definition at line 32 of file J_computations.py.

```
32 def i_j(T_abs):
33     """!
34     Computes the vectors needed to optimize Jtransp
35     @param Tabs: the transformation matrices from joint to 0 frame in current configuration
36     @return i_j: axes of joints projected on zero.
37     """
38
39     i = []
40     j = []
41     i_j = []
42
43     n_matrices = len(T_abs)
44
45     for h in range(n_matrices-1):
46         tmp_i = np.array([[T_abs[h][0][0], T_abs[h][1][0], T_abs[h][2][0]]])
47         tmp_j = np.array([[T_abs[h][0][1], T_abs[h][1][1], T_abs[h][2][1]]])
48
49         tmp_i = np.transpose(tmp_i)
50         tmp_j = np.transpose(tmp_j)
51
52         i.append(tmp_i)
53         j.append(tmp_j)
54
55     i_j.append(i)
56     i_j.append(j)
57
58     return i_j
59
```

6.10.1.4 jacob()

```
def J_computations.jacob (
                k,
                r,
                n_joints,
                info )
```

Computes the jacobian matrix given the geometric vectors, number of joints and info.

Parameters

k	versors of axis z of the joints projected on 0.
r	distance between joints and e.e. projected on 0.
n_joints	explains it self.
info	1->revolute, 0->prismatic. In case there is a change in the serial chain the algorithm still works.

Returns

J: jacobian matrix.

Definition at line 83 of file J_computations.py.

```
83 def jacob(k, r, n_joints, info):
84   """!
85   Computes the jacobian matrix given the geometric vectors, number of joints and info.
86   @param k: versors of axis z of the joints projected on 0.
87   @param r: distance between joints and e.e. projected on 0.
88   @param n_joints: explains it self.
89   @param info: 1->revolute, 0->prismatic. In case there is a change in the serial chain the algorithm
       still works.
90   @return J: jacobian matrix.
91   """
92
93   Ja = np.array([[],
94                  [],
95                  []])
96   Jl = np.array([[],
97                  [],
98                  []])
99
100   for i in range(n_joints):
101     if info[i] == 1:
102       Ja = np.concatenate((Ja, k[i]), axis = 1)
103       kx = k[i][0][0]
104       ky = k[i][1][0]
105       kz = k[i][2][0]
106       k_skew = np.array([[0, -kz, ky],
107                          [kz, 0, -kx],
108                          [-ky, kx, 0]])
109       l_column = np.dot(k_skew, r[i])
110       Jl = np.concatenate((Jl, l_column), axis = 1)
111
```

6.10.2 Variable Documentation

6.10.2.1 J

```
J_computations.J = np.concatenate((Jl, Ja), axis = 0)
```

else: zero = np.array([[0], [0], [0]]) Ja = np.concatenate((Ja, zero), axis = 1) Jl = np.concatenate((Jl, k[i]), axis = 1)

Definition at line 118 of file J_computations.py.

6.11 J_Transp_server Namespace Reference

Functions

- def init_float64_multiarray (rows, columns)

  Function that initializes a Float64MultiArray of size rows x columns.
- def j_transp (err, err_dot, J, Wp, Wd, delta_t)

  Function that performs Inverse kinematics using the Jacobian Transpose approach.
- def handle_IK_Jtransp (req)
- def error_callback (message)
- def jacobian_callback (message)
- def JT_server ()

Variables

- bool readyErr = False
- bool readyJ = False
- list wp = [50, 50, 50, 1, 1, 1]
- list wd = [50,50,50,0.5,0.5,0.5]
- script_dir = os.path.dirname(__file__)
- string rel_path = "./Output/Output.txt"
- abs_file_path = os.path.join(script_dir, rel_path)
- int LOG_FLAG = 0

6.11.1 Function Documentation

6.11.1.1 error_callback()

```
def J_Transp_server.error_callback (
                message )
```

Definition at line 133 of file J_Transp_server.py.

```
133 def error_callback(message):
134 
135     # Declaration to work with global variables
136     global error,readyErr
137 
138     # Re-arranging the error vector in the needed form:
139     err_orient = np.array([message.data[:3]]).T
140     err_pos = np.array([message.data[3:6]]).T
141     err_vel_lin = np.array([message.data[6:9]]).T
142     err_vel_rot = np.array([[0,0,0]]).T
143     error = np.concatenate((err_pos, err_orient, err_vel_lin, err_vel_rot), axis=0)
144     #error = np.array([message.data[:6]]).T
145     rospy.loginfo("Received Position Error:\n%s\n", str(error))
146     #print(readyErr)
147 
148     # Set the Error as available
149     readyErr = True
150     #print(readyErr)
151 
152 # Callback Function for the Jacobian matrix
```

6.11.1.2 handle_IK_Jtransp()

```
def J_Transp_server.handle_IK_Jtransp (
            req )
```

Definition at line 107 of file J_Transp_server.py.

```
107 def handle_IK_Jtransp(req):
108
109     # Declaration to work with global variables
110     global readyErr, readyJ, wp, wd
111
112     print"Server J Transpose accepted request\n"
113
114     if (not (readyErr and readyJ)):
115         readyErr = readyJ = False
116         rospy.logerr("J Transpose service could not run: missing data.")
117         return
118     else:
119         readyErr = readyJ = False
120     if (LOG_FLAG):
121             now = rospy.get_rostime()
122             timestamp =  now.secs + float(now.nsecs)/1000000000
123             with open(abs_file_path, 'a') as f:
124                 f.write(str(timestamp))
125                 f.write("\t")
126                 f.write(str(j_transp(error[:6], error[6:], J, wp, wd, 0.01).velocity.T))
127                 f.write("\n\n")
128
129         return IK_JtraResponse(j_transp(error[:6], error[6:], J, wp, wd, 0.01))
130     #return IK_JtraResponse(j_transp(error, J, 0.01))
131
132 # Callback Function for the error on the position (error on Xee)
```

6.11.1.3 init_float64_multiarray()

```
def J_Transp_server.init_float64_multiarray (
            rows,
            columns )
```

Function that initializes a Float64MultiArray of size rows x columns.

Parameters

rows	Number of rows of the returned multiarray.
columns	Number of columns of the returned multiarray.

Returns

empty Float64MultiArray instance.

Definition at line 34 of file J_Transp_server.py.

```
34 def init_float64_multiarray(rows,columns):
35     """!
36     Function that initializes a Float64MultiArray of size rows x columns.
37     @param rows: Number of rows of the returned multiarray.
38     @param columns: Number of columns of the returned multiarray.
39     @return empty Float64MultiArray instance.
40     """
41     a = Float64MultiArray()
42     a.layout.dim.append(MultiArrayDimension())
43     a.layout.dim.append(MultiArrayDimension())
44     a.layout.dim[0].label ="rows"
45     a.layout.dim[0].size = rows
46     a.layout.dim[1].label ="columns"
47     a.layout.dim[1].size = columns
```

```
    return a

```

6.11.1.4 j_transp()

```
def J_Transp_server.j_transp (
              err,
              err_dot,
              J,
              Wp,
              Wd,
              delta_t )
```

Function that performs Inverse kinematics using the Jacobian Transpose approach.

Parameters

err	the Error on the position and orientation of the end effector.
J	the Jacobian matrix of the manipulator.
delta_t	sampling time.

Returns

: a Float64MultiArray containing the Joint Velocities.

Definition at line 50 of file J_Transp_server.py.

```
50 def j_transp(err, err_dot, J, Wp, Wd, delta_t):
51     """!
52     Function that performs Inverse kinematics using the Jacobian Transpose
53     approach.
54     @param err: the Error on the position and orientation of the end effector.
55     @param J: the Jacobian matrix of the manipulator.
56     @param delta_t: sampling time.
57     @return: a Float64MultiArray containing the Joint Velocities.
58     """
59     # Norm of the position error
60     err_norm = np.linalg.norm(err)
61 
62     # Norm of the linear part of the position error
63     err_norm_lin = np.linalg.norm(err[:3])
64 
65     # Norm of the rotational part of the position error
66     err_norm_rot = np.linalg.norm(err[3:])
67 
68     # Norm of the derivative of the position error
69     err_dot_norm = np.linalg.norm(err_dot)
70 
71     # Norm of the linear part of the derivative of the position error
72     err_dot_lin = np.linalg.norm(err_dot[:3])
73 
74     # Norm of the rotational part of the derivative of the position error
75     err_dot_rot = np.linalg.norm(err_dot[3:])
76 
77     # q_dot initialization
78     q_dot = JointState()
79 
80     # Working Case
81     if ((err_norm>0.0001) or (err_dot_norm>0.001)):
82 
83         # Weights as diagonal Matrices:
84         Kp = np.diag([err_norm_lin*10, err_norm_lin*10, err_norm_lin*10, err_norm_rot, err_norm_rot,
    err_norm_rot]) #200 and 50 before
85         #Kp = np.diag(Wp)
86 
```

```
87          Kd = np.diag([err_dot_lin*0, err_dot_lin*0, err_dot_lin*0, 5*err_dot_rot, 5*err_dot_rot,
       5*err_dot_rot])
88          #Kd = np.diag(Wd)
89
90          # Delta Joint positions using: K*J_transpose*error_transpose (Paper Formula)
91          dq = J.T.dot(Kp.dot(err)+Kd.dot(err_dot))
92
93          # Joint velocities, being dq = q_dot*delta_t
94          #q_dot.velocity = dq/delta_t
95      q_dot.velocity = np.zeros((max(J.shape))).T
96
97      # Position Reached
98      else:
99
100          # Stop the Joint: zero velocities
101          q_dot.velocity = np.zeros((max(J.shape))).T
102
103      return q_dot
104
105
106 # Handler for the Server
```

6.11.1.5 jacobian_callback()

```
def J_Transp_server.jacobian_callback (
                message )
```

Definition at line 153 of file J_Transp_server.py.

```
153 def jacobian_callback(message):
154
155     # Declaration to work with global variables
156     global J, readyJ
157
158     J = np.array(message.data)
159     J = J.reshape(6,7)
160     #rospy.loginfo("Received Jacobian::\n%s\n", str(J))
161
162     # Set the J as available
163     readyJ = True
164
165 # Main body containing 2 Subscribers and the Service defition
```

6.11.1.6 JT_server()

```
def J_Transp_server.JT_server ( )
```

Definition at line 166 of file J_Transp_server.py.

```
166 def JT_server():
167
168     # Node Initialization
169     rospy.init_node('IK_Jtransp_server')
170
171     rospy.loginfo("Server Initialized\n")
172
173     # Subscribers
174     rospy.Subscriber("errors", Float64MultiArray, error_callback)
175     rospy.Subscriber("jacobian", Float64MultiArray, jacobian_callback)
176
177     s = rospy.Service('IK_Jtransp', IK_Jtra, handle_IK_Jtransp)
178     rospy.spin()
179
```

6.11.2 Variable Documentation

6.11.2.1 abs_file_path

```
J_Transp_server.abs_file_path = os.path.join(script_dir, rel_path)
```

Definition at line 31 of file J_Transp_server.py.

6.11.2.2 LOG_FLAG

```
int J_Transp_server.LOG_FLAG = 0
```

Definition at line 32 of file J_Transp_server.py.

6.11.2.3 readyErr

```
bool J_Transp_server.readyErr = False
```

Definition at line 20 of file J_Transp_server.py.

6.11.2.4 readyJ

```
bool J_Transp_server.readyJ = False
```

Definition at line 21 of file J_Transp_server.py.

6.11.2.5 rel_path

```
string J_Transp_server.rel_path = "./Output/Output.txt"
```

Definition at line 30 of file J_Transp_server.py.

6.11.2.6 script_dir

```
J_Transp_server.script_dir = os.path.dirname(__file__)
```

Definition at line 29 of file J_Transp_server.py.

6.11.2.7 wd

```
list J_Transp_server.wd = [50,50,50,0.5,0.5,0.5]
```

Definition at line 25 of file J_Transp_server.py.

6.11.2.8 wp

```
list J_Transp_server.wp = [50, 50, 50, 1, 1, 1]
```

Definition at line 24 of file J_Transp_server.py.

6.12 J_Transp_server_mod Namespace Reference

Functions

- def j_transp (err, err_dot, J, Wp, Wd, delta_t, Jp, Ji, alpha, beta, q_dot_pref, min, max)

  Function that performs Inverse kinematics using the Jacobian Transpose approach, considering also the velocity error and the enhancement for singularity escaping (from "Advances in Robot Kinematics: Analysis and Control", J.Lenarcic and M.
- def handle_IK_Jtransp (req)
- def error_callback (message)
- def jacobian_callback (message)
- def axes_callback (message)
- def JT_server ()

Variables

- bool readyErr = False
- bool readyJ = False
- bool readyJp = False
- list wp = [10, 10, 10, 1, 1, 1]
- list wd = [0,0,0,0,0,0]
- q_dot_elbow = np.array([[0, -1, 0, 10, 0, -0.0001, 0]], dtype=float).T
- int ALPHA = 250
- int BETA = 250
- int MIN = -1
- int MAX = 1
- script_dir = os.path.dirname(__file__)
- string rel_path = "./Output_Jtranspose.txt"
- abs_file_path = os.path.join(script_dir, rel_path)
- int LOG_FLAG = 0

6.12.1 Function Documentation

6.12.1.1 axes_callback()

```
def J_Transp_server_mod.axes_callback (
              message )
```

Definition at line 178 of file J_Transp_server_mod.py.

```
178 def axes_callback(message):
179 
180     # Declaration to work with global variables
181     global Jp, Ji, readyJp
182 
183     # Inter-joints axes and rotation axes used in the correction term
184     Jp = np.array(message.data[:18]).reshape(6,3)
185     Ji = np.array(message.data[18:]).reshape(3,3)
186 
187     # Set the Jp as available
188     readyJp = True
189 
190 # Main body containing 2 Subscribers and the Service defition
```

6.12.1.2 error_callback()

```
def J_Transp_server_mod.error_callback (
              message )
```

Definition at line 147 of file J_Transp_server_mod.py.

```
147 def error_callback(message):
148 
149     # Declaration to work with global variables
150     global error,readyErr
151 
152     # Re-arranging the error (position and velocity) vector in the needed form:
153     err_orient = np.array([message.data[:3]]).T
154     err_pos = np.array([message.data[3:6]]).T
155 
156     err_vel_lin = np.array([message.data[6:9]]).T
157     err_vel_rot = np.array([[0,0,0]]).T
158 
159     error = np.concatenate((err_pos, err_orient, err_vel_lin, err_vel_rot), axis=0)
160 
161     # Set the Error as available
162     readyErr = True
163 
164 # Callback Function for the Jacobian matrix
```

6.12.1.3 handle_IK_Jtransp()

```
def J_Transp_server_mod.handle_IK_Jtransp (
              req )
```

Definition at line 120 of file J_Transp_server_mod.py.

```
120 def handle_IK_Jtransp(req):
121 
122     # Declaration to work with global variables
123     global readyErr, readyJ, readyJp, Jp, Ji, q_dot_elbow, wp, wd
124 
125     print"Server J Transpose accepted request.\n"
126 
127     # Case in which the necessary data is not available
128     if (not (readyErr and readyJ and readyJp)):
129         readyErr = readyJ = readyJp = False
130         rospy.logerr("J Transpose service could not run: missing data.")
131         return
132     # Case in which data is available
133     else:
134         readyErr = readyJ = readyJp = False
```

```
        if (LOG_FLAG):
            now = rospy.get_rostime()
            timestamp =  now.secs + float(now.nsecs)/1000000000
            with open(abs_file_path, 'a') as f:
                f.write(str(timestamp))
                f.write("\t")
                f.write(str(j_transp(error[:6], error[6:], J, wp, wd, 0.01, Jp, Ji, ALPHA, BETA,
       q_dot_elbow, MIN, MAX).velocity.T))
                f.write("\n\n")

        return IK_JtraResponse(j_transp(error[:6], error[6:], J, wp, wd, 0.01, Jp, Ji, ALPHA, BETA,
       q_dot_elbow,MIN, MAX))

# Callback Function for the error on the position (error on Xee)
```

6.12.1.4 j_transp()

```
def J_Transp_server_mod.j_transp (
                err,
                err_dot,
                J,
                Wp,
                Wd,
                delta_t,
                Jp,
                Ji,
                alpha,
                beta,
                q_dot_pref,
                min,
                max )
```

Function that performs Inverse kinematics using the Jacobian Transpose approach, considering also the velocity error and the enhancement for singularity escaping (from "Advances in Robot Kinematics: Analysis and Control", J.Lenarcic and M.

L. Husty).

Parameters

err	the Error on the position and orientation of the end effector.
J	the Jacobian matrix of the manipulator.
delta_t	sampling time.
Wp	weights for the error on the position and orientation.
Wd	weights for the error on the velocity.
Jp	matrix with inter-joints axes (by rows) used in the enhancement.
Ji	matrix with rotation axes (by rows) used in the enhancement.
alpha	tuning parameter (scalar) used in the enhancement.
beta	tuning parameter (scalar) used in the enhancement.
q_dot_pref	joint velocities to escape from singularity.
min	minimum value for the output saturation.
max	maximum value for the output saturation.

Returns

: a Float64MultiArray containing the Joint Velocities.

Definition at line 51 of file J_Transp_server_mod.py.

```
51 def j_transp(err, err_dot, J, Wp, Wd, delta_t, Jp, Ji, alpha, beta, q_dot_pref, min, max):
52     """!
53     Function that performs Inverse kinematics using the Jacobian Transpose
54     approach, considering also the velocity error and the enhancement for
55     singularity escaping (from "Advances in Robot Kinematics: Analysis and
56     Control", J.Lenarcic and M. L. Husty).
57     @param err: the Error on the position and orientation of the end effector.
58     @param J: the Jacobian matrix of the manipulator.
59     @param delta_t: sampling time.
60     @param Wp: weights for the error on the position and orientation.
61     @param Wd: weights for the error on the velocity.
62     @param Jp: matrix with inter-joints axes (by rows) used in the enhancement.
63     @param Ji: matrix with rotation axes (by rows) used in the enhancement.
64     @parameter alpha: tuning parameter (scalar) used in the enhancement.
65     @param beta: tuning parameter (scalar) used in the enhancement.
66     @param q_dot_pref: joint velocities to escape from singularity.
67     @param min: minimum value for the output saturation.
68     @param max: maximum value for the output saturation.
69     @return: a Float64MultiArray containing the Joint Velocities.
70     """
71     # Norm of the position error
72     err_norm = np.linalg.norm(err)
73
74     # Norm of the linear part of the position error
75     err_norm_lin = np.linalg.norm(err[:3])
76
77     # Norm of the rotational part of the position error
78     err_norm_rot = np.linalg.norm(err[3:])
79
80     # Norm of the derivative of the position error
81     err_dot_norm = np.linalg.norm(err_dot)
82
83     # Norm of the linear part of the derivative of the position error
84     err_dot_lin = np.linalg.norm(err_dot[:3])
85
86     # Norm of the rotational part of the derivative of the position error
87     err_dot_rot = np.linalg.norm(err_dot[3:])
88
89     # q_dot initialization
90     q_dot = JointState()
91
92     # Working Case
93     if ((err_norm>0.0001) or (err_dot_norm>0.001)):
94
95         # Weights as diagonal Matrices:
96         Kp = np.diag([err_norm_lin*Wp[0], err_norm_lin*Wp[1], err_norm_lin*Wp[2], err_norm_rot*Wp[3],
       err_norm_rot*Wp[4], err_norm_rot*Wp[5]])
97
98         Kd = np.diag([err_dot_lin*Wd[0], err_dot_lin*Wd[1], err_dot_lin*Wd[2], err_dot_rot*Wd[3],
       err_dot_rot*Wd[4], err_dot_rot*Wd[5]])
99
100         # Delta Joint positions using: J_transpose*K*error_transpose + correction term (Paper Formula)
101         dq = J.T.dot(Kp.dot(err)+Kd.dot(err_dot))+JT_enhance(Jp, Ji, err[:3], alpha, beta, q_dot_pref)
102
103         # Joint velocities, being dq = q_dot*delta_t
104         q_dot.velocity = dq/delta_t
105
106         # Saturation of the output within the desired interval (min,max)
107         for i in range(len(q_dot.velocity)):
108             q_dot.velocity[i] = sat(q_dot.velocity[i],min,max)
109
110     # Position Reached (within the indicated precision)
111     else:
112
113         # Stop the Joints: zero velocities
114         q_dot.velocity = np.zeros((max(J.shape))).T
115
116     return q_dot
117
118
119 # Handler for the Server
```

6.12.1.5 jacobian_callback()

```
def J_Transp_server_mod.jacobian_callback (
            message )
```

Definition at line 165 of file J_Transp_server_mod.py.

```
165 def jacobian_callback(message):
166
167     # Declaration to work with global variables
168     global J, readyJ
169
170     # Baxter Jacobian Matrix
171     J = np.array(message.data)
172     J = J.reshape(6,7)
173
174     # Set the J as available
175     readyJ = True
176
177 # Callback Function for the Axes required for the Correction Term
```

6.12.1.6 JT_server()

```
def J_Transp_server_mod.JT_server ( )
```

Definition at line 191 of file J_Transp_server_mod.py.

```
191 def JT_server():
192
193     # Node Initialization
194     rospy.init_node('IK_Jtransp_server')
195
196     rospy.loginfo("Server Initialized.\n")
197
198     # Subscribers
199     rospy.Subscriber("errors", Float64MultiArray, error_callback)
200     rospy.Subscriber("jacobian", Float64MultiArray, jacobian_callback)
201     rospy.Subscriber("axes", Float64MultiArray, axes_callback)
202
203     s = rospy.Service('IK_Jtransp', IK_Jtra, handle_IK_Jtransp)
204     rospy.spin()
205
```

6.12.2 Variable Documentation

6.12.2.1 abs_file_path

```
J_Transp_server_mod.abs_file_path = os.path.join(script_dir, rel_path)
```

Definition at line 46 of file J_Transp_server_mod.py.

6.12.2.2 ALPHA

```
int J_Transp_server_mod.ALPHA = 250
```

Definition at line 35 of file J_Transp_server_mod.py.

6.12.2.3 BETA

```
int J_Transp_server_mod.BETA = 250
```

Definition at line 36 of file J_Transp_server_mod.py.

6.12.2.4 LOG_FLAG

```
int J_Transp_server_mod.LOG_FLAG = 0
```

Definition at line 49 of file J_Transp_server_mod.py.

6.12.2.5 MAX

```
int J_Transp_server_mod.MAX = 1
```

Definition at line 40 of file J_Transp_server_mod.py.

6.12.2.6 MIN

```
int J_Transp_server_mod.MIN = -1
```

Definition at line 39 of file J_Transp_server_mod.py.

6.12.2.7 q_dot_elbow

```
J_Transp_server_mod.q_dot_elbow = np.array([[0, -1, 0, 10, 0, -0.0001, 0]], dtype=float).T
```

Definition at line 32 of file J_Transp_server_mod.py.

6.12.2.8 readyErr

```
bool J_Transp_server_mod.readyErr = False
```

Definition at line 23 of file J_Transp_server_mod.py.

6.12.2.9 readyJ

```
bool J_Transp_server_mod.readyJ = False
```

Definition at line 24 of file J_Transp_server_mod.py.

6.12.2.10 readyJp

```
bool J_Transp_server_mod.readyJp = False
```

Definition at line 25 of file J_Transp_server_mod.py.

6.12.2.11 rel_path

```
string J_Transp_server_mod.rel_path = "./Output_Jtranspose.txt"
```

Definition at line 45 of file J_Transp_server_mod.py.

6.12.2.12 script_dir

```
J_Transp_server_mod.script_dir = os.path.dirname(__file__)
```

Definition at line 44 of file J_Transp_server_mod.py.

6.12.2.13 wd

```
list J_Transp_server_mod.wd = [0,0,0,0,0,0]
```

Definition at line 29 of file J_Transp_server_mod.py.

6.12.2.14 wp

```
list J_Transp_server_mod.wp = [10, 10, 10, 1, 1, 1]
```

Definition at line 28 of file J_Transp_server_mod.py.

6.13 jac_mat Namespace Reference

Functions

- def sat (x, xmin, xmax)

  Saturates the input value in the given interval.
- def bell (s)

  Creates a bell shaped function to regularize singular values.
- def regularized_pseudoinverse (J)

  Computes the regularized pseudo inverse of mxn matrix J by applying svd decomposition and multiplication for bell shaped function.
- def calculations_6 (q_coppelia)

  Builds the 6 dof Jacobian by computing the transformation matrices, extracting the geometric vectors from the results and finally building the matrix.
- def jacobian_callback (data)

  Callback Function for the Joints Positions.
- def error_callback (message)

  Callback Function for the error on the position of the ee.
- def vel_callback (message)

  Callback Function for the linear and angular velocities of the ee.
- def handle_IK_JAnalytic (req)

  Handler for the Server: computes the ee velocity Vee and the joint velocities vector q_dot Vee is computed as: Vee = vel + K∗error q_dot is computed as: q_dot = J# ∗ Vee then each value of q_dot is saturated in a pre-determined interval.
- def jac_mat ()

Variables

- bool readyErr = False
- bool readyJ = False
- bool readyVel = False

6.13.1 Function Documentation

6.13.1.1 bell()

```
def jac_mat.bell (
              s )
```

Creates a bell shaped function to regularize singular values.

Parameters

s	singular value

Returns

p: regularized singular value

Definition at line 45 of file jac_mat.py.

```
45 def bell(s):
46     """!
47     Creates a bell shaped function to regularize singular values
48     @param s: singular value
49     @return p: regularized singular value
50     """
51 
52     b = -np.log(0.5)/0.00001
53     p = np.exp(-b*s*s)
54 
55     return p
56 
57 
```

6.13.1.2 calculations_6()

```
def jac_mat.calculations_6 (
              q_coppelia )
```

Builds the 6 dof Jacobian by computing the transformation matrices, extracting the geometric vectors from the results and finally building the matrix.

Parameters

q_coppelia	vector of the joint positions

Returns

Js: Jacobian matrix

Definition at line 87 of file jac_mat.py.

```
87 def calculations_6(q_coppelia):
88     """!
89     Builds the 6 dof Jacobian by computing the transformation matrices,
90     extracting the geometric vectors from the results and finally building the matrix.
91     @param q_coppelia: vector of the joint positions
92     @return Js: Jacobian matrix
93     """
94 
95     p = np.pi
96     n_joints = 6
97 
98     # Links lengths [m]
99     L0 = 270.35/1000
100     L1 = 69.00/1000
101     L2 = 364.35/1000
102     L3 = 69.00/1000
103     Lh = math.sqrt(L2 ** 2 + L3 ** 2)
104     L4 = 374.29/1000
105     L5 = 0
106     L6 = 368.30/1000
107 
108     # DH table of Baxter: alpha(i-1), a(i-1), d(i), theta(i)
109     DH = np.array([[0, 0, L0, 0],
110                    [-p/2, L1, 0, 0],
111                    [0, Lh, 0, p/2],
112                    [p/2, 0, L4, 0],
113                    [-p/2, L5, 0, 0],
114                    [p/2, 0, 0, 0],
115                    [0, 0, L6, 0]])
116 
117     # Transformation matrices given DH table. T0,1 T1,2 ... T7,e
```

```
    T_rel_ini = t.DH_to_T(DH)

    # Type of joints, 1 = revolute, 0 = prismatic
    info = np.array([1, 1, 1, 1, 1, 1])

    # Initial q
    q = np.array([0, 0, 0, 0, 0, 0])

    # Transformation matrices given the configuration
    T_trans = t.transformations(T_rel_ini, q_coppelia, info)

    # T0,1 T0,2 T0,3...T0,e
    T_abs = t.abs_trans(T_trans)

    # Extract geometric vectors needed for computations
    geom_v = j.geometric_vectors(T_abs)

    np.set_printoptions(precision=4)
    np.set_printoptions(suppress=True)

    # axis of rotation of the revolute joins projected on zero
    k = geom_v[0]
    # distances end_effector-joints projected on zero
    r = geom_v[1]

    Js = j.jacob(k, r, n_joints, info)

    return Js
```

6.13.1.3 error_callback()

```
def jac_mat.error_callback (
                message )
```

Callback Function for the error on the position of the ee.

Definition at line 171 of file jac_mat.py.

```
171 def error_callback(message):
172     """!
173    Callback Function for the error on the position of the ee
174     """
175
176     global error, readyErr
177     err_orient = np.array([message.data[:3]]).T
178     err_pos = np.array([message.data[3:6]]).T
179     error = np.concatenate((err_pos, err_orient), axis=0)
180
181     rospy.loginfo("Received Position Error:\n%s\n", str(error))
182
183     # Set the Error as available
184     readyErr = True
185
186
```

6.13.1.4 handle_IK_JAnalytic()

```
def jac_mat.handle_IK_JAnalytic (
                req )
```

Handler for the Server: computes the ee velocity Vee and the joint velocities vector q_dot Vee is computed as: Vee = vel + K∗error q_dot is computed as: q_dot = J# ∗ Vee then each value of q_dot is saturated in a pre-determined interval.

Parameters

	request for service

Returns

: Jacobian and ee velocity are sent as service

Definition at line 201 of file jac_mat.py.

```
201 def handle_IK_JAnalytic(req):
202     """!
203     Handler for the Server: computes the ee velocity Vee and the joint velocities vector q_dot
204         Vee is computed as: Vee = vel + K*error
205         q_dot is computed as: q_dot = J# * Vee
206     then each value of q_dot is saturated in a pre-determined interval.
207     @param: request for service
208     @return: Jacobian and ee velocity are sent as service
209     """
210 
211     # Declaration to work with global variables
212     global readyErr, readyJ, readyVel
213 
214     print("Server Analytic accepted request\n")
215 
216     if (not (readyErr and readyJ and readyVel)):
217         readyErr = readyJ = readyVel = False
218         rospy.logerr("Analytic J_6 service could not run: missing data.")
219         return
220 
221     else:
222         readyErr = readyJ = readyVel = False
223 
224         # q_dot initialization
225         q_dot = JointState()
226 
227         # Gain for the Control Law
228         K = 10
229 
230         # Inverse of the 6dof jacobian
231         J_inv = regularized_pseudoinverse(J_6)
232 
233         # Ee velocity
234         vee = vel+K*error
235 
236         # q_dot computation
237         q_dot_6 = J_inv.dot(vee)
238 
239         # q_dot saturation
240         for i in range(6):
241             q_dot_6[i] = sat(q_dot_6[i], -1, 1)
242 
243         # Since the third Joint is blocked, its velocity must be set to 0
244         q_dot.velocity = np.insert(q_dot_6, 2, 0)
245 
246         return IK_JTAResponse(q_dot)
247 
248 
```

6.13.1.5 jac_mat()

```
def jac_mat.jac_mat ( )
```

```
This ROS node computes the 6dof analytic jacobian.
It subscribes to the following topics: errors, tracking, logtopic.
It also works as a server for the IK_JTA service, on which it sends
the q_dot and the end effector velocity computed by the algorithm.
```

Definition at line 249 of file jac_mat.py.

```
def jac_mat():
    """
    This ROS node computes the 6dof analytic jacobian.
    It subscribes to the following topics: errors, tracking, logtopic.
    It also works as a server for the IK_JTA service, on which it sends
    the q_dot and the end effector velocity computed by the algorithm.
    """

    # Init node
    rospy.init_node('jac_mat', anonymous=True)

    rospy.loginfo("Server Analytic Initialized\n")

    # Subscribe for error positions
    rospy.Subscriber("errors", Float64MultiArray, error_callback)

    # Subscribe for ee velocity
    rospy.Subscriber("tracking", Float64MultiArray, vel_callback)

    # Subscribe for joint positions
    rospy.Subscriber("logtopic", JointState, jacobian_callback)

    # Service Definition
    s_vel = rospy.Service('IK_JAnalytic', IK_JTA, handle_IK_JAnalytic)

    rospy.spin()


```

6.13.1.6 jacobian_callback()

```
def jac_mat.jacobian_callback (
                data )
```

Callback Function for the Joints Positions.

Definition at line 148 of file jac_mat.py.

```
def jacobian_callback(data):
    """!
    Callback Function for the Joints Positions
    """

    global J_6, readyJ

    q_coppelia = np.array(data.position)

    # It assumes one joint to be fixed (3rd in this case), in order to pass from 7 to 6
    q_coppelia = np.delete(q_coppelia, 2, 0)

    rospy.loginfo("Joint positions: %s\n", str(q_coppelia))

    # Compute the matrix
    J_6 = calculations_6(q_coppelia)

    rospy.loginfo("Jacobian:\n%s\n", J_6)

    # Set the Jacobian as available
    readyJ = True


```

6.13.1.7 regularized_pseudoinverse()

```
def jac_mat.regularized_pseudoinverse (
                J )
```

Computes the regularized pseudo inverse of mxn matrix J by applying svd decomposition and multiplication for bell shaped function.

Parameters

J	nxn matrix

Returns

Jx: regularized pseudo inverse of the matrix

Definition at line 58 of file jac_mat.py.

```
58 def regularized_pseudoinverse(J):
59     """!
60     Computes the regularized pseudo inverse of mxn matrix J
61     by applying svd decomposition and multiplication for bell shaped function
62     @param J: nxn matrix
63     @return Jx: regularized pseudo inverse of the matrix
64     """
65
66     rows = len(J)
67     cols = len(J[0])
68
69     # Svd decomposition of the input matrix
70     U, s, Vt = np.linalg.svd(J)
71     Sx = np.zeros((cols, rows))
72
73     for i in range(cols):
74         for j in range(rows):
75             if i == j:
76                 Sx[i][j] = s[i]/(s[i]**2 + bell(s[i])**2)
77
78     Ut = U.transpose()
79     V = Vt.transpose()
80
81     # Reconstruct the pseudo inverse of the matrix with the new regularized values
82     Jx = V.dot(Sx.dot(Ut))
83
84     return Jx
85
86
```

6.13.1.8 sat()

```
def jac_mat.sat (
                x,
                xmin,
                xmax )
```

Saturates the input value in the given interval.

Parameters

x	value to saturate
xmin	minimum value of the interval
xmax	maximum value of the interval

Returns

: saturated value

Definition at line 30 of file jac_mat.py.

```
30 def sat(x, xmin, xmax):
```

```
    """!
    Saturates the input value in the given interval
    @param x: value to saturate
    @param xmin: minimum value of the interval
    @param xmax: maximum value of the interval
    @return: saturated value
    """
    if x > xmax:
        return xmax
    if x < xmin:
        return xmin
    return x
```

6.13.1.9 vel_callback()

```
def jac_mat.vel_callback (
                message )
```

Callback Function for the linear and angular velocities of the ee.

Definition at line 187 of file jac_mat.py.

```
def vel_callback(message):
    """!
    Callback Function for the linear and angular velocities of the ee
    """

    global vel, readyVel
    vel = np.array([message.data[:6]]).T

    rospy.loginfo("Received Velocities End Effector:\n%s\n", str(vel))

    # Set the Vel as available
    readyVel = True
```

6.13.2 Variable Documentation

6.13.2.1 readyErr

```
bool jac_mat.readyErr = False
```

Definition at line 25 of file jac_mat.py.

6.13.2.2 readyJ

```
bool jac_mat.readyJ = False
```

Definition at line 26 of file jac_mat.py.

6.13.2.3 readyVel

```
bool jac_mat.readyVel = False
```

Definition at line 27 of file jac_mat.py.

6.14 JT_enhance Namespace Reference

Functions

- def JT_enhance (Jp, Ji, err_tran, alpha, beta, q_dot_pref)
  Function that computes a Correction Term to be added to the Jacobian Transpose's formula, in order to escape from singularity (from "Advances in Robot Kinematics: Analysis and Control", J.Lenarcic and M.

6.14.1 Detailed Description

```
@author: dipen
```

6.14.2 Function Documentation

6.14.2.1 JT_enhance()

```
def JT_enhance.JT_enhance (
              Jp,
              Ji,
              err_tran,
              alpha,
              beta,
              q_dot_pref )
```

Function that computes a Correction Term to be added to the Jacobian Transpose's formula, in order to escape from singularity (from "Advances in Robot Kinematics: Analysis and Control", J.Lenarcic and M.

L. Husty).

Parameters

Jp	matrix with inter-joints axes (by rows) used to detect singularity.
Ji	matrix with rotation axes (by rows) used to detect singularity.
err_tran	position error vector.
alpha	tuning parameter (scalar).
beta	tuning parameter (scalar).
q_dot_pref	joint velocities to escape from singularity (elbow configuration), expected [7x1].

Returns

: a numpy array containing the Joint Velocities for the correction term [7x1].

Definition at line 8 of file JT_enhance.py.

```
8 def JT_enhance(Jp, Ji, err_tran, alpha, beta, q_dot_pref):
9     """!
10     Function that computes a Correction Term to be added to the Jacobian Transpose’s
11     formula, in order to escape from singularity (from "Advances in Robot Kinematics:
12     Analysis and Control", J.Lenarcic and M. L. Husty).
13     @param Jp: matrix with inter-joints axes (by rows) used to detect singularity.
14     @param Ji: matrix with rotation axes (by rows) used to detect singularity.
15     @param err_tran: position error vector.
16     @param alpha: tuning parameter (scalar).
17     @param beta: tuning parameter (scalar).
18     @param q_dot_pref: joint velocities to escape from singularity (elbow configuration), expected [7x1].
19     @return: a numpy array containing the Joint Velocities for the correction term [7x1].
20     """
21 
22     # Checking on dimensions of Jp and Ji
23     if (len(Jp)/2!=len(Ji)):
24         print("Wrong Dimensions for Jp and Ji: it must be Jp_rows = 2*Ji_rows\n")
25         return
26 
27     # Initialization of the Normalized Vectors Matrix (Jp)
28     Jp_norm = np.zeros(Jp.shape)
29 
30     # Initialization of the Normalized Vectors Matrix (Ji)
31     Ji_norm = np.zeros(Ji.shape)
32 
33     # Initialization of the Mu Vector (one elementfor each pair)
34     Mu = np.zeros((int(len(Jp)/2),1))
35 
36     # Initialization of the Gamma Vector
37     Gamma = np.zeros((len(Ji),1))
38 
39     # Initialization of f Vector
40     f = np.zeros((int(len(Jp)/2),1))
41 
42     # Initialization of the g Vector
43     g = np.zeros(q_dot_pref.shape)
44 
45     # Indices Initialization
46     k = 0
47     j = 0
48 
49     # Normalization of the Jp Vectors
50     for i in range(len(Jp)):
51         Jp_norm[i,:] = Jp[i,:]/np.linalg.norm(Jp[i,:])
52 
53     # Normalization of the Ji Vectors
54     for i in range(len(Ji)):
55         Ji_norm[i,:] = Ji[i,:]/np.linalg.norm(Ji[i,:])
56 
57     # Mu Computation
58     while (k<len(Jp)):
59         Mu[j,:] = -np.dot(Jp_norm[k,:],Jp_norm[k+1,:])*np.dot(Jp_norm[k,:],err_tran)
60         k = k+2
61         j= j+1
62 
63     # f computation (to consider only worse singularity situations)
64     f = alpha*np.power(Mu,4)
65 
66     # Gamma Computation
67     for i in range(len(Ji_norm)):
68         Gamma[i,:] = np.linalg.norm(np.cross(Ji_norm[i,:],err_tran.T))
69 
70     # Temporary Vector to be element-wise with the other terms: only 3 joints
71     # are considered, since only responsible ones for escaping the singularity
72     temp = np.array([[1, Gamma[0]*f[0], 1, Gamma[1]*f[1], 1, Gamma[2]*f[2], 1]],dtype = float).T
73 
74     # g Enhancing Term Computation (element-wise product)
75     g = np.multiply(temp, q_dot_pref*beta*np.linalg.norm(err_tran))
76 
77     return g
78 
```

6.15 offlineAnalysis Namespace Reference

Functions

- def plotData (df, subplt, x_axis, y_lable, titlePlot)

This function simply plots data into a a graph.

Variables

- script_dir = os.path.dirname(__file__)
- string rel_path1 = "../output/lin_acc.csv"
- abs_file_path1 = os.path.join(script_dir, rel_path1)
- string rel_path2 = "../output/orientation.csv"
- abs_file_path2 = os.path.join(script_dir, rel_path2)
- string rel_path3 = "../output/angVel.csv"
- abs_file_path3 = os.path.join(script_dir, rel_path3)
- dictionary font
- df_linacc
- df_rot
- df_angVel
- t = len(df_linacc.X)
- x_axis = np.arange(start=1, stop=t + 1, step=1)
- figsize
- temp = df_linacc.astype(float)

6.15.1 Detailed Description

```
Documentation for the offline analysis tool
This piece of code is mostly used to run tests relative to the incoming imu data
and filtering/transformation performed in 'remove_gravity.py'
```

6.15.2 Function Documentation

6.15.2.1 plotData()

```
def offlineAnalysis.plotData (
              df,
              subplt,
              x_axis,
              y_lable,
              titlePlot )
```

This function simply plots data into a a graph.

Parameters

df	specifies the data set (orientation/linear acceleration/angular velocity) to be plotted
subplt	this specifies the position of the plot in the figure
x_axis	ascending numbers (i.e. data samples)
y_lable	data type and measurement unit
titlePlot	title of the plot

Definition at line 29 of file offlineAnalysis.py.

```
29 def plotData(df, subplt, x_axis, y_lable, titlePlot):
30     """!
31     This function simply plots data into a a graph
32     @param df specifies the data set (orientation/linear acceleration/angular velocity) to be plotted
33     @param subplt this specifies the position of the plot in the figure
34     @param x_axis ascending numbers (i.e. data samples)
35     @param y_lable data type and measurement unit
36     @param titlePlot title of the plot
37     """
38     plt.subplot(subplt)
39     plt.plot(x_axis, df.X)
40     plt.plot(x_axis, df.Y)
41     plt.plot(x_axis, df.Z)
42     plt.xlabel('#samples', fontdict=font)
43     plt.ylabel(y_lable, fontdict=font)
44     plt.title(titlePlot, fontdict=font)
45     plt.legend()
46
47
```

6.15.3 Variable Documentation

6.15.3.1 abs_file_path1

```
offlineAnalysis.abs_file_path1 = os.path.join(script_dir, rel_path1)
```

Definition at line 16 of file offlineAnalysis.py.

6.15.3.2 abs_file_path2

```
offlineAnalysis.abs_file_path2 = os.path.join(script_dir, rel_path2)
```

Definition at line 18 of file offlineAnalysis.py.

6.15.3.3 abs_file_path3

```
offlineAnalysis.abs_file_path3 = os.path.join(script_dir, rel_path3)
```

Definition at line 20 of file offlineAnalysis.py.

6.15.3.4 df_angVel

```
offlineAnalysis.df_angVel
```

Initial value:

```
=  pd.read_csv(abs_file_path3, names=[
                              'X', 'Y', 'Z'], header=0, decimal=',')
```

Definition at line 57 of file offlineAnalysis.py.

6.15.3.5 df_linacc

```
offlineAnalysis.df_linacc
```

Initial value:

```
=  pd.read_csv(abs_file_path1, names=[
                              ’X’, ’Y’, ’Z’], header=0, decimal=’,’)
```

Definition at line 53 of file offlineAnalysis.py.

6.15.3.6 df_rot

```
offlineAnalysis.df_rot
```

Initial value:

```
=  pd.read_csv(abs_file_path2, names=[
                           ’X’, ’Y’, ’Z’], header=0, decimal=’,’)
```

Definition at line 55 of file offlineAnalysis.py.

6.15.3.7 figsize

```
offlineAnalysis.figsize
```

Definition at line 65 of file offlineAnalysis.py.

6.15.3.8 font

```
dictionary offlineAnalysis.font
```

Initial value:

```
=  {’family’: ’serif’,
        ’color’: ’darkred’,
        ’weight’: ’normal’,
        ’size’: 12,
        }
```

Definition at line 22 of file offlineAnalysis.py.

6.15.3.9 rel_path1

```
string offlineAnalysis.rel_path1 = "../output/lin_acc.csv"
```

Definition at line 15 of file offlineAnalysis.py.

6.15.3.10 rel_path2

```
string offlineAnalysis.rel_path2 = "../output/orientation.csv"
```

Definition at line 17 of file offlineAnalysis.py.

6.15.3.11 rel_path3

```
string offlineAnalysis.rel_path3 = "../output/angVel.csv"
```

Definition at line 19 of file offlineAnalysis.py.

6.15.3.12 script_dir

```
offlineAnalysis.script_dir = os.path.dirname(__file__)
```

Definition at line 14 of file offlineAnalysis.py.

6.15.3.13 t

```
offlineAnalysis.t = len(df_linacc.X)
```

Definition at line 61 of file offlineAnalysis.py.

6.15.3.14 temp

```
offlineAnalysis.temp = df_linacc.astype(float)
```

Definition at line 66 of file offlineAnalysis.py.

6.15.3.15 x_axis

```
offlineAnalysis.x_axis = np.arange(start=1, stop=t + 1, step=1)
```

Definition at line 62 of file offlineAnalysis.py.

6.16 removeGravity Namespace Reference

Functions

- def removeGravity (lin_acc, Rot_m, g)

6.16.1 Detailed Description

```
This piece of code carries out the gravity removal
@param lin_acc linear acceleration data incoming from the accelerometer
@param Rot_m rotation matrix computed with eulerAnglesToRotationMatrix()
@param g gravity vector
@returns the output is the same linear acceleration vector provided, but without gravity influence
```

6.16.2 Function Documentation

6.16.2.1 removeGravity()

```
def removeGravity.removeGravity (
            lin_acc,
            Rot_m,
            g )
```

Definition at line 12 of file removeGravity.py.

```
12 def removeGravity(lin_acc, Rot_m, g):
13
14     # rotate g vector in the current frame
15     g_frame_i = np.dot(Rot_m, g)
16     g_removed = [0, 0, 0]  # define linear acceleration without gravity
17
18     #check the sign of the current linear acceleration, and remove the gravity properly
19     for i in range(0, 3):
20         if lin_acc[i] >= 0:
21             g_removed[i] = lin_acc[i] - abs(g_frame_i[i])
22         if lin_acc[i] < 0:
23             g_removed[i] = lin_acc[i] + abs(g_frame_i[i])
24
25     return g_removed
```

6.17 rotation_matrix_server Namespace Reference

Functions

- def eulerAnglesToRotationMatrix (angles)

  Function that transforms euler angle coordinates into the rotation matrix.
- def anglesCompensate (angles)

  Function used to filter unwanted minimal incoming data fluctuations, due to noise as well as human operator shake.
- def callback (data)

  This is the callback function: it is invoked every time there is incoming data.
- def serv_callback ()

  This is the server callback function.
- def smartphone_server_setup ()

  Setup of the server.

Variables

- float dx = 0.0174
- list angles = [0, 0, 0]

6.17.1 Detailed Description

```
Documentation for the rotation matrix sever
This file waits for rotation matrix transformations requests;
it has been made mostly for convenience of the optimization project subgroup
```

6.17.2 Function Documentation

6.17.2.1 anglesCompensate()

```
def rotation_matrix_server.anglesCompensate (
              angles )
```

Function used to filter unwanted minimal incoming data fluctuations, due to noise as well as human operator shake.

Parameters

angles	orientation with respect to X, Y, Z axes

Returns

returns a filtered version (if necessary) of the input angles

Definition at line 51 of file rotation_matrix_server.py.

```
51 def anglesCompensate(angles):
52     """!
53     Function used to filter unwanted minimal incoming data fluctuations,
54     due to noise as well as human operator shake
55     @param angles orientation with respect to X, Y, Z axes
56     @returns returns a filtered version (if necessary) of the input angles
57     """
58     # reduce sensibility of sensor: minimum precision is dx
59     compensatedAngles = [0, 0, 0]
60
61     for i in range(0, 3):  # i = 0, 1, 2
62         if abs(angles[i] / dx) >= 1:
63             compensatedAngles[i] = angles[i]
64
65     return compensatedAngles
66
67
```

6.17.2.2 callback()

```
def rotation_matrix_server.callback (
              data )
```

This is the callback function: it is invoked every time there is incoming data.

It transforms a quaternion into euler angles, and the invokes anglesCompensate()

Parameters

data	incoming from the imu sensor

Definition at line 68 of file rotation_matrix_server.py.

```
68 def callback(data):
69     """!
70     This is the callback function: it is invoked every time there is incoming data.
71     It transforms a quaternion into euler angles, and the invokes anglesCompensate()
72     @param data incoming from the imu sensor
73     """
74     global angles
75 
76     # get data
77     orientation = [data.orientation.x, data.orientation.y,
78                    data.orientation.z, data.orientation.w]
79 
80     # transform quaternion to euler angles
81     angles = tf.transformations.euler_from_quaternion(orientation, "sxyz")
82 
83     angles = anglesCompensate(angles)
84 
85 
```

6.17.2.3 eulerAnglesToRotationMatrix()

```
def rotation_matrix_server.eulerAnglesToRotationMatrix (
                angles )
```

Function that transforms euler angle coordinates into the rotation matrix.

Parameters

angles	euler angles, i.e. orientation with respect to X, Y, Z axes

Returns

rotation matrix

Definition at line 24 of file rotation_matrix_server.py.

```
24 def eulerAnglesToRotationMatrix(angles):  # angles [roll, pitch, yaw]
25     """!
26     Function that transforms euler angle coordinates into the rotation matrix
27     @param angles euler angles, i.e. orientation with respect to X, Y, Z axes
28     @returns rotation matrix
29     """
30 
31     R_x = np.array([[1,         0,                    0],
32                     [0,         math.cos(angles[0]),   math.sin(angles[0])],
33                     [0,         -math.sin(angles[0]),  math.cos(angles[0])]
34                     ])
35 
36     R_y = np.array([[math.cos(angles[1]),    0,       -math.sin(angles[1])],
37                     [0,                      1,       0],
38                     [math.sin(angles[1]),    0,       math.cos(angles[1])]
39                     ])
40 
41     R_z = np.array([[math.cos(angles[2]),      math.sin(angles[2]),     0],
42                     [-math.sin(angles[2]),     math.cos(angles[2]),     0],
43                     [0,                        0,                       1]
44                     ])
45 
46     R = np.dot(R_x, np.dot(R_y, R_z))
47 
48     return R
49 
50 
```

6.17.2.4 serv_callback()

```
def rotation_matrix_server.serv_callback ( )
```

This is the server callback function.

It simply formats the data coming from the Imu in order to be compatible with the Optimization subgroup's code

Returns

the a formatted version of the rotation matrix

Definition at line 86 of file rotation_matrix_server.py.

```
86 def serv_callback():
87     """!
88     This is the server callback function. It simply formats the data coming from the Imu in order
89     to be compatible with the Optimization subgroup’s code
90     @returns the a formatted version of the rotation matrix
91     """
92
93     serv_rot_matrix = Float64MultiArray()
94     serv_rot_matrix.layout.dim.append(MultiArrayDimension())
95     serv_rot_matrix.layout.dim.append(MultiArrayDimension())
96     serv_rot_matrix.layout.dim[0].label = "rows"
97     serv_rot_matrix.layout.dim[0].size = 3
98     serv_rot_matrix.layout.dim[1].label = "columns"
99     serv_rot_matrix.layout.dim[1].size = 3
100     serv_rot_matrix.data = eulerAnglesToRotationMatrix(angles)
101
102     return SmartphoneResponse(serv_rot_matrix)
103
104
```

6.17.2.5 smartphone_server_setup()

```
def rotation_matrix_server.smartphone_server_setup ( )
```

Setup of the server.

Inside this function there is the 'smartphone_service' node initialization and the subscription to the 'android/imu' topic, from which it receives incoming data of smarthpone/smartwatch accelerometers. Finally, the rotation matrix server is activated

Definition at line 105 of file rotation_matrix_server.py.

```
105 def smartphone_server_setup():
106     """!
107     Setup of the server. Inside this function there is the ’smartphone_service’ node initialization
108     and the subscription to the ’android/imu’ topic, from which it receives incoming data
109     of smarthpone/smartwatch accelerometers. Finally, the rotation matrix server is activated
110     """
111
112     # In ROS, nodes are uniquely named. If two nodes with the same
113     # name are launched, the previous one is kicked off. The
114     # anonymous=True flag means that rospy will choose a unique
115     # name for our ’smartphone_service’ node, so that multiple nodes can
116     # run simultaneously without any issue.
117     rospy.init_node(’smartphone_service’, anonymous=True)  # initialize node
118
119     # This declares that your node subscribes to the android/imu topic,
120     # which is of type sensor_msgs.msg.Imu. When new data is received,
121     # callback is invoked with that data as argument.
122     rospy.Subscriber("android/imu", Imu, callback)
123
124     # activate service
125     serv = rospy.Service(’smartphone_serv’, Smartphone, serv_callback)
126     print ("\nService server setup and running")
127
128     # spin() simply keeps python from exiting until this node is stopped
129     rospy.spin()
130
131
```

6.17.3 Variable Documentation

6.17.3.1 angles

```
list rotation_matrix_server.angles = [0, 0, 0]
```

Definition at line 21 of file rotation_matrix_server.py.

6.17.3.2 dx

```
float rotation_matrix_server.dx = 0.0174
```

Definition at line 19 of file rotation_matrix_server.py.

6.18 rotationMatrix Namespace Reference

Functions

- def eulerAnglesToRotationMatrix (angles)

  Function that transforms euler angle coordinates into the rotation matrix.

6.18.1 Detailed Description

```
Documentation for rotationMatrix.py
This file is consisting in one function only, and its only aim is to provide rotation matrix transformations
```

6.18.2 Function Documentation

6.18.2.1 eulerAnglesToRotationMatrix()

```
def rotationMatrix.eulerAnglesToRotationMatrix (
              angles )
```

Function that transforms euler angle coordinates into the rotation matrix.

Parameters

angles	euler angles, i.e. orientation with respect to X, Y, Z axes

Returns

rotation matrix

Definition at line 12 of file rotationMatrix.py.

```
12 def eulerAnglesToRotationMatrix(angles):  # angles [roll, pitch, yaw]
13     """!
14     Function that transforms euler angle coordinates into the rotation matrix
15     @param angles euler angles, i.e. orientation with respect to X, Y, Z axes
16     @returns rotation matrix
17     """
18 
19     R_x = np.array([[1,         0,                  0],
20                     [0,         math.cos(angles[0]),    math.sin(angles[0])],
21                     [0,         -math.sin(angles[0]),   math.cos(angles[0])]
22                     ])
23 
24     R_y = np.array([[math.cos(angles[1]),    0,      -math.sin(angles[1])],
25                     [0,                      1,      0],
26                     [math.sin(angles[1]),    0,      math.cos(angles[1])]
27                     ])
28 
29     R_z = np.array([[math.cos(angles[2]),      math.sin(angles[2]),     0],
30                     [-math.sin(angles[2]),     math.cos(angles[2]),     0],
31                     [0,                        0,                       1]
32                     ])
33 
34     R = np.dot(R_x, np.dot(R_y, R_z))
35 
36     return R
```

6.19 T_computations Namespace Reference

Functions

- def DH_to_T (DH)

  Computes the transformation matrices given the DH table of the serial link.
- def transformations (T_rel_ini, q, info)

  Computes tranformations given T_relatives, q's and the info.
- def abs_trans (T_rel)

  Computes trasformations matrices w.r.t.

Variables

- p = np.pi
- Tmp = np.dot(T_rel_ini[i], Tel)

  Tel = np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 0, q[i]] [0, 0, 0, 1]])

6.19.1 Function Documentation

6.19.1.1 abs_trans()

```
def T_computations.abs_trans (
              T_rel )
```

Computes trasformations matrices w.r.t.

0 frame.

Parameters

T_rel	trasformation matrices of a joint with respect to previous one.

Returns

T: absolute transformation matrices.

Definition at line 66 of file T_computations.py.

```
66 def abs_trans(T_rel):
67   """!
68   Computes trasformations matrices w.r.t. 0 frame.
69   @param T_rel: trasformation matrices of a joint with respect to previous one.
70   @return T: absolute transformation matrices.
71   """
72   T = []
73   # First is the same.
74   T.append(T_rel[0])
75
76   for i in range(1, len(T_rel)):
77     Tmp = np.dot(T[i-1], T_rel[i])
78     T.append(Tmp)
79
80   return T
```

6.19.1.2 DH_to_T()

```
def T_computations.DH_to_T (
                DH )
```

Computes the transformation matrices given the DH table of the serial link.

Parameters

DH	devavitt-hartemberg parameters.

Returns

T: transformation matrices of a joint with respect to previous joint.

Definition at line 7 of file T_computations.py.

```
7 def DH_to_T(DH):
8   """!
9   Computes the transformation matrices given the DH table of the serial link.
10   @param DH: devavitt-hartemberg parameters.
11   @return T: transformation matrices of a joint with respect to previous joint.
12   """
13   # Get the number of rows, to know how many T matrices should create.
14   rows = len(DH)
15
16   T = []
17
18   for i in range(rows):
19     Tmp = np.array([[np.cos(DH[i,3]), -np.sin(DH[i,3]), 0, DH[i,1]],
20             [np.sin(DH[i,3])*np.cos(DH[i,0]), np.cos(DH[i,3])*np.cos(DH[i,0]), -np.sin(DH[i,0]),
     -DH[i,2]*np.sin(DH[i,0])],
21             [np.sin(DH[i,3])*np.sin(DH[i,0]), np.cos(DH[i,3])*np.sin(DH[i,0]), np.cos(DH[i,0]),
     DH[i,2]*np.cos(DH[i,0])],
22             [0, 0, 0, 1]])
23     T.append(Tmp)
24
25   return T
26
```

6.19.1.3 transformations()

```
def T_computations.transformations (
            T_rel_ini,
            q,
            info )
```

Computes tranformations given T_relatives, q's and the info.

Parameters

T_rel_ini	the ones computed with DH_to_T.
q	current configuration of baxter's arm.
info	1->revolute, 0->prismatic.

Returns

T: transformation matrices of a joint with respect to previous joint in the new configuration.

Definition at line 27 of file T_computations.py.

```
27 def transformations(T_rel_ini, q, info):
28   """!
29   Computes tranformations given T_relatives, q's and the info.
30   @param T_rel_ini: the ones computed with DH_to_T.
31   @param q: current configuration of baxter's arm.
32   @param info: 1->revolute, 0->prismatic.
33   @return T: transformation matrices of a joint with respect to previous joint in
34   the new configuration.
35   """
36   row_q = q.size
37   row_info = info.size
38 
39   T = []
40 
41   if row_q != row_info:
42     print("Warning. q and info must have same size.")
43     return
44 
45   for i in range(row_q):
46     if info[i] == 1:
47       Tel = np.array([[np.cos(q[i]), -np.sin(q[i]), 0 , 0],
48                       [np.sin(q[i]), np.cos(q[i]), 0 , 0],
49                       [0, 0, 1, 0],
50                       [0, 0, 0, 1]])
51   #  else:
52       # Case in which there are prismatic joints.
```

6.19.2 Variable Documentation

6.19.2.1 p

```
T_computations.p = np.pi
```

Definition at line 5 of file T_computations.py.

6.19.2.2 Tmp

```
T_computations.Tmp = np.dot(T_rel_ini[i], Tel)
```

Tel = np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 0, q[i]] [0, 0, 0, 1]])

Definition at line 58 of file T_computations.py.

6.20 test_publisher_halfcircle Namespace Reference

Functions

- def simulate_callback (data)

  Handles changes in the simulation.
- def talker ()

  Publishes mock smartphone signals, read from files.

Variables

- int key = 0
- int reset = 0
- pub = None
- anonymous

6.20.1 Function Documentation

6.20.1.1 simulate_callback()

```
def test_publisher_halfcircle.simulate_callback (
              data )
```

Handles changes in the simulation.

If data = 0, then the initial conditions must be resetted. If data = 1, then the algorithm moves on. If data = 2, then the algorithm pauses.

Parameters

data	coming from coppelia_sim.

Definition at line 15 of file test_publisher_halfcircle.py.

```
15 def simulate_callback(data):
16     """!
17     Handles changes in the simulation. If data = 0, then the initial conditions must be resetted.
18     If data = 1, then the algorithm moves on. If data = 2, then the algorithm pauses.
19     @param data: coming from coppelia_sim.
```

```
    """
    global key,reset
    if data.data == 0:
        key = 0; reset = 1
    elif data.data == 1:
        key = 1; reset = 0
    elif data.data == 2:
        key = 0; reset = 0



```

6.20.1.2 talker()

```
def test_publisher_halfcircle.talker ( )
```

Publishes mock smartphone signals, read from files.

Definition at line 31 of file test_publisher_halfcircle.py.

```
31 def talker():
32     """!
33     Publishes mock smartphone signals, read from files.
34     """
35     global pub,key,reset
36 
37     rate = rospy.Rate(100)
38 
39     # Load reference signals from MATLAB files
40     # Must be run in same folder as such files
41     agmat = scipy.io.loadmat("ag1halfcircle.mat")
42     wgmat = scipy.io.loadmat("wg1halfcircle.mat")
43     quamat = scipy.io.loadmat("quatg1halfcircle.mat")
44     ag = agmat["ag1"]
45     wg = wgmat["wg1"]
46     qua = quamat["quatg1"]
47 
48     K = 0
49     imu = Imu()
50 
51     while True:
52         if key == 1:
53            imu.orientation.w = qua[0,K]
54            imu.orientation.x = qua[1,K]
55            imu.orientation.y = qua[2,K]
56            imu.orientation.z = qua[3,K]
57            imu.linear_acceleration.x = ag[0,K]
58            imu.linear_acceleration.y = ag[1,K]
59            imu.linear_acceleration.z = ag[2,K]
60            imu.angular_velocity.x = wg[0,K]
61            imu.angular_velocity.y = wg[1,K]
62            imu.angular_velocity.z = wg[2,K]
63            if K < 300: K = K + 1 # 300 is the number of samples of the trajectory
64            pub.publish(imu)
65         elif reset == 1: K = 0
66         rate.sleep()
67 
```

6.20.2 Variable Documentation

6.20.2.1 anonymous

```
test_publisher_halfcircle.anonymous
```

Definition at line 70 of file test_publisher_halfcircle.py.

6.20.2.2 key

```
int test_publisher_halfcircle.key = 0
```

Definition at line 10 of file test_publisher_halfcircle.py.

6.20.2.3 pub

```
test_publisher_halfcircle.pub = None
```

Definition at line 12 of file test_publisher_halfcircle.py.

6.20.2.4 reset

```
int test_publisher_halfcircle.reset = 0
```

Definition at line 11 of file test_publisher_halfcircle.py.

6.21 utilities Namespace Reference

Functions

- def init_float64_multiarray (rows, columns)

  Function that initializes a Float64MultiArray of size rows x columns.
- def anglesCompensate (angles)

  Function used to filter unwanted minimal incoming data fluctuations, due to noise as well as human operator shake.
- def eulerAnglesToRotationMatrix (angles)

  Function that transforms euler angle coordinates into the rotation matrix.

6.21.1 Function Documentation

6.21.1.1 anglesCompensate()

```
def utilities.anglesCompensate (
              angles )
```

Function used to filter unwanted minimal incoming data fluctuations, due to noise as well as human operator shake.

Parameters

angles	orientation with respect to X, Y, Z axes

Returns

compensatedAngles: returns a filtered version (if necessary) of the input angles

Definition at line 20 of file utilities.py.

```
def anglesCompensate(angles):
    """!
    Function used to filter unwanted minimal incoming data fluctuations,
    due to noise as well as human operator shake
    @param angles: orientation with respect to X, Y, Z axes
    @returns compensatedAngles: returns a filtered version (if necessary) of the input angles
    """
    dx = 0.0174  # min angle perceived [rad], about 1 [deg]
    # reduce sensibility of sensor: minimum precision is dx
    compensatedAngles = [0, 0, 0]

    for i in range(0, 3):  # i = 0, 1, 2
        if abs(angles[i] / dx) >= 1:
            compensatedAngles[i] = angles[i]

    return compensatedAngles

```

6.21.1.2 eulerAnglesToRotationMatrix()

```
def utilities.eulerAnglesToRotationMatrix (
                angles )
```

Function that transforms euler angle coordinates into the rotation matrix.

Parameters

angles	euler angles, i.e. orientation with respect to X, Y, Z axes

Returns

R: rotation matrix

Definition at line 37 of file utilities.py.

```
def eulerAnglesToRotationMatrix(angles):  # angles [roll, pitch, yaw]
    """!
    Function that transforms euler angle coordinates into the rotation matrix
    @param angles: euler angles, i.e. orientation with respect to X, Y, Z axes
    @returns R: rotation matrix
    """

    R_x = np.array([[1,         0,                  0],
                    [0,         np.cos(angles[0]),   np.sin(angles[0])],
                    [0,         -np.sin(angles[0]),  np.cos(angles[0])]
                    ])

    R_y = np.array([[np.cos(angles[1]),    0,       -np.sin(angles[1])],
                    [0,                      1,       0],
                    [np.sin(angles[1]),    0,       np.cos(angles[1])]
                    ])

    R_z = np.array([[np.cos(angles[2]),      np.sin(angles[2]),     0],
                    [-np.sin(angles[2]),     np.cos(angles[2]),     0],
```

```
                  [0,                     0,                    1]
                  ])

     R = np.dot(R_x, np.dot(R_y, R_z))

     return R
```

6.21.1.3 init_float64_multiarray()

```
def utilities.init_float64_multiarray (
            rows,
            columns )
```

Function that initializes a Float64MultiArray of size rows x columns.

Parameters

rows	Number of rows of the returned multiarray.
columns	Number of columns of the returned multiarray.

Returns

a: empty Float64MultiArray instance.

Definition at line 4 of file utilities.py.

```
4 def init_float64_multiarray(rows,columns):
5     """!
6     Function that initializes a Float64MultiArray of size rows x columns.
7     @param rows: Number of rows of the returned multiarray.
8     @param columns: Number of columns of the returned multiarray.
9     @return a: empty Float64MultiArray instance.
10     """
11     a = Float64MultiArray()
12     a.layout.dim.append(MultiArrayDimension())
13     a.layout.dim.append(MultiArrayDimension())
14     a.layout.dim[0].label ="rows"
15     a.layout.dim[0].size = rows
16     a.layout.dim[1].label ="columns"
17     a.layout.dim[1].size = columns
18     return a
19
```

Chapter 7

Class Documentation

7.1 imu_calib::AccelCalib Class Reference

`#include <accel_calib.h>`

Public Types

- enum Orientation {
  XPOS = 0, XNEG, YPOS, YNEG,
  ZPOS, ZNEG }

Public Member Functions

- AccelCalib ()
- AccelCalib (std::string calib_file)
- bool calibReady ()
- bool loadCalib (std::string calib_file)
- bool saveCalib (std::string calib_file)
- void beginCalib (int measurements, double reference_acceleration)
- bool addMeasurement (Orientation orientation, double ax, double ay, double az)
- bool computeCalib ()
- void applyCalib (double raw[3], double corrected[3])
- void applyCalib (double raw_x, double raw_y, double raw_z, double ∗corr_x, double ∗corr_y, double ∗corr_z)

Protected Attributes

- bool calib_ready_
- Eigen::Matrix3d SM_

  combined scale and misalignment parameters
- Eigen::Vector3d bias_

  scaled and rotated bias parameters
- double reference_acceleration_

  expected acceleration measurement (e.g. 1.0 for unit of g's, 9.80665 for unit of m/s^2)
- bool calib_initialized_
- int orientation_count_ [6]

- Eigen::MatrixXd meas_

  least squares measurements matrix
- Eigen::VectorXd ref_

  least squares expected measurements vector
- int num_measurements_

  number of measurements expected for this calibration
- int measurements_received_

  number of measurements received for this calibration

Static Protected Attributes

- static const int reference_index_ [6] = { 0, 0, 1, 1, 2, 2 }
- static const int reference_sign_ [6] = { 1, -1, 1, -1, 1, -1 }

7.1.1 Detailed Description

Definition at line 49 of file accel_calib.h.

7.1.2 Member Enumeration Documentation

7.1.2.1 Orientation

```
enum imu_calib::AccelCalib::Orientation
```

Enumerator

XPOS	
XNEG	
YPOS	
YNEG	
ZPOS	
ZNEG	

Definition at line 53 of file accel_calib.h.

```
53 { XPOS = 0, XNEG, YPOS, YNEG, ZPOS, ZNEG };
```

7.1.3 Constructor & Destructor Documentation

7.1.3.1 AccelCalib() [1/2]

```
imu_calib::AccelCalib::AccelCalib ( )
```

Definition at line 54 of file accel_calib.cpp.

```
54                          :
55   calib_ready_(false),
56   calib_initialized_(false) {}
```

7.1.3.2 AccelCalib() [2/2]

```
imu_calib::AccelCalib::AccelCalib (
            std::string calib_file )
```

Definition at line 58 of file accel_calib.cpp.

```
59 {
60   AccelCalib();
61   loadCalib(calib_file);
62 }
```

7.1.4 Member Function Documentation

7.1.4.1 addMeasurement()

```
bool imu_calib::AccelCalib::addMeasurement (
            AccelCalib::Orientation orientation,
            double ax,
            double ay,
            double az )
```

Definition at line 145 of file accel_calib.cpp.

```
146 {
147   if (calib_initialized_ && measurements_received_ < num_measurements_)
148   {
149     for (int i = 0; i < 3; i++)
150     {
151       meas_(3*measurements_received_ + i, 3*i) = ax;
152       meas_(3*measurements_received_ + i, 3*i + 1) = ay;
153       meas_(3*measurements_received_ + i, 3*i + 2) = az;
154 
155       meas_(3*measurements_received_ + i, 9 + i) = -1.0;
156     }
157 
158     ref_(3*measurements_received_ + reference_index_[orientation], 0) = reference_sign_[orientation] *
      reference_acceleration_;
159 
160     measurements_received_++;
161     orientation_count_[orientation]++;
162 
163     return true;
164   }
165   else
166   {
167     return false;
168   }
169 }
```

7.1.4.2 applyCalib() [1/2]

```
void imu_calib::AccelCalib::applyCalib (
            double raw[3],
            double corrected[3] )
```

Definition at line 201 of file accel_calib.cpp.

```
{
  Eigen::Vector3d raw_accel(raw[0], raw[1], raw[2]);

  Eigen::Vector3d corrected_accel = SM_*raw_accel - bias_;

  corrected[0] = corrected_accel(0);
  corrected[1] = corrected_accel(1);
  corrected[2] = corrected_accel(2);
}
```

7.1.4.3 applyCalib() [2/2]

```
void imu_calib::AccelCalib::applyCalib (
            double raw_x,
            double raw_y,
            double raw_z,
            double * corr_x,
            double * corr_y,
            double * corr_z )
```

Definition at line 212 of file accel_calib.cpp.

```
{
  Eigen::Vector3d raw_accel(raw_x, raw_y, raw_z);

  Eigen::Vector3d corrected_accel = SM_*raw_accel - bias_;

  *corr_x = corrected_accel(0);
  *corr_y = corrected_accel(1);
  *corr_z = corrected_accel(2);
}
```

7.1.4.4 beginCalib()

```
void imu_calib::AccelCalib::beginCalib (
            int measurements,
            double reference_acceleration )
```

Definition at line 128 of file accel_calib.cpp.

```
{
  reference_acceleration_ = reference_acceleration;

  num_measurements_ = measurements;
  measurements_received_ = 0;

  meas_.resize(3*measurements, 12);
  meas_.setZero();

  ref_.resize(3*measurements);
  ref_.setZero();

  memset(orientation_count_, 0, sizeof(orientation_count_));
  calib_initialized_ = true;
}
```

7.1.4.5 calibReady()

```
bool imu_calib::AccelCalib::calibReady ( )
```

Definition at line 64 of file accel_calib.cpp.

```
{
  return calib_ready_;
}
```

7.1.4.6 computeCalib()

```
bool imu_calib::AccelCalib::computeCalib ( )
```

Definition at line 171 of file accel_calib.cpp.

```
{
  // check status
  if (measurements_received_ < 12)
    return false;

  for (int i = 0; i < 6; i++)
  {
    if (orientation_count_[i] == 0)
      return false;
  }

  // solve least squares
  Eigen::VectorXd xhat = meas_.jacobiSvd(Eigen::ComputeThinU | Eigen::ComputeThinV).solve(ref_);

  // extract solution
  for (int i = 0; i < 9; i++)
  {
    SM_(i/3, i%3) = xhat(i);
  }

  for (int i = 0; i < 3; i++)
  {
    bias_(i) = xhat(9+i);
  }

  calib_ready_ = true;
  return true;
}
```

7.1.4.7 loadCalib()

```
bool imu_calib::AccelCalib::loadCalib (
            std::string calib_file )
```

Definition at line 69 of file accel_calib.cpp.

```
{
  try
  {
    YAML::Node node = YAML::LoadFile(calib_file);

    assert(node["SM"].IsSequence() && node["SM"].size() == 9);
    assert(node["bias"].IsSequence() && node["bias"].size() == 3);

    for (int i = 0; i < 9; i++)
    {
      SM_(i/3,i%3) = node["SM"][i].as<double>();
    }

    for (int i = 0; i < 3; i++)
    {
      bias_(i) = node["bias"][i].as<double>();
    }

```

```
    calib_ready_ = true;
    return true;
  }
  catch (...)
  {
    return false;
  }
}
```

7.1.4.8 saveCalib()

```
bool imu_calib::AccelCalib::saveCalib (
               std::string calib_file )
```

Definition at line 97 of file accel_calib.cpp.

```
{
  if (!calib_ready_)
    return false;

  YAML::Node node;
  for (int i = 0; i < 9; i++)
  {
      node["SM"].push_back(SM_(i/3,i%3));
  }

  for (int i = 0; i < 3; i++)
  {
    node["bias"].push_back(bias_(i));
  }

  try
  {
    std::ofstream fout;
    fout.open(calib_file.c_str());
    fout << node;
    fout.close();
  }
  catch (...)
  {
    return false;
  }

  return true;
}
```

7.1.5 Member Data Documentation

7.1.5.1 bias_

```
Eigen::Vector3d imu_calib::AccelCalib::bias_  [protected]
```

scaled and rotated bias parameters

Definition at line 80 of file accel_calib.h.

7.1.5.2 calib_initialized_

```
bool imu_calib::AccelCalib::calib_initialized_ [protected]
```

Definition at line 84 of file accel_calib.h.

7.1.5.3 calib_ready_

```
bool imu_calib::AccelCalib::calib_ready_ [protected]
```

Definition at line 77 of file accel_calib.h.

7.1.5.4 meas_

```
Eigen::MatrixXd imu_calib::AccelCalib::meas_ [protected]
```

least squares measurements matrix

Definition at line 87 of file accel_calib.h.

7.1.5.5 measurements_received_

```
int imu_calib::AccelCalib::measurements_received_ [protected]
```

number of measurements received for this calibration

Definition at line 90 of file accel_calib.h.

7.1.5.6 num_measurements_

```
int imu_calib::AccelCalib::num_measurements_ [protected]
```

number of measurements expected for this calibration

Definition at line 89 of file accel_calib.h.

7.1.5.7 orientation_count_

```
int imu_calib::AccelCalib::orientation_count_[6]  [protected]
```

Definition at line 85 of file accel_calib.h.

7.1.5.8 ref_

```
Eigen::VectorXd imu_calib::AccelCalib::ref_  [protected]
```

least squares expected measurements vector

Definition at line 88 of file accel_calib.h.

7.1.5.9 reference_acceleration_

```
double imu_calib::AccelCalib::reference_acceleration_  [protected]
```

expected acceleration measurement (e.g. 1.0 for unit of g's, 9.80665 for unit of m/s^2)

Definition at line 82 of file accel_calib.h.

7.1.5.10 reference_index_

```
const int imu_calib::AccelCalib::reference_index_ = { 0, 0, 1, 1, 2, 2 }  [static], [protected]
```

Definition at line 75 of file accel_calib.h.

7.1.5.11 reference_sign_

```
const int imu_calib::AccelCalib::reference_sign_ = { 1, -1, 1, -1, 1, -1 }  [static], [protected]
```

Definition at line 76 of file accel_calib.h.

7.1.5.12 SM_

```
Eigen::Matrix3d imu_calib::AccelCalib::SM_  [protected]
```

combined scale and misalignment parameters

Definition at line 79 of file accel_calib.h.

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

- Smartphone/compensation/include/imu_calib/accel_calib.h
- Smartphone/compensation/src/accel_calib/accel_calib.cpp

7.2 imu_calib::ApplyCalib Class Reference

```
#include <apply_calib.h>
```

Public Member Functions

- ApplyCalib ()

7.2.1 Detailed Description

Definition at line 50 of file apply_calib.h.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 ApplyCalib()

```
imu_calib::ApplyCalib::ApplyCalib ( )
```

Definition at line 47 of file apply_calib.cpp.

```
47                          :
48   gyro_sample_count_(0),
49   gyro_bias_x_(0.0),
50   gyro_bias_y_(0.0),
51   gyro_bias_z_(0.0)
52 {
53   ros::NodeHandle nh;
54   ros::NodeHandle nh_private("~");
55
56   std::string calib_file;
57   nh_private.param<std::string>("calib_file", calib_file, "imu_calib.yaml");
58
59   if (!calib_.loadCalib(calib_file) || !calib_.calibReady())
60   {
61     ROS_FATAL("Calibration could not be loaded");
62     ros::shutdown();
63   }
64
65   nh_private.param<bool>("calibrate_gyros", calibrate_gyros_, true);
66   nh_private.param<int>("gyro_calib_samples", gyro_calib_samples_, 100);
67
68   int queue_size;
69   nh_private.param<int>("queue_size", queue_size, 5);
70
71   raw_sub_ = nh.subscribe("android/imu", queue_size, &ApplyCalib::rawImuCallback, this);
72   corrected_pub_ = nh.advertise<sensor_msgs::Imu>("/android/imu_corrected", queue_size);
73 }
```

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

- Smartphone/compensation/include/imu_calib/apply_calib.h
- Smartphone/compensation/src/apply_calib.cpp

7.3 imu_calib::DoCalib Class Reference

```
#include <do_calib.h>
```

Public Member Functions

- DoCalib ()
- bool running ()

7.3.1 Detailed Description

Definition at line 54 of file do_calib.h.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 DoCalib()

```
imu_calib::DoCalib::DoCalib ( )
```

Definition at line 47 of file do_calib.cpp.

```
47                   :
48   state_(START)
49 {
50   ros::NodeHandle nh;
51   imu_sub_ = nh.subscribe("/android/imu", 1, &DoCalib::imuCallback, this);
52
53   ros::NodeHandle nh_private("~");
54   nh_private.param<int>("measurements", measurements_per_orientation_, 500);
55   nh_private.param<double>("reference_acceleration", reference_acceleration_, 9.80665);
56   nh_private.param<std::string>("output_file", output_file_, "imu_calib.yaml");
57
58   orientations_.push(AccelCalib::XPOS);
59   orientations_.push(AccelCalib::XNEG);
60   orientations_.push(AccelCalib::YPOS);
61   orientations_.push(AccelCalib::YNEG);
62   orientations_.push(AccelCalib::ZPOS);
63   orientations_.push(AccelCalib::ZNEG);
64
65   orientation_labels_[AccelCalib::XPOS] = "X+";
66   orientation_labels_[AccelCalib::XNEG] = "X-";
67   orientation_labels_[AccelCalib::YPOS] = "Y+";
68   orientation_labels_[AccelCalib::YNEG] = "Y-";
69   orientation_labels_[AccelCalib::ZPOS] = "Z+";
70   orientation_labels_[AccelCalib::ZNEG] = "Z-";
71 }
```

7.3.3 Member Function Documentation

7.3.3.1 running()

```
bool imu_calib::DoCalib::running ( )
```

Definition at line 73 of file do_calib.cpp.

```
74 {
75   return state_ != DONE;
76 }
```

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

- Smartphone/compensation/include/imu_calib/do_calib.h
- Smartphone/compensation/src/do_calib.cpp

Chapter 8

File Documentation

8.1 install.sh File Reference

8.2 launcher.sh File Reference

Variables

- gnome terminal x sh c roscore
- bash echo Place the phone on an even horizontal surface echo Press enter to proceed read a gnome terminal x sh c rostopic echo smartphone
- bash echo If you want to launch the sensor calibration process type yes and press enter read b if
- then gnome terminal x sh c rostopic echo android imu
- bash rosrun imu_calib do_calib fi gnome terminal x sh c rosrun imu_calib apply_calib
- bash rosrun smartphone removeGravity py echo With your hand
- bash rosrun smartphone removeGravity py echo With your hold the phone in the calibration position and mantain the position echo Press enter to proceed read a gnome terminal x sh c rosrun smartphone clipping py
- bash gnome terminal x sh c roslaunch wait math_pkg mathAll launch

8.2.1 Variable Documentation

8.2.1.1 apply_calib

```
bash rosrun imu_calib do_calib fi gnome terminal x sh c rosrun imu_calib apply_calib
```

Definition at line 15 of file launcher.sh.

8.2.1.2 hand

```
bash rosrun smartphone removeGravity py echo With your hand
```

Definition at line 17 of file launcher.sh.

8.2.1.3 if

```
bash echo If you want to launch the sensor calibration process type yes and press enter read b
if
```

Definition at line 11 of file launcher.sh.

8.2.1.4 imu

```
then gnome terminal x sh c rostopic echo android imu
```

Definition at line 12 of file launcher.sh.

8.2.1.5 launch

```
bash gnome terminal x sh c roslaunch wait math_pkg mathAll launch
```

Definition at line 22 of file launcher.sh.

8.2.1.6 py

```
bash rosrun smartphone removeGravity py echo With your hold the phone in the calibration position
and mantain the position echo Press enter to proceed read a gnome terminal x sh c rosrun smartphone
clipping py
```

Definition at line 20 of file launcher.sh.

8.2.1.7 roscore

```
gnome terminal x sh c roscore
```

Definition at line 3 of file launcher.sh.

8.2.1.8 smartphone

```
bash echo Place the phone on an even horizontal surface echo Press enter to proceed read a
gnome terminal x sh c rostopic echo smartphone
```

Definition at line 8 of file launcher.sh.

8.3 launcher_test.sh File Reference

Variables

- gnome terminal x sh c roslaunch wait math_pkg mathAll_halfcircle launch
- bash cd Math math_pkg scripts halfcircle_files gnome terminal x sh c rosrun math_pkg test_publisher_↩ halfcircle py

8.3.1 Variable Documentation

8.3.1.1 launch

```
gnome terminal x sh c roslaunch wait math_pkg mathAll_halfcircle launch
```

Definition at line 6 of file launcher_test.sh.

8.3.1.2 py

```
bash cd Math math_pkg scripts halfcircle_files gnome terminal x sh c rosrun math_pkg test_↩
publisher_halfcircle py
```

Definition at line 10 of file launcher_test.sh.

8.4 Math/math_pkg/CMakeLists.txt File Reference

8.5 Smartphone/compensation/CMakeLists.txt File Reference

Functions

- cmake_minimum_required (VERSION 2.8.3) project(imu_calib) find_package(catkin REQUIRED COMPO↩ NENTS cmake_modules roscpp sensor_msgs) find_package(Eigen REQUIRED) pkg_check_modules(YAML yaml-cpp) catkin_package(INCLUDE_DIRS include LIBRARIES accel_calib CATKIN_DEPENDS cmake_↩ modules roscpp sensor_msgs DEPENDS Eigen yaml-cpp) include_directories(include) include_directories($

8.5.1 Function Documentation

8.5.1.1 cmake_minimum_required()

```
cmake_minimum_required (
            VERSION 2.8.   3 )
```

Definition at line 1 of file CMakeLists.txt.

```
30    {catkin_INCLUDE_DIRS}
```

8.6 Smartphone/smartphone/CMakeLists.txt File Reference

8.7 V-rep/CMakeLists.txt File Reference

8.8 Math/math_pkg/scripts/calibration2.py File Reference

Namespaces

- calibration2

Functions

- def calibration2.imu_ee_calibration (data)

  Computes the orientation between 0 and global frame using an initial configuration.
- def calibration2.simulate_callback (data)

  Waits for the start from handle simulation topic.
- def calibration2.calibrate_orientation ()

Variables

- calibration2.pub_rot_matrices
- calibration2.R0e
- int calibration2.start = 0

8.9 Math/math_pkg/scripts/Enhanced_J_Transpose/Forward_Kine_JT.py File Reference

Namespaces

- Forward_Kine_JT

Functions

- def Forward_Kine_JT.main_callback ()

  Publish to error node and inverse kinematics nodes the data, in particular the jacobian matrix, the tracking vectors and the data needed to compute the errors.
- def Forward_Kine_JT.baxter_callback (data)

  Computes the configuration of baxter's arm whenever the data are available, then extracts the rotation matrix from 0 to e.e and the position of the e.e.
- def Forward_Kine_JT.dot_callback (data)

  Reads the q_dots provided by the weighter.
- def Forward_Kine_JT.smart_callback (data)

  Computes the linear acceleration, angular velocity projected on 0 given the data.
- def Forward_Kine_JT.simulate_callback (data)

  Handles changes in the simulation.
- def Forward_Kine_JT.subs ()

Variables

- Forward_Kine_JT.pub_err = rospy.Publisher('Data_for_errors', Float64MultiArray, queue_size=10)
- Forward_Kine_JT.pub_track = rospy.Publisher('tracking', Float64MultiArray, queue_size=10)
- Forward_Kine_JT.pub_jac = rospy.Publisher('jacobian', Float64MultiArray, queue_size=10)
- Forward_Kine_JT.pub_axes = rospy.Publisher('axes', Float64MultiArray, queue_size=10)

  New part.
- Forward_Kine_JT.axis_vect = np.zeros((9,3))
- Forward_Kine_JT.p = np.pi
- int Forward_Kine_JT.n_joints = 7
- float Forward_Kine_JT.L0 = 0.27035
- float Forward_Kine_JT.L1 = 0.06900
- float Forward_Kine_JT.L2 = 0.36435
- float Forward_Kine_JT.L3 = 0.06900
- float Forward_Kine_JT.L4 = 0.37429
- float Forward_Kine_JT.L5 = 0.01000
- float Forward_Kine_JT.L6 = 0.36830
- int Forward_Kine_JT.xeflag = 0
- Forward_Kine_JT.DH
- Forward_Kine_JT.T_dh = t.DH_to_T(DH)
- Forward_Kine_JT.Jkmin1 = np.zeros((6,7))
- Forward_Kine_JT.info = np.array([1, 1, 1, 1, 1, 1, 1])
- int Forward_Kine_JT.ini_bax = 0
- int Forward_Kine_JT.ini_dot = 0
- int Forward_Kine_JT.ini_smart = 0
- int Forward_Kine_JT.key_bax = 0
- int Forward_Kine_JT.key_smart = 0
- int Forward_Kine_JT.key_dot = 0
- int Forward_Kine_JT.key = -1
- int Forward_Kine_JT.flag_bax = 0
- int Forward_Kine_JT.flag_dot = 0
- float Forward_Kine_JT.dt = 0.01
- Forward_Kine_JT.x_0e_kmin1 = np.zeros((3,1))
- Forward_Kine_JT.x_0e_k = x_0e_kmin1
- Forward_Kine_JT.x_0e_kmin1B = np.zeros((3,1))
- Forward_Kine_JT.v_0e_kmin1 = np.zeros((3,1))
- Forward_Kine_JT.v_0e_k = v_0e_kmin1

- Forward_Kine_JT.v_0e_kmin1B = np.zeros((3,1))
- Forward_Kine_JT.q = np.zeros(7)
- Forward_Kine_JT.q_dot = np.zeros((7,1))
- Forward_Kine_JT.R0inert = np.zeros((3,3))
- Forward_Kine_JT.R0e_ini = np.zeros((3,3))
- Forward_Kine_JT.Reimu_ini
- Forward_Kine_JT.Rimu_inert_k = np.zeros((3,3))
- Forward_Kine_JT.R0e_kmin1 = np.zeros((3,3))
- Forward_Kine_JT.R0e_k = np.zeros((3,3))
- Forward_Kine_JT.omega_imu_inert = np.zeros((3,1))
- Forward_Kine_JT.a_imu_inert = np.zeros((3,1))
- Forward_Kine_JT.omega_0e = np.zeros((3,1))
- Forward_Kine_JT.a_0e = np.zeros((3,1))

8.10 Math/math_pkg/scripts/Enhanced_J_Transpose/J_computations.py File Reference

Namespaces

- J_computations

Functions

- def J_computations.geometric_vectors (T_abs)

  Computes the vectors needed to compute geometric jacobian.
- def J_computations.i_j (T_abs)

  New part.
- def J_computations.axis_vector (i, j, k)

  Computes the vector needed for Jtransp optimisation.
- def J_computations.jacob (k, r, n_joints, info)

  Computes the jacobian matrix given the geometric vectors, number of joints and info.

Variables

- J_computations.J = np.concatenate((Jl, Ja), axis = 0)

  else: zero = np.array([[0], [0], [0]]) Ja = np.concatenate((Ja, zero), axis = 1) Jl = np.concatenate((Jl, k[i]), axis = 1)

8.11 Math/math_pkg/scripts/J_computations.py File Reference

Namespaces

- J_computations

Functions

- def J_computations.geometric_vectors (T_abs)

  Computes the vectors needed to compute geometric jacobian.
- def J_computations.jacob (k, r, n_joints, info)

  Computes the jacobian matrix given the geometric vectors, number of joints and info.

8.12 Math/math_pkg/scripts/Enhanced_J_Transpose/J_Transp_server_mod.py File Reference

Namespaces

- J_Transp_server_mod

Functions

- def J_Transp_server_mod.j_transp (err, err_dot, J, Wp, Wd, delta_t, Jp, Ji, alpha, beta, q_dot_pref, min, max)

  Function that performs Inverse kinematics using the Jacobian Transpose approach, considering also the velocity error and the enhancement for singularity escaping (from "Advances in Robot Kinematics: Analysis and Control", J.Lenarcic and M.
- def J_Transp_server_mod.handle_IK_Jtransp (req)
- def J_Transp_server_mod.error_callback (message)
- def J_Transp_server_mod.jacobian_callback (message)
- def J_Transp_server_mod.axes_callback (message)
- def J_Transp_server_mod.JT_server ()

Variables

- bool J_Transp_server_mod.readyErr = False
- bool J_Transp_server_mod.readyJ = False
- bool J_Transp_server_mod.readyJp = False
- list J_Transp_server_mod.wp = [10, 10, 10, 1, 1, 1]
- list J_Transp_server_mod.wd = [0,0,0,0,0,0]
- J_Transp_server_mod.q_dot_elbow = np.array([[0, -1, 0, 10, 0, -0.0001, 0]], dtype=float).T
- int J_Transp_server_mod.ALPHA = 250
- int J_Transp_server_mod.BETA = 250
- int J_Transp_server_mod.MIN = -1
- int J_Transp_server_mod.MAX = 1
- J_Transp_server_mod.script_dir = os.path.dirname(__file__)
- string J_Transp_server_mod.rel_path = "./Output_Jtranspose.txt"
- J_Transp_server_mod.abs_file_path = os.path.join(script_dir, rel_path)
- int J_Transp_server_mod.LOG_FLAG = 0

8.13 Math/math_pkg/scripts/Enhanced_J_Transpose/JT_enhance.py File Reference

Namespaces

- JT_enhance

Functions

- def JT_enhance.JT_enhance (Jp, Ji, err_tran, alpha, beta, q_dot_pref)

    Function that computes a Correction Term to be added to the Jacobian Transpose's formula, in order to escape from singularity (from "Advances in Robot Kinematics: Analysis and Control", J.Lenarcic and M.

8.14 Math/math_pkg/scripts/Errors.py File Reference

Namespaces

- Errors

Functions

- def Errors.ang_mis (Rg, Re)

    Computes the angular misalignment between goal frame and e.e.
- def Errors.errors (data)

    Computes the errors needed by the inverse kinematics algorithms.
- def Errors.errors_node ()

Variables

- Errors.pub = rospy.Publisher('errors', Float64MultiArray, queue_size=10)
- Errors.Rg = np.zeros((3,3))
- Errors.Re = np.zeros((3,3))
- Errors.xg = np.zeros((3,1))
- Errors.xe = np.zeros((3,1))
- Errors.vg = np.zeros((3,1))
- Errors.ve = np.zeros((3,1))

8.15 Math/math_pkg/scripts/Forward_Kine2.py File Reference

Namespaces

- Forward_Kine2

Functions

- def Forward_Kine2.main_callback ()

    Publish to error node and inverse kinematics nodes the data, in particular the jacobian matrix, the tracking vectors and the data needed to compute the errors.
- def Forward_Kine2.baxter_callback (data)

    Computes the configuration of baxter's arm whenever the data are available, then extracts the rotation matrix from 0 to e.e and the position of the e.e.
- def Forward_Kine2.dot_callback (data)

    Reads the q_dots provided by the weighter.
- def Forward_Kine2.smart_callback (data)

    Computes the linear acceleration, angular velocity projected on 0 given the data.
- def Forward_Kine2.calib_callback (data)

    Receives the orientation matrices from the calibration node.
- def Forward_Kine2.simulate_callback (data)

    Handles changes in the simulation.
- def Forward_Kine2.FK ()

Variables

- Forward_Kine2.pub_err = rospy.Publisher('Data_for_errors', Float64MultiArray, queue_size=10)
- Forward_Kine2.pub_track = rospy.Publisher('tracking', Float64MultiArray, queue_size=10)
- Forward_Kine2.pub_jac = rospy.Publisher('jacobian', Float64MultiArray, queue_size=10)
- Forward_Kine2.p = np.pi
- int Forward_Kine2.n_joints = 7
- float Forward_Kine2.L0 = 0.27035
- float Forward_Kine2.L1 = 0.06900
- float Forward_Kine2.L2 = 0.36435
- float Forward_Kine2.L3 = 0.06900
- float Forward_Kine2.L4 = 0.37429
- float Forward_Kine2.L5 = 0.01000
- float Forward_Kine2.L6 = 0.36830
- Forward_Kine2.DH
- Forward_Kine2.T_dh = t.DH_to_T(DH)
- Forward_Kine2.Jkmin1 = np.zeros((6, 7))
- Forward_Kine2.info = np.array([1, 1, 1, 1, 1, 1, 1])
- int Forward_Kine2.ini_bax = 0
- int Forward_Kine2.ini_dot = 0
- int Forward_Kine2.ini_smart = 0
- int Forward_Kine2.key_bax = 0
- int Forward_Kine2.key_smart = 0
- int Forward_Kine2.key_dot = 0
- int Forward_Kine2.key = -1
- int Forward_Kine2.calib_ok = 0
- int Forward_Kine2.sequence = 0
- int Forward_Kine2.steps = 10
- int Forward_Kine2.index = 1
- float Forward_Kine2.dt = 0.01
- Forward_Kine2.x_0e_kmin1 = np.zeros((3, 1))
- Forward_Kine2.x_0e_k = x_0e_kmin1
- Forward_Kine2.x_0e_kmin1B = np.zeros((3, 1))
- Forward_Kine2.v_0e_kmin1 = np.zeros((3, 1))
- Forward_Kine2.v_0e_k = v_0e_kmin1
- Forward_Kine2.v_0e_kmin1B = np.zeros((3, 1))
- Forward_Kine2.q = np.zeros(7)
- Forward_Kine2.q_dot = np.zeros((7, 1))
- Forward_Kine2.R0e_ini = np.zeros((3, 3))
- Forward_Kine2.R0global = np.zeros((3, 3))
- Forward_Kine2.Re_imu = np.zeros((3, 3))
- Forward_Kine2.Rimu_global_k = np.zeros((3, 3))
- Forward_Kine2.R0e_kmin1 = np.zeros((3, 3))
- Forward_Kine2.R0e_k = np.zeros((3, 3))
- Forward_Kine2.omega_imu_global = np.zeros((3, 1))
- Forward_Kine2.a_imu_global = np.zeros((3, 1))
- Forward_Kine2.omega_0e = np.zeros((3, 1))
- Forward_Kine2.a_0e = np.zeros((3, 1))

8.16 Math/math_pkg/scripts/Forward_Kine_halfcircle.py File Reference

Namespaces

- Forward_Kine_halfcircle

Functions

- def Forward_Kine_halfcircle.main_callback ()

  Publish to error node and inverse kinematics nodes the data, in particular the jacobian matrix, the tracking vectors and the data needed to compute the errors.
- def Forward_Kine_halfcircle.baxter_callback (data)

  Computes the configuration of baxter's arm whenever the data are available, then extracts the rotation matrix from 0 to e.e and the position of the e.e.
- def Forward_Kine_halfcircle.dot_callback (data)

  Reads the q_dots provided by the weighter.
- def Forward_Kine_halfcircle.smart_callback (data)

  Computes the linear acceleration, angular velocity projected on 0 given the data.
- def Forward_Kine_halfcircle.simulate_callback (data)

  Handles changes in the simulation.
- def Forward_Kine_halfcircle.FK ()

Variables

- Forward_Kine_halfcircle.pub_err = rospy.Publisher('Data_for_errors', Float64MultiArray, queue_size=10)
- Forward_Kine_halfcircle.pub_track = rospy.Publisher('tracking', Float64MultiArray, queue_size=10)
- Forward_Kine_halfcircle.pub_jac = rospy.Publisher('jacobian', Float64MultiArray, queue_size=10)
- Forward_Kine_halfcircle.p = np.pi
- int Forward_Kine_halfcircle.n_joints = 7
- float Forward_Kine_halfcircle.L0 = 0.27035
- float Forward_Kine_halfcircle.L1 = 0.06900
- float Forward_Kine_halfcircle.L2 = 0.36435
- float Forward_Kine_halfcircle.L3 = 0.06900
- float Forward_Kine_halfcircle.L4 = 0.37429
- float Forward_Kine_halfcircle.L5 = 0.01000
- float Forward_Kine_halfcircle.L6 = 0.36830
- int Forward_Kine_halfcircle.xeflag = 0
- Forward_Kine_halfcircle.DH
- Forward_Kine_halfcircle.T_dh = t.DH_to_T(DH)
- Forward_Kine_halfcircle.Jkmin1 = np.zeros((6,7))
- Forward_Kine_halfcircle.info = np.array([1, 1, 1, 1, 1, 1, 1])
- int Forward_Kine_halfcircle.ini_bax = 0
- int Forward_Kine_halfcircle.ini_dot = 0
- int Forward_Kine_halfcircle.ini_smart = 0
- int Forward_Kine_halfcircle.key_bax = 0
- int Forward_Kine_halfcircle.key_smart = 0
- int Forward_Kine_halfcircle.key_dot = 0
- int Forward_Kine_halfcircle.key = 1
- int Forward_Kine_halfcircle.flag_bax = 0
- int Forward_Kine_halfcircle.flag_dot = 0
- float Forward_Kine_halfcircle.dt = 0.01
- Forward_Kine_halfcircle.x_0e_kmin1 = np.zeros((3,1))
- Forward_Kine_halfcircle.x_0e_k = x_0e_kmin1
- Forward_Kine_halfcircle.x_0e_kmin1B = np.zeros((3,1))
- Forward_Kine_halfcircle.v_0e_kmin1 = np.zeros((3,1))
- Forward_Kine_halfcircle.v_0e_k = v_0e_kmin1
- Forward_Kine_halfcircle.v_0e_kmin1B = np.zeros((3,1))
- Forward_Kine_halfcircle.q = np.zeros(7)
- Forward_Kine_halfcircle.q_dot = np.zeros((7,1))

- Forward_Kine_halfcircle.R0inert = np.zeros((3,3))
- Forward_Kine_halfcircle.R0e_ini = np.zeros((3,3))
- Forward_Kine_halfcircle.Rimu_inert_k = np.zeros((3,3))
- Forward_Kine_halfcircle.R0e_kmin1 = np.zeros((3,3))
- Forward_Kine_halfcircle.R0e_k = np.zeros((3,3))
- Forward_Kine_halfcircle.omega_imu_inert = np.zeros((3,1))
- Forward_Kine_halfcircle.a_imu_inert = np.zeros((3,1))
- Forward_Kine_halfcircle.omega_0e = np.zeros((3,1))
- Forward_Kine_halfcircle.a_0e = np.zeros((3,1))

8.17 Math/math_pkg/scripts/integrator.py File Reference

Namespaces

- integrator

Functions

- def integrator.sat (x, xmin, xmax)
- def integrator.qdot_callback (qdot_data)

    Receives qdot vector and sents q vector obtained by integration.
- def integrator.simulate_callback (data)

    Handles changes in the simulation.
- def integrator.integr ()

    Integrator function.

Variables

- integrator.q = np.zeros((7,1))
- integrator.qdot = np.zeros((7,1))
- integrator.qdotprev = None
- integrator.qdotprevprev = None
- bool integrator.qdotppnone = True
- bool integrator.qdotpnone = True
- integrator.qold = np.zeros((7,1))
- int integrator.eff = 0
- int integrator.key = 0
- float integrator.DT = 0.01
- list integrator.qmin = [-1.6817,-2.1268,-3.0343,-0.3,-3.0396,-1.5508,-3.0396]
- list integrator.qmax = [1.6817,1.0272,3.0343,2.5829,3.0378,2.0744,3.0378]
- integrator.anonymous
- integrator.pub = rospy.Publisher('logtopic', JointState, queue_size=10)

8.18 Math/math_pkg/scripts/J_Transp_server.py File Reference

Namespaces

- J_Transp_server

Functions

- def J_Transp_server.init_float64_multiarray (rows, columns)

  Function that initializes a Float64MultiArray of size rows x columns.
- def J_Transp_server.j_transp (err, err_dot, J, Wp, Wd, delta_t)

  Function that performs Inverse kinematics using the Jacobian Transpose approach.
- def J_Transp_server.handle_IK_Jtransp (req)
- def J_Transp_server.error_callback (message)
- def J_Transp_server.jacobian_callback (message)
- def J_Transp_server.JT_server ()

Variables

- bool J_Transp_server.readyErr = False
- bool J_Transp_server.readyJ = False
- list J_Transp_server.wp = [50, 50, 50, 1, 1, 1]
- list J_Transp_server.wd = [50,50,50,0.5,0.5,0.5]
- J_Transp_server.script_dir = os.path.dirname(__file__)
- string J_Transp_server.rel_path = "./Output/Output.txt"
- J_Transp_server.abs_file_path = os.path.join(script_dir, rel_path)
- int J_Transp_server.LOG_FLAG = 0

8.19 Math/math_pkg/scripts/jac_mat.py File Reference

Namespaces

- jac_mat

Functions

- def jac_mat.sat (x, xmin, xmax)

  Saturates the input value in the given interval.
- def jac_mat.bell (s)

  Creates a bell shaped function to regularize singular values.
- def jac_mat.regularized_pseudoinverse (J)

  Computes the regularized pseudo inverse of mxn matrix J by applying svd decomposition and multiplication for bell shaped function.
- def jac_mat.calculations_6 (q_coppelia)

  Builds the 6 dof Jacobian by computing the transformation matrices, extracting the geometric vectors from the results and finally building the matrix.
- def jac_mat.jacobian_callback (data)

  Callback Function for the Joints Positions.
- def jac_mat.error_callback (message)

  Callback Function for the error on the position of the ee.
- def jac_mat.vel_callback (message)

  Callback Function for the linear and angular velocities of the ee.
- def jac_mat.handle_IK_JAnalytic (req)

  Handler for the Server: computes the ee velocity Vee and the joint velocities vector q_dot Vee is computed as: Vee = vel + K∗error q_dot is computed as: q_dot = J# ∗ Vee then each value of q_dot is saturated in a pre-determined interval.
- def jac_mat.jac_mat ()

Variables

- bool jac_mat.readyErr = False
- bool jac_mat.readyJ = False
- bool jac_mat.readyVel = False

8.20 Math/math_pkg/scripts/T_computations.py File Reference

Namespaces

- T_computations

Functions

- def T_computations.DH_to_T (DH)

  Computes the transformation matrices given the DH table of the serial link.
- def T_computations.transformations (T_rel_ini, q, info)

  Computes tranformations given T_relatives, q's and the info.
- def T_computations.abs_trans (T_rel)

  Computes trasformations matrices w.r.t.

Variables

- T_computations.p = np.pi
- T_computations.Tmp = np.dot(T_rel_ini[i], Tel)

  Tel = np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 0, q[i]] [0, 0, 0, 1]])

8.21 Math/math_pkg/scripts/test_publisher_halfcircle.py File Reference

Namespaces

- test_publisher_halfcircle

Functions

- def test_publisher_halfcircle.simulate_callback (data)

  Handles changes in the simulation.
- def test_publisher_halfcircle.talker ()

  Publishes mock smartphone signals, read from files.

Variables

- int test_publisher_halfcircle.key = 0
- int test_publisher_halfcircle.reset = 0
- test_publisher_halfcircle.pub = None
- test_publisher_halfcircle.anonymous

8.22 Math/math_pkg/scripts/utilities.py File Reference

Namespaces

- utilities

Functions

- def utilities.init_float64_multiarray (rows, columns)

  Function that initializes a Float64MultiArray of size rows x columns.
- def utilities.anglesCompensate (angles)

  Function used to filter unwanted minimal incoming data fluctuations, due to noise as well as human operator shake.
- def utilities.eulerAnglesToRotationMatrix (angles)

  Function that transforms euler angle coordinates into the rotation matrix.

8.23 Math/math_pkg/src/cost.cpp File Reference

```
#include <algorithm>
#include <iostream>
#include "math_pkg/Cost.h"
#include "math_pkg/IK.h"
#include "ros/ros.h"
#include "sensor_msgs/JointState.h"
#include "std_msgs/Float64MultiArray.h"
#include "utilities.h"
```

Functions

- void costCallbackq (const sensor_msgs::JointState &msg)
- void computeCostResponse (VectorXd q, VectorXd qdot1, VectorXd qdot2, MatrixXd Q2, math_pkg::Cost::↩ Response &res)
- bool computeOptqdot (math_pkg::Cost::Request &req, math_pkg::Cost::Response &res)
- int main (int argc, char **argv)

Variables

- bool readyq
- VectorXd q
- ros::ServiceClient client
- math_pkg::IK ikSrv
- int seqtry
- int costFail = 0

8.23.1 Function Documentation

8.23.1.1 computeCostResponse()

```
void computeCostResponse (
            VectorXd q,
            VectorXd qdot1,
            VectorXd qdot2,
            MatrixXd Q2,
            math_pkg::Cost::Response & res )
```

Function that computes the optimized velocity vectors.

Parameters

q	Most recent arm configuration.
qdot1	Joint velocity vector computed with closed loop IK of order 1.
qdot2	Joint velocity vector computed with closed loop IK of order 2.
Q2	Auxiliary matrix for tracking task.
res	Response to client of cost service.

Definition at line 46 of file cost.cpp.

```
46
      {
47      VectorXd qdot1opt1,qdot1opt2,qdot2opt1,qdot2opt2; // initialization
48
49      double cond1,cond2;
50
51      // Each optimized velocity vector is given by: non-optimized vector + G * z, where z is a NJOINTSx1
        vector.
52      // Optimization n. 1: minimize qdot
53      /* qdot1opt1 and qdot2opt1 are computed according to the paper "A Novel Practical Technique to
        Integrate Inequality Control
54       * Objectives and Task Transitions in Priority Based Control" by Casalino & Simetti, p. 20, sec.
        4.4.*/
55      MatrixXd IdMinusQ2 = ID_MATRIX_NJ - Q2;
56      MatrixXd toPinv = Q2.transpose() * Q2 + 0.1 * IdMinusQ2.transpose()*(IdMinusQ2);
57      MatrixXd temp1 = -regPinv(toPinv,ID_MATRIX_NJ,ID_MATRIX_NJ,ETA,cond1) * Q2.transpose();
58      qdot1opt1 = qdot1 + Q2*temp1*qdot1; // optimize solution 1
59      qdot2opt1 = qdot2 + Q2*temp1*qdot2; // optimize solution 2
60
61      // Optimization n. 2: stay close to favourite pose (initial pose)
62      MatrixXd Q2Sharp = regPinv(Q2,ID_MATRIX_NJ,ID_MATRIX_NJ,ETA,cond2);
63      VectorXd qdot_fav = 0.001 * q;
64      int multFact = 0;
65      if (cond2 < 100) multFact = 1;
66      qdot1opt2 = qdot1 + multFact*Q2*Q2Sharp*(qdot_fav - qdot1); // optimize solution 1
67      qdot2opt2 = qdot2 + multFact*Q2*Q2Sharp*(qdot_fav - qdot2); // optimize solution 2
68
69      // Fill the response object.
70      res.qdot1opt1.velocity = vector<double> (qdot1opt1.data(),qdot1opt1.data()+qdot1opt1.size());
71      res.qdot1opt2.velocity = vector<double> (qdot1opt2.data(),qdot1opt2.data()+qdot1opt2.size());
72      res.qdot2opt1.velocity = vector<double> (qdot2opt1.data(),qdot2opt1.data()+qdot2opt1.size());
73      res.qdot2opt2.velocity = vector<double> (qdot2opt2.data(),qdot2opt2.data()+qdot2opt2.size());
74 }
```

8.23.1.2 computeOptqdot()

```
bool computeOptqdot (
            math_pkg::Cost::Request & req,
            math_pkg::Cost::Response & res )
```

Service function for the Safety service, which computes the optimized joint velocities according to 2 optimization criteria: closeness to a preferred positional configuration and closeness to a preferred velocity vector.

Parameters

req	Contains current sequence number, sent by the weighter.
res	Four optimized joint velocities, respectively computed from IK 1 with cost function 1 and 2, and from IK 2 again with both cost functions.

Returns

true if client-service call succeeded, false otherwise.

Definition at line 84 of file cost.cpp.

```
84                                                                                    {
85      //clock_t begin = clock();
86      if (client.call(ikSrv)) { // if Jacobian is available and the service call succeeded
87          if (!(readyq && seqtry == req.seq.data)) { // Jacobian not available
88              readyq = false; // reset availability flag
89              ROS_ERROR("cost service could not run: missing data.");
90              return false;
91          }
92          readyq = false; // reset availability flag
93 
94          // Map the non-optimized vectors and matrix returned by the call into Eigen library objects.
95          VectorXd qdot1 = Map<VectorXd>(ikSrv.response.qdot1.velocity.data(),NJOINTS);
96          VectorXd qdot2 = Map<VectorXd>(ikSrv.response.qdot2.velocity.data(),NJOINTS);
97          MatrixXd Q2 = Map<MatrixXd>(ikSrv.response.Q2.data.data(),NJOINTS,NJOINTS);
98 
99          // Compute optimized vectors and fill the response object.
100          computeCostResponse(q,qdot1,qdot2,Q2,res);
101      }
102      else { // if Jacobian is available but the service call did not succeed
103          readyq = false; // reset availability flag
104          ROS_ERROR("Call to ik service failed.");
105          return false;
106      }
107      return true;
108 }
```

8.23.1.3 costCallbackq()

```
void costCallbackq (
            const sensor_msgs::JointState & msg )
```

Callback function for joint position.

Parameters

msg	The received joint position vector.

Definition at line 30 of file cost.cpp.

```
31 {
32      seqtry = (int)msg.effort[0]; // store seq number of received q
33      vector<double> rcvdq = msg.position;
34      q = Map<VectorXd>(rcvdq.data(),NJOINTS); // store q in global variable
35      readyq = true;
36 }
```

8.23.1.4 main()

```
int main (
```

```
int argc,
char ** argv )
```

Main function of the node.

Definition at line 113 of file cost.cpp.

```
113                             {
114     ros::init(argc, argv, "cost_server"); // initialize node
115     ros::NodeHandle n; // define node handle
116 
117     int queSize = 10;
118     ros::Subscriber sub = n.subscribe("logtopic", queSize, costCallbackq); // subscribe to integrator
119 
120     ros::ServiceServer service = n.advertiseService("cost", computeOptqdot); // activate Cost service
121 
122     client = n.serviceClient<math_pkg::IK>("ik"); // Cost is a client of IK
123     ros::spin();
124     return 0;
125 }
```

8.23.2 Variable Documentation

8.23.2.1 client

```
ros::ServiceClient client
```

Client object needed to perform service calls.

Definition at line 18 of file cost.cpp.

8.23.2.2 costFail

```
int costFail = 0
```

Failure counter, used for debug.

Definition at line 24 of file cost.cpp.

8.23.2.3 ikSrv

```
math_pkg::IK ikSrv
```

IK server object.

Definition at line 20 of file cost.cpp.

8.23.2.4 q

```
VectorXd q
```

Current Jacobian matrix.

Definition at line 16 of file cost.cpp.

8.23.2.5 readyq

```
bool readyq
```

Availability flag for Jacobian matrix.

Definition at line 14 of file cost.cpp.

8.23.2.6 seqtry

```
int seqtry
```

Sequence number for received q message.

Definition at line 22 of file cost.cpp.

8.24 Math/math_pkg/src/ik.cpp File Reference

```
#include <algorithm>
#include <fstream>
#include <iostream>
#include "math_pkg/IK.h"
#include "math_pkg/Safety.h"
#include "ros/ros.h"
#include "sensor_msgs/JointState.h"
#include "std_msgs/Float64MultiArray.h"
#include "utilities.h"
```

Macros

- #define Kpp 10
- #define Kp 100
- #define Kv 20
- #define Krot 10

Functions

- void ikCallbackJ (const std_msgs::Float64MultiArray &msg)
- void ikCallbackErr (const std_msgs::Float64MultiArray &msg)
- void ikCallbackVwa (const std_msgs::Float64MultiArray &msg)
- void ikCallbackqdot (const sensor_msgs::JointState &msg)
- void computeqdot (VectorXd partialqdot, MatrixXd Q1, MatrixXd J, MatrixXd JL, MatrixXd JLdot, VectorXd qdot, VectorXd eta, VectorXd rho, VectorXd etadot, VectorXd v, VectorXd w, VectorXd a, VectorXd &qdot1, VectorXd &qdot2)
- bool computeIKqdot (math_pkg::IK::Request &req, math_pkg::IK::Response &res)
- int main (int argc, char **argv)

Variables

- bool firstStep = true
- bool readyJ
- bool readyErr
- bool readyVwa
- bool readyqdot
- MatrixXd J
- MatrixXd JL
- MatrixXd JLdot
- MatrixXd JLold
- VectorXd eta
- VectorXd rho
- VectorXd nu
- VectorXd v
- VectorXd w
- VectorXd a
- VectorXd qdot
- ros::ServiceClient client
- math_pkg::Safety safeSrv
- const double thr_still = 1e-4

8.24.1 Macro Definition Documentation

8.24.1.1 Kp

```
#define Kp 100
```

Linear positional gain for tracking in CLIK2.

Definition at line 16 of file ik.cpp.

8.24.1.2 Kpp

```
#define Kpp 10
```

Linear positional gain for tracking in CLIK1.

Definition at line 14 of file ik.cpp.

8.24.1.3 Krot

```
#define Krot 10
```

Rotational gain for tracking.

Definition at line 20 of file ik.cpp.

8.24.1.4 Kv

```
#define Kv 20
```

Linear velocity gain for tracking in CLIK2.

Definition at line 18 of file ik.cpp.

8.24.2 Function Documentation

8.24.2.1 computeIKqdot()

```
bool computeIKqdot (
            math_pkg::IK::Request & req,
            math_pkg::IK::Response & res )
```

Service function for the Safety service, which computes the partial joint velocities based on the safety task.

Parameters

req	Server request.
res	Server response.

Returns

true if client-service call succeeded, false otherwise.

Definition at line 166 of file ik.cpp.

```
166                                                                  {
167     //clock_t begin = clock(); // timing
168     if (client.call(safeSrv)) { // if all subscription data is available and service call succeeded
169         if (!(readyJ && readyErr && readyVwa && readyqdot)) { // at least one subscribtion data is missing
170             readyJ = readyErr = readyVwa = readyqdot = false; // reset availability flags
171             ROS_ERROR("ik service could not run: missing data.");
172             return false;
173         }
174         readyJ = readyErr = readyVwa = readyqdot = false; // reset availability flags
175         double cond;
176 
177         // Map the vectors returned by the call into Eigen library objects.
178         VectorXd partialqdot = Map<VectorXd>(safeSrv.response.qdot.velocity.data(),NJOINTS);
179         MatrixXd Q1 = Map<MatrixXd>(safeSrv.response.Q1.data.data(),NJOINTS,NJOINTS);
180 
181         // Compute Q2 matrix and store it in std::vector
182         MatrixXd Q2 = Q1*(ID_MATRIX_NJ - regPinv(J*Q1,ID_MATRIX_SPACE_DOFS,ID_MATRIX_NJ,ETA,cond)*J*Q1);
183         Map<MatrixXd> Q2v (Q2.data(), NJOINTS*NJOINTS,1);
184         res.Q2.data = vector<double> (Q2v.data(), Q2v.data() + Q2v.size());
185 
186         JL = J.block<3,NJOINTS>(0,0); // Extract linear part of the Jacobian matrix.
187         if (firstStep) firstStep = false;
188         else JLdot = (JL - JLold) / DT; // unless it's the first step, compute JLdot as a finite difference of linear Jacobian matrices.
189 
190         // Compute the non-optimized joint velocities.
191         VectorXd qdot1; // will contain qdot computed according to closed loop IK first order.
192         VectorXd qdot2; // will contain qdot computed according to closed loop IK second order.
193         computeqdot(partialqdot,Q1,J,JL,JLdot,qdot,eta,rho,nu,v,w,a,qdot1,qdot2);
194         JLold = JL; // update JLold
195 
196         // Fill response objects.
197         res.qdot1.velocity = vector<double> (qdot1.data(), qdot1.data() + qdot1.size());
198         res.qdot2.velocity = vector<double> (qdot2.data(), qdot2.data() + qdot2.size());
199     }
200     else { // if all subscription data is available but service call did not succeed
201         readyJ = readyErr = readyVwa = readyqdot = false; // reset availability flags
202         ROS_ERROR("Call to safety service failed.");
203         return false;
204     }
205 
206     return true; // call succeeded
207 }
```

8.24.2.2 computeqdot()

```
void computeqdot (
            VectorXd partialqdot,
            MatrixXd Q1,
            MatrixXd J,
            MatrixXd JL,
            MatrixXd JLdot,
            VectorXd qdot,
            VectorXd eta,
            VectorXd rho,
            VectorXd etadot,
            VectorXd v,
            VectorXd w,
            VectorXd a,
            VectorXd & qdot1,
            VectorXd & qdot2 )
```

Function that computes non-optimized qdot with 2 CLIK algorithms.

Parameters

partialqdot	Safety task based qdot vector.
Q1	Auxiliary matrix for tracking task.
J	Jacobian matrix.
JL	Linear Jacobian matrix.
JLdot	Derivative of linear Jacobian matrix.
qdot	Previous qdot vector.
eta	Linear error vector.
rho	Rotational error vector.
etadot	Linear velocity error vector.
v	Target velocity vector.
w	Target angular velocity vector.
a	Target acceleration vector.
qdot1	Reference to CLIK 1st order solution, to be filled.
qdot2	Reference to CLIK 2nd order solution, to be filled.

Definition at line 137 of file ik.cpp.

```
                                                                              {
    VectorXd ve1 = v + Kpp*eta; // ee lin velocity for CLIK1
    VectorXd ve2 = DT*(a - JLdot*qdot + Kv*etadot + Kp*eta) + JL*qdot; // ee lin velocity for CLIK2
    VectorXd xedot1 = VectorXd(6);
    VectorXd xedot2 = VectorXd(6);
    xedot1 << ve1,w+Krot*rho; // ee velocity for CLIK1
    xedot2 << ve2,w+Krot*rho; // ee velocity for CLIK2
    double cond;
    /* qdots are computed according to the paper "A Novel Practical Technique to Integrate Inequality
   Control
      * Objectives and Task Transitions in Priority Based Control" by Casalino & Simetti, pp. 16-17, sec.
   3.4. */
    MatrixXd JTimesQ1 = J*Q1;
    MatrixXd pinvAux = regPinv(JTimesQ1,ID_MATRIX_SPACE_DOFS,Q1,ETA,cond);
    MatrixXd pinvQZero = regPinv(JTimesQ1,ID_MATRIX_SPACE_DOFS,ID_MATRIX_NJ,ETA,cond);
    MatrixXd W2 = JTimesQ1*pinvAux;
    MatrixXd tempProduct1 = Q1*pinvQZero*W2;
    MatrixXd tempProduct2 = J*partialqdot;

    qdot1 = partialqdot + tempProduct1 * (xedot1 - tempProduct2);
    qdot2 = partialqdot + tempProduct1 * (xedot2 - tempProduct2);
}
```

8.24.2.3 ikCallbackErr()

```
void ikCallbackErr (
            const std_msgs::Float64MultiArray & msg )
```

Callback function for error vectors.

Parameters

msg	The received error vectors.

Definition at line 77 of file ik.cpp.

```
{
    vector<double> rcvdErr = msg.data;
    rho = Map<VectorXd>(rcvdErr.data(),3);
    eta = Map<VectorXd>(rcvdErr.data()+3,3);
    nu = Map<VectorXd>(rcvdErr.data()+6,3);
    if (abs(eta(0)) < thr_still && abs(eta(1)) < thr_still && abs(eta(2)) < thr_still &&
```

```
        abs(rho(0)) < thr_still && abs(rho(1)) < thr_still && abs(rho(2)) < thr_still) {
            eta(0) = eta(1) = eta(2) = rho(0) = rho(1) = rho(2) = nu(0) = nu(1) = nu(2) = 0;
            stay_still = true;
    }
    else stay_still = false;

    readyErr = true;
}
```

8.24.2.4 ikCallbackJ()

```
void ikCallbackJ (
        const std_msgs::Float64MultiArray & msg )
```

Callback function for Jacobian matrix.

Parameters

msg	The received Jacobian matrix.

Definition at line 64 of file ik.cpp.

```
{

    vector<double> rcvdJ = msg.data;
    J = Map<MatrixXd>(rcvdJ.data(),NJOINTS,6).transpose();
    readyJ = true;
}
```

8.24.2.5 ikCallbackqdot()

```
void ikCallbackqdot (
        const sensor_msgs::JointState & msg )
```

Callback function for joint velocities.

Parameters

msg	The received joint velocities vector.

Definition at line 112 of file ik.cpp.

```
{
    vector<double> rcvdqdot = msg.velocity;
    qdot = Map<VectorXd>(rcvdqdot.data(),NJOINTS);
    readyqdot = true;
}
```

8.24.2.6 ikCallbackVwa()

```
void ikCallbackVwa (
        const std_msgs::Float64MultiArray & msg )
```

Callback function for tracking signals.

Parameters

msg	The received tracking signals (target linear velocity,angular velocity,linear acceleration).

Definition at line 98 of file ik.cpp.

```
99 {
100     vector<double> rcvdVwa = msg.data;
101     v = Map<VectorXd>(rcvdVwa.data(),3);
102     w = Map<VectorXd>(rcvdVwa.data()+3,3);
103     a = Map<VectorXd>(rcvdVwa.data()+6,3);
104     readyVwa = true;
105 }
```

8.24.2.7 main()

```
int main (
        int argc,
        char ** argv )
```

Main function of the node.

Definition at line 213 of file ik.cpp.

```
213                                    {
214     ros::init(argc, argv, "ik_server"); // initialize node
215     ros::NodeHandle n; // define node handle
216     int queSize = 10;
217     ros::Subscriber sub1 = n.subscribe("jacobian", queSize, ikCallbackJ); // subscribe to Jacobian
218     ros::Subscriber sub2 = n.subscribe("errors", queSize, ikCallbackErr); // subscribe to errors
219     ros::Subscriber sub3 = n.subscribe("tracking", queSize, ikCallbackVwa); // subscribe to tracking
       signals
220     ros::Subscriber sub4 = n.subscribe("cmdtopic", queSize, ikCallbackqdot); // describe to weighter
221 
222     ros::ServiceServer service = n.advertiseService("ik", computeIKqdot); // activate IK service
223 
224     client = n.serviceClient<math_pkg::Safety>("safety"); // IK is client of Safety
225 
226     JLdot = MatrixXd::Zero(3,NJOINTS);
227 
228     ros::spin(); // keep the node alive
229     return 0;
230 }
```

8.24.3 Variable Documentation

8.24.3.1 a

```
VectorXd a
```

Current linear acceleration vector.

Definition at line 50 of file ik.cpp.

8.24.3.2 client

```
ros::ServiceClient client
```

Client object needed to perform service calls.

Definition at line 54 of file ik.cpp.

8.24.3.3 eta

```
VectorXd eta
```

Current linear error vector.

Definition at line 40 of file ik.cpp.

8.24.3.4 firstStep

```
bool firstStep = true
```

First step flag.

Definition at line 22 of file ik.cpp.

8.24.3.5 J

```
MatrixXd J
```

Current Jacobian matrix.

Definition at line 32 of file ik.cpp.

8.24.3.6 JL

```
MatrixXd JL
```

Current linear part of the Jacobian matrix.

Definition at line 34 of file ik.cpp.

8.24.3.7 JLdot

```
MatrixXd JLdot
```

Current derivative of Jacobian matrix.

Definition at line 36 of file ik.cpp.

8.24.3.8 JLold

```
MatrixXd JLold
```

Previous value of linear part of the Jacobian matrix.

Definition at line 38 of file ik.cpp.

8.24.3.9 nu

```
VectorXd nu
```

Current linear velocity error vector.

Definition at line 44 of file ik.cpp.

8.24.3.10 qdot

```
VectorXd qdot
```

Current joint velocities vector.

Definition at line 52 of file ik.cpp.

8.24.3.11 readyErr

```
bool readyErr
```

Availability flag for error vector.

Definition at line 26 of file ik.cpp.

8.24.3.12 readyJ

```
bool readyJ
```

Availability flag for Jacobian matrix.

Definition at line 24 of file ik.cpp.

8.24.3.13 readyqdot

```
bool readyqdot
```

Availability flag for joint velocities vector.

Definition at line 30 of file ik.cpp.

8.24.3.14 readyVwa

```
bool readyVwa
```

Availability flag for tracking signals.

Definition at line 28 of file ik.cpp.

8.24.3.15 rho

```
VectorXd rho
```

Current rotational error vector.

Definition at line 42 of file ik.cpp.

8.24.3.16 safeSrv

```
math_pkg::Safety safeSrv
```

Safety server object.

Definition at line 56 of file ik.cpp.

8.24.3.17 thr_still

```
const double thr_still = 1e-4
```

Threshold under which error is considered zero.

Definition at line 58 of file ik.cpp.

8.24.3.18 v

```
VectorXd v
```

Current linear velocity vector.

Definition at line 46 of file ik.cpp.

8.24.3.19 w

```
VectorXd w
```

Current angular velocity vector.

Definition at line 48 of file ik.cpp.

8.25 Math/math_pkg/src/safety.cpp File Reference

```
#include <algorithm>
#include "math_pkg/Safety.h"
#include "ros/ros.h"
#include "sensor_msgs/JointState.h"
#include "std_msgs/Float64MultiArray.h"
#include "utilities.h"
```

Macros

- #define CORRPOLE_POS 70
- #define CORRPOLE_VEL 8
- #define JOINTS_MARGIN 0.1
- #define VEL_MARGIN 0.01

Functions

- void safetyCallbackq (const sensor_msgs::JointState &msg)
- void safetyCallbackqdot (const sensor_msgs::JointState &msg)
- double cos_sigmoid (double x, double y, double mrgn)
- bool jointConstr (double x, const double xmin, const double xmax, const double mrgn, const double cPole, double &currentPole, const bool isJoint, double &rdot, double &Adiag)
- void safetyLoop (VectorXd &rdot, VectorXd &Adiag)
- bool computePartialqdot (math_pkg::Safety::Request &req, math_pkg::Safety::Response &res)
- int main (int argc, char **argv)

Variables

- bool readyq
- bool readyqdot
- double currentAnglePole [NJOINTS]
- double currentVelPole [NJOINTS]
- double q [NJOINTS]
- double qdot [NJOINTS]
- int seqtry
- int seqqdot

8.25.1 Macro Definition Documentation

8.25.1.1 CORRPOLE_POS

```
#define CORRPOLE_POS 70
```

Abs value of the position correction pole for safety task.

Definition at line 23 of file safety.cpp.

8.25.1.2 CORRPOLE_VEL

```
#define CORRPOLE_VEL 8
```

Abs value of the velocity correction pole for safety task.

Definition at line 25 of file safety.cpp.

8.25.1.3 JOINTS_MARGIN

```
#define JOINTS_MARGIN 0.1
```

Soft margin of joint angles.

Definition at line 27 of file safety.cpp.

8.25.1.4 VEL_MARGIN

```
#define VEL_MARGIN 0.01
```

Soft margin of joint velocities.

Definition at line 29 of file safety.cpp.

8.25.2 Function Documentation

8.25.2.1 computePartialqdot()

```
bool computePartialqdot (
            math_pkg::Safety::Request & req,
            math_pkg::Safety::Response & res )
```

Service function for the Safety service, which computes the partial joint velocities based on the safety task.

Parameters

req	Server request object.
res	Server response object.

Returns

true if client-service call succeeded, false otherwise.

Definition at line 149 of file safety.cpp.

```
                                                                                {
    if (!(readyq && readyqdot)||(seqtry != seqqdot)) { // at least one subscribtion data is missing
        readyq = readyqdot = false; // reset availability flag
        ROS_ERROR("safety service could not run: missing data.");
        return false;
    }

    readyq = readyqdot = false; // reset availability flag
    VectorXd partial_qdot(NJOINTS),rdot(NJOINTS),Adiag(NJOINTS);

    // Obtain the derivative of the task vector and the activation values, stored in 1D vectors.
    safetyLoop(rdot,Adiag);
    MatrixXd A = Adiag.asDiagonal(); // store the diagonal of A into a sparse matrix structure (needed
   for the following computations)
```

```

    // For the safety task, the Jacobian is the identity and Q is the zero matrix.
    double cond;
    MatrixXd pinvJ = regPinv(ID_MATRIX_NJ,A,ID_MATRIX_NJ,ETA,cond);
    MatrixXd Q1 = ID_MATRIX_NJ - pinvJ; // Q1 will be needed for the tracking task.
    partial_qdot = pinvJ * pinvJ * rdot;

    // Store Q2 into a 1D vector so that it can be sent to the client in a Float64MultiArray object.
    Map<MatrixXd> Q1v (Q1.data(), NJOINTS*NJOINTS,1);

    // Fill response object.
    res.qdot.velocity = vector<double> (partial_qdot.data(), partial_qdot.data() + partial_qdot.size());
    res.Q1.data = vector<double> (Q1v.data(), Q1v.data() + Q1v.size());

    return true; // request successful
}
```

8.25.2.2 cos_sigmoid()

```
double cos_sigmoid (
            double x,
            double y,
            double mrgn )
```

Cosinoidal sigmoid function departing from zero at y+/-mrgn and gets to 1 at y, with period 2∗mrgn.

Parameters

x	Point at which the sigmoid is evaluated.
y	Point at which the sigmoid gets to 1.
mrgn	Half the sigmoid's period.

Returns

the sigmoid value at x.

Definition at line 69 of file safety.cpp.

```
69 {
70     return (.5 + .5 * cos((x - y) * M_PI/mrgn));
71 }
```

8.25.2.3 jointConstr()

```
bool jointConstr (
            double x,
            const double xmin,
            const double xmax,
            const double mrgn,
            const double cPole,
            double & currentPole,
            const bool isJoint,
            double & rdot,
            double & Adiag )
```

Function that evaluates task element derivative and activation value for quantity x.

Parameters

x	Scalar quantity.
xmin	Min value of x.
xmax	Max value of x.
mrgn	Soft margin of x.
cPole	Absolute value of correction pole.
currentPole	Current correction pole for x, if any, 0 otherwise.
isJoint	True if x is a joint angle, false otherwise.
rdot	Reference to task reference derivative for quantity x.
Adiag	Reference to activation value for quantity x.

Returns

true if no correction needed to behaviour of x, false otherwise.

Definition at line 87 of file safety.cpp.

```
89                                                                            {
90     bool ok = false;
91 
92     if (x > xmax - mrgn) { // x too high
93         //cout « "too high" « x « ">" « xmax - mrgn « ", is joint = " « isJoint « endl;
94         if (currentPole == 0) currentPole = cPole; // if the task has just been activated, assign pole for the task
95         if (isJoint) rdot = currentPole * (-x + xmax - mrgn);
96         else rdot = currentPole * (-x + xmax - mrgn) * DT + x;
97         if (x > xmax) Adiag = 1; // x over the limit, task activation value = 1 (full activation)
98         else Adiag = cos_sigmoid(x,xmax,mrgn); // task activation value is between 0 and 1
99     }
100     else if (x < xmin + mrgn) { // x too low
101         //cout « "too low" « x « "<" « xmin + mrgn « ", is joint = " « isJoint « endl;
102         if (currentPole == 0) currentPole = cPole; // if the task has just been activated, assign pole for the task
103         if (isJoint) rdot = currentPole * (-x + xmin + mrgn);
104         else rdot = currentPole * (-x + xmin + mrgn) * DT + x;
105         if (x < xmin) Adiag = 1; // x under the limit, task activation value = 1 (full activation)
106         else Adiag = cos_sigmoid(x,xmin,mrgn); // task activation value is between 0 and 1
107     }
108     else { // x has a safe value
109         Adiag = 0; rdot = 0;// task inactive
110         ok = true;
111         currentPole = 0; // task is inactive, thus no pole is necessary
112     }
113     return ok;
114 }
```

8.25.2.4 main()

```
int main (
            int argc,
            char ** argv )
```

Main function of the node.

Definition at line 183 of file safety.cpp.

```
183                                   {
184 
185     ros::init(argc, argv, "safety_server"); // initialize node
186     ros::NodeHandle n; // define node handle
187     int queSize = 10;
188 
189     ros::Subscriber sub1 = n.subscribe("logtopic", queSize, safetyCallbackq); // subscribe to integrator
190     ros::Subscriber sub2 = n.subscribe("cmdtopic", queSize, safetyCallbackqdot); // subscribe to weighter
```

```

    ros::ServiceServer service = n.advertiseService("safety", computePartialqdot); // activate safety
   service

    ros::spin(); // keep node alive
    return 0;
}
```

8.25.2.5 safetyCallbackq()

```
void safetyCallbackq (
            const sensor_msgs::JointState & msg )
```

Callback function for joint angles.

Parameters

msg	The received joint angles vector.

Definition at line 38 of file safety.cpp.

```
38                                                        {
39      seqtry = (int)msg.effort[0]; // q sequence number
40      if (seqtry < seqqdot) return; // the received q is old, nothing to do with it
41
42      // At this point, the received q is up to date: store it in global variable q.
43      vector<double> rcvd_q = msg.position; // should look at the position field, but Coppelia has a
        problem with it...
44      std::copy(rcvd_q.begin(), rcvd_q.end(), q);
45      readyq = true; // q available, thus set flag to true
46 }
```

8.25.2.6 safetyCallbackqdot()

```
void safetyCallbackqdot (
            const sensor_msgs::JointState & msg )
```

Callback function for joint velocities.

Parameters

msg	The received joint velocities vector.

Definition at line 53 of file safety.cpp.

```
53                                                           {
54      // Store data in global variable qdot.
55      vector<double> rcvd_qdot = msg.velocity;
56      std::copy(rcvd_qdot.begin(), rcvd_qdot.end(), qdot);
57
58      seqqdot = (int)msg.effort[0]; // update sequence number
59      readyqdot = true; // qdot available, thus set flag to true
60 }
```

8.25.2.7 safetyLoop()

```
void safetyLoop (
            VectorXd & rdot,
            VectorXd & Adiag )
```

Function that computes the derivative of the task vector and the diagonal of the activation matrix

Parameters

rdot	Vector passed by reference; on return it will contain the derivative of the task vector.
Adiag	Vector passed by reference; on return it will contain the elements of the diagonal of the activation matrix A.

Definition at line 123 of file safety.cpp.

```
123                                                 {
124     bool angleOk;
125     double rdot_i,Adiag_i;
126     for(short i = 0; i<NJOINTS; i++) { // for all joints
127         // Compute and store the i-th element of rdot and of the diagonal of A, based on the angle constraint task.
128         angleOk = jointConstr(q[i],QMIN[i],QMAX[i],JOINTS_MARGIN,CORRPOLE_POS,currentAnglePole[i],true,rdot_i,Adiag_i);
129 
130         // If joint i has a safe rotation angle, check for the safety of its velocity. Otherwise, the angle correction task will automatically bring the joint velocity to 0.
131         if (angleOk) {
132 
133             // Compute and store the i-th element of rdot and of the diagonal of A, based on the velocity constraint task.
134             jointConstr(qdot[i],-QDOTMAX[i],QDOTMAX[i],VEL_MARGIN,CORRPOLE_VEL,currentVelPole[i],false,rdot_i,Adiag_i);
135         }
136         // Update i-th element of the output vectors.
137         rdot(i) = rdot_i;
138         Adiag(i) = Adiag_i;
139     }
140 }
```

8.25.3 Variable Documentation

8.25.3.1 currentAnglePole

```
double currentAnglePole[NJOINTS]
```

Array whose i-th element keeps the current position correction pole for joint i if joint i is being brought back from being close to a joint limit, 0 otherwise.

Definition at line 15 of file safety.cpp.

8.25.3.2 currentVelPole

```
double currentVelPole[NJOINTS]
```

Array whose i-th element keeps the current position correction pole for joint i if joint i is being slowed down because it was close to its saturation velocity, 0 otherwise.

Definition at line 17 of file safety.cpp.

8.25.3.3 q

```
double q[NJOINTS]
```

Array of joint angle values.

Definition at line 19 of file safety.cpp.

8.25.3.4 qdot

```
double qdot[NJOINTS]
```

Array of joint velocity values.

Definition at line 21 of file safety.cpp.

8.25.3.5 readyq

```
bool readyq
```

Availability flag for joint angles vector: true iff a valid q vector is available.

Definition at line 11 of file safety.cpp.

8.25.3.6 readyqdot

```
bool readyqdot
```

Availability flag for joint velocities vector: true iff a valid qdot vector is available.

Definition at line 13 of file safety.cpp.

8.25.3.7 seqqdot

```
int seqqdot
```

Definition at line 33 of file safety.cpp.

8.25.3.8 seqtry

```
int seqtry
```

At each step, the sequence number of the received q message, for synchronization.

Definition at line 31 of file safety.cpp.

8.26 Math/math_pkg/src/utilities.h File Reference

```
#include <fstream>
#include <iostream>
#include <vector>
#include "cmath"
#include "Eigen/Dense"
#include "Eigen/SVD"
#include "math_pkg/Cost.h"
```

Macros

- #define DT 0.01
- #define ETA 5
- #define NJOINTS 7
- #define NUM_IK_SERVICES 3
- #define NUM_IK_SOLUTIONS 6
- #define NUM_OPTIMIZED_SOLUTIONS 4
- #define SPACE_DOFS 6

Functions

- MatrixXd mypinv (MatrixXd A)
- MatrixXd regPinv (MatrixXd X, MatrixXd A, MatrixXd Q, double eta, double &cond)
- void printVectord (vector< double > v, char *name)
- void printArrayd (double v[ ], int size, char name[ ])
- void saturate (vector< double > &qdots)

Variables

- const double QMIN [ ] = {-1.6817,-2.1268,-3.0343,-0.3,-3.0396,-1.5508,-3.0396}
- const double QMAX [ ] = {1.6817,1.0272,3.0343,2.5829,3.0378,2.0744,3.0378}
- const double QDOTMIN [ ] = {-1,-1,-1,-1,-1,-1,-1}
- const double QDOTMAX [ ] = {1,1,1,1,1,1,1}
- const double QINIT [ ] = {0,0,0,0,0,0,0}
- const double bellConstantGX = -log(0.5)/0.000001
- const MatrixXd ID_MATRIX_NJ = MatrixXd::Identity(NJOINTS,NJOINTS)
- const MatrixXd ID_MATRIX_SPACE_DOFS = MatrixXd::Identity(SPACE_DOFS,SPACE_DOFS)
- const VectorXd ONES_VEC_NJ = VectorXd::Constant(NJOINTS,1)
- const MatrixXd ZERO_MATRIX_NJ = MatrixXd::Zero(NJOINTS,NJOINTS)
- int seq = 0
- bool stay_still = false

8.26.1 Macro Definition Documentation

8.26.1.1 DT

```
#define DT 0.01
```

Simulation timestep.

Definition at line 16 of file utilities.h.

8.26.1.2 ETA

```
#define ETA 5
```

Auxiliary parameter for pseudoinversion.

Definition at line 18 of file utilities.h.

8.26.1.3 NJOINTS

```
#define NJOINTS 7
```

Number of robot joint.

Definition at line 20 of file utilities.h.

8.26.1.4 NUM_IK_SERVICES

```
#define NUM_IK_SERVICES 3
```

Number of invkin services.

Definition at line 22 of file utilities.h.

8.26.1.5 NUM_IK_SOLUTIONS

```
#define NUM_IK_SOLUTIONS 6
```

Number of invkin solutions (J transpose + analytical + 4 from Cost service).

Definition at line 24 of file utilities.h.

8.26.1.6 NUM_OPTIMIZED_SOLUTIONS

```
#define NUM_OPTIMIZED_SOLUTIONS 4
```

Number of optimized invkin solutions (the 4 from Cost service).

Definition at line 26 of file utilities.h.

8.26.1.7 SPACE_DOFS

```
#define SPACE_DOFS 6
```

Spatial dofs of the robot.

Definition at line 28 of file utilities.h.

8.26.2 Function Documentation

8.26.2.1 mypinv()

```
MatrixXd mypinv (
             MatrixXd A )
```

Pseudoinverse of matrix A. Since the pseudoinverse routine of the Eigen library is unstable (see https://eigen.tuxfamily.org/dox/classEigen_1_1CompleteOrthogonalDecomposition.html), this function was coded following MATLAB's pinv's source code.

Parameters

A	matrix to be pseudoinverted

Returns

the pseudoinverse of A.

Definition at line 61 of file utilities.h.

```
61                              {
62     JacobiSVD<MatrixXd> svd(A, ComputeThinU | ComputeThinV);
63     VectorXd s2 = svd.singularValues();
64     double tol = 1e-16; // threshold for singular values
65     int cnt = 0;
66     int s2sz = s2.size();
67     for (int i = 0; i < s2sz; i++) {
68         if (s2(i) > tol) cnt++;
69         else break;
70     }
71 
72     // Throw away portions of U, V related to null singular values of A.
73     VectorXd s2rev = s2.head(cnt);
74     MatrixXd U = svd.matrixU();
```

```
    MatrixXd Urev = U.block(0,0,U.rows(),cnt);
    MatrixXd V = svd.matrixV();
    MatrixXd Vrev = V.block(0,0,V.rows(),cnt);

    // Compute pseudoinverse.
    s2rev = s2rev.cwiseInverse();
    MatrixXd s2diag = s2rev.asDiagonal();
    MatrixXd pinvA = Vrev * s2diag * Urev.transpose();

    return pinvA;
}
```

8.26.2.2 printArrayd()

```
void printArrayd (
            double v[],
            int size,
            char name[] )
```

Function that prints an array of doubles, used for debug.

Parameters

v	A std::vector of doubles.

Definition at line 130 of file utilities.h.

```
                                                    {
    clog « name « ":" « endl;
    for (int i = 0; i < size; i++) {
        clog « v[i] « ",";
    }
    clog « endl;
}
```

8.26.2.3 printVectord()

```
void printVectord (
            vector< double > v,
            char * name )
```

Function that prints a std::vector of doubles, used for debug.

Parameters

v	A std::vector of doubles.

Definition at line 119 of file utilities.h.

```
                                                 {
    clog « name « ":" « endl;
    for (int i = 0; i < v.size(); i++) {
        clog « v[i] « ",";
    }
    clog « endl;
}
```

8.26.2.4 regPinv()

```
MatrixXd regPinv (
            MatrixXd X,
            MatrixXd A,
            MatrixXd Q,
            double eta,
            double & cond )
```

Function that computes generalized regularized pseudoinverse of matrix X according to the paper "A Novel Practical Technique to Integrate Inequality Control Objectives and Task Transitions in Priority Based Control" by Casalino & Simetti, p. 20, sec. 4.3.

Parameters

X	Matrix to be pseudoinverted.
A	Activation matrix.
Q	Auxiliary matrix.
eta	Auxiliary parameter.

Returns

the regularized pseudoinverse of X.

Definition at line 96 of file utilities.h.

```
96                                                                              {
97      // Adiag has size NJOINTS
98      MatrixXd XT = X.transpose();
99      MatrixXd idMinusQ = ID_MATRIX_NJ - Q;
100     MatrixXd toSVD = XT*A*X + eta*(idMinusQ.transpose()*idMinusQ);
101
102     JacobiSVD<MatrixXd> svd(toSVD, ComputeThinU | ComputeThinV); // compute SVD
103     VectorXd sv = svd.singularValues();
104     int svsz = sv.size(); // number of singular values
105
106     for (int i = 0; i < svsz; i++) {
107         if (abs(sv(i)) > 1e-8) sv(i) = exp(-bellConstantGX*sv(i)*sv(i)); // bell-shaped regularization function
108         else break;
109     }
110     cond = sv(0) / sv(svsz-1); // condition number
111     return mypinv(toSVD + svd.matrixV().transpose() * sv.asDiagonal() * svd.matrixV()) * XT * A * A;
112 }
```

8.26.2.5 saturate()

```
void saturate (
            vector< double > & qdots )
```

Function that saturates joint velocities.

Parameters

qdots	Vector to be saturated

Definition at line 141 of file utilities.h.

```
                                                   {
    for (short i = 0; i < NJOINTS; i++) {
        if (qdots[i] > QDOTMAX[i]) qdots[i] = QDOTMAX[i];
        else if (qdots[i] < QDOTMIN[i]) qdots[i] = QDOTMIN[i];
    }
}
```

8.26.3 Variable Documentation

8.26.3.1 bellConstantGX

```
const double bellConstantGX = -log(0.5)/0.000001
```

Constant used in Gaussian computation for pseudoinversion.

Definition at line 40 of file utilities.h.

8.26.3.2 ID_MATRIX_NJ

```
const MatrixXd ID_MATRIX_NJ = MatrixXd::Identity(NJOINTS,NJOINTS)
```

Identity matrix of size NJOINTS.

Definition at line 42 of file utilities.h.

8.26.3.3 ID_MATRIX_SPACE_DOFS

```
const MatrixXd ID_MATRIX_SPACE_DOFS = MatrixXd::Identity(SPACE_DOFS,SPACE_DOFS)
```

Identity matrix of size 6.

Definition at line 44 of file utilities.h.

8.26.3.4 ONES_VEC_NJ

```
const VectorXd ONES_VEC_NJ = VectorXd::Constant(NJOINTS,1)
```

Vector of ones of size NJOINTS.

Definition at line 46 of file utilities.h.

8.26.3.5 QDOTMAX

```
const double QDOTMAX[] = {1,1,1,1,1,1,1}
```

Max joint velocities.

Definition at line 36 of file utilities.h.

8.26.3.6 QDOTMIN

```
const double QDOTMIN[] = {-1,-1,-1,-1,-1,-1,-1}
```

Min joint velocities.

Definition at line 34 of file utilities.h.

8.26.3.7 QINIT

```
const double QINIT[] = {0,0,0,0,0,0,0}
```

Initial joint angles.

Definition at line 38 of file utilities.h.

8.26.3.8 QMAX

```
const double QMAX[] = {1.6817,1.0272,3.0343,2.5829,3.0378,2.0744,3.0378}
```

Max joint angles.

Definition at line 32 of file utilities.h.

8.26.3.9 QMIN

```
const double QMIN[] = {-1.6817,-2.1268,-3.0343,-0.3,-3.0396,-1.5508,-3.0396}
```

Min joint angles.

Definition at line 30 of file utilities.h.

8.26.3.10 seq

```
int seq = 0
```

Sequence number used for synchronization.

Definition at line 50 of file utilities.h.

8.26.3.11 stay_still

```
bool stay_still = false
```

Becomes true when error is zero.

Definition at line 52 of file utilities.h.

8.26.3.12 ZERO_MATRIX_NJ

```
const MatrixXd ZERO_MATRIX_NJ = MatrixXd::Zero(NJOINTS,NJOINTS)
```

Zero matrix of size NJOINTS.

Definition at line 48 of file utilities.h.

8.27 Math/math_pkg/src/weighter.cpp File Reference

```
#include <algorithm>
#include <iostream>
#include "math_pkg/Cost.h"
#include "math_pkg/IK.h"
#include "math_pkg/IK_JTA.h"
#include "math_pkg/IK_Jtra.h"
#include "math_pkg/Safety.h"
#include "ros/ros.h"
#include "sensor_msgs/JointState.h"
#include "std_msgs/Float64MultiArray.h"
#include "std_msgs/Int8.h"
#include "utilities.h"
```

Functions

- void handleCallback (const std_msgs::Int8 &msg)
- int getAllqdots (vector< double > qdots[ ], bool obtained[ ])
- int computeWeightedqdot (JointState &finalqdotState)
- int main (int argc, char **argv)

Variables

- ros::ServiceClient clients [NUM_IK_SERVICES]
- math_pkg::Cost costSrv
- math_pkg::IK_Jtra traSrv
- math_pkg::IK_JTA TASrv
- bool reset = false
- bool mustPause = false
- bool moveOn = false
- int bestIdx = 0

8.27.1 Function Documentation

8.27.1.1 computeWeightedqdot()

```
int computeWeightedqdot (
            JointState & finalqdotState )
```

Function that computes the final joint velocities.

Parameters

finalqdotState	Joint state object to be filled in.

Returns

number of retrieved velocities vectors.

Definition at line 119 of file weighter.cpp.

```
{
    vector<double> finalqdot(NJOINTS, 0);        // initialize content of object to be published
    vector<double> qdots[NUM_IK_SOLUTIONS];      // will contain all qdots computed by the invkin services
    bool obt[NUM_IK_SOLUTIONS];                  // i-th element is true if i-th solution was obtained, false otherwise
    int num_obtained = getAllqdots(qdots, obt); // get all computed qdots

    // Select best qdot.
    if (obt[0] && !obt[1] && !obt[2])
    {
        bestIdx = 0;
    }
    else if (obt[1] && !obt[2])
    {
        bestIdx = 1;
    }
    else if (obt[2])
    {
        bestIdx = 2;
    }

    if (num_obtained > 0)
    {
        finalqdot = qdots[bestIdx]; // best qdot assigned
    }
    saturate(finalqdot);
    if (num_obtained > 0)
    {
        finalqdotState.velocity = finalqdot; // store final velocity vector into the velocity field of the object to be published.
```

```
    }
    seq = seq + 1;
    vector<double> eff(1, seq);
    finalqdotState.effort = eff;
    if (isnan(finalqdot[0]))
        num_obtained = -1;
    return num_obtained;
}
```

8.27.1.2 getAllqdots()

```
int getAllqdots (
            vector< double > qdots[],
            bool obtained[] )
```

Function that calls all the invkin modules and retrieves the computed joint velocities.

Parameters

qdots	Vector that will contain the computed qdots, to be filled.
obtained	Vector that at position i contains a boolean that is true if i-th solution was retrieved, false otherwise.

Returns

number of retrieved velocities vectors.

Definition at line 62 of file weighter.cpp.

```
{

    bool costObtained;    // will be true if call to Cost service will succeed.
    int num_obtained = 0; // initialization
    costSrv.request.seq.data = seq;
// Each service call is performed in parallel.
#pragma omp sections
    {
#pragma omp section
        {
            costObtained = clients[0].call(costSrv); // call service Cost
        }
#pragma omp section
        {
            obtained[0] = clients[1].call(traSrv);
        }

#pragma omp section
        {
            obtained[1] = clients[2].call(TASrv);
        }
    }

    if (costObtained)
    { // store solutions obtained from Cost and update num_obtained
        qdots[2] = costSrv.response.qdot1opt1.velocity;
        qdots[3] = costSrv.response.qdot1opt2.velocity;
        qdots[4] = costSrv.response.qdot2opt1.velocity;
        qdots[5] = costSrv.response.qdot2opt2.velocity;
        num_obtained = num_obtained + 4;
    }

    if (obtained[0])
    {
        qdots[0] = traSrv.response.q_dot.velocity;
        num_obtained++;
    }

     if (obtained[1])
     {
         qdots[1] = TASrv.response.q_dot.velocity;
```

```
        num_obtained++;
    }

    for (int i = 2; i < NUM_IK_SOLUTIONS; i++)
    {
        obtained[i] = costObtained;
    }

    return num_obtained;
}
```

8.27.1.3 handleCallback()

```
void handleCallback (
            const std_msgs::Int8 & msg )
```

Callback function for simulation signals. Handles changes in the simulation. If data = 0, then the initial conditions must be resetted. If data = 1, then the algorithm moves on. If data = 2, then the algorithm pauses.

Parameters

msg	The received data.

Definition at line 44 of file weighter.cpp.

```
{
    if (msg.data == 0)
    {
        reset = true;
        moveOn = false;
    }
    else if (msg.data == 1)
        moveOn = true;
    else if (msg.data == 2)
        moveOn = false;
}
```

8.27.1.4 main()

```
int main (
            int argc,
            char ** argv )
```

Main function of the node.

Definition at line 159 of file weighter.cpp.

```
{

    ros::init(argc, argv, "weighter"); // initialize node
    ros::NodeHandle n;                 // define node handle

    int queSize = 10;

    ros::Subscriber sub1 = n.subscribe("handleSimulation", queSize, handleCallback); // subscribe to Jacobian

    ros::Publisher pub = n.advertise<sensor_msgs::JointState>("cmdtopic", queSize); // activate qdot publisher
    ros::Rate loopRate(100);                                                         // define publishing rate

    // This node acts as a client for three services.
    clients[0] = n.serviceClient<math_pkg::Cost>("cost");
```

```
    clients[1] = n.serviceClient<math_pkg::IK_Jtra>("IK_Jtransp");
    clients[2] = n.serviceClient<math_pkg::IK_JTA>("IK_JAnalytic");

    while (ros::ok())
    {
        vector<double> tosendFirst(NJOINTS, 0);
        vector<double> firstEffort(1, 0);
        sensor_msgs::JointState toSend; // initialize object to be published
        toSend.velocity = tosendFirst;
        toSend.effort = firstEffort;

        int obt;

        while (ros::ok())
        {
            if (moveOn)
            {
                obt = computeWeightedqdot(toSend);
                if (obt == -1)
                    break;
                pub.publish(toSend); // publish weighted qdot
            }
            else if (reset)
            {
                reset = false;
                stay_still = false;
                break;
            }
            ros::spinOnce();
            loopRate.sleep();
        }
    }

    return 0;
}
```

8.27.2 Variable Documentation

8.27.2.1 bestIdx

`int bestIdx = 0`

Definition at line 37 of file weighter.cpp.

8.27.2.2 clients

`ros::ServiceClient clients[NUM_IK_SERVICES]`

Client objects needed for service calls.

Definition at line 16 of file weighter.cpp.

8.27.2.3 costSrv

`math_pkg::Cost costSrv`

Cost server object.

Definition at line 19 of file weighter.cpp.

8.27.2.4 moveOn

```
bool moveOn = false
```

Definition at line 34 of file weighter.cpp.

8.27.2.5 mustPause

```
bool mustPause = false
```

Definition at line 31 of file weighter.cpp.

8.27.2.6 reset

```
bool reset = false
```

Definition at line 28 of file weighter.cpp.

8.27.2.7 TASrv

```
math_pkg::IK_JTA TASrv
```

Definition at line 25 of file weighter.cpp.

8.27.2.8 traSrv

```
math_pkg::IK_Jtra traSrv
```

Definition at line 22 of file weighter.cpp.

8.28 README.md File Reference

8.29 Smartphone/compensation/README.md File Reference

8.30 Smartphone/compensation/include/imu_calib/accel_calib.h File Reference

```
#include <Eigen/Dense>
#include <string>
```

Classes

- class imu_calib::AccelCalib

Namespaces

- imu_calib

8.30.1 Detailed Description

Author

Daniel Koch danielpkoch@gmail.com

Class for calculating and applying accelerometer calibration parameters

8.31 Smartphone/compensation/include/imu_calib/apply_calib.h File Reference

```
#include <ros/ros.h>
#include <sensor_msgs/Imu.h>
#include <imu_calib/accel_calib.h>
```

Classes

- class imu_calib::ApplyCalib

Namespaces

- imu_calib

8.31.1 Detailed Description

Author

Daniel Koch daniel.p.koch@gmail.com

Class for applying a previously computed calibration to IMU data

8.32 Smartphone/compensation/include/imu_calib/do_calib.h File Reference

```
#include <ros/ros.h>
#include <sensor_msgs/Imu.h>
#include <string>
#include <vector>
#include <queue>
#include <imu_calib/accel_calib.h>
```

Classes

- class imu_calib::DoCalib

Namespaces

- imu_calib

8.32.1 Detailed Description

Author

Daniel Koch daniel.p.koch@gmail.com

Class for performing IMU calibration

8.33 Smartphone/compensation/src/accel_calib/accel_calib.cpp File Reference

```
#include "imu_calib/accel_calib.h"
#include <yaml-cpp/yaml.h>
#include <fstream>
```

Namespaces

- imu_calib

8.33.1 Detailed Description

Author

Daniel Koch danielpkoch@gmail.com

Class for calculating and applying accelerometer calibration parameters

8.34 Smartphone/compensation/src/apply_calib.cpp File Reference

```
#include "imu_calib/apply_calib.h"
```

Namespaces

- imu_calib

8.34.1 Detailed Description

Author

Daniel Koch daniel.p.koch@gmail.com

Class for applying a previously computed calibration to IMU data

8.35 Smartphone/compensation/src/apply_calib_node.cpp File Reference

```
#include <ros/ros.h>
#include "imu_calib/apply_calib.h"
```

Functions

- int main (int argc, char **argv)

8.35.1 Detailed Description

Author

Daniel Koch < daniel.p.koch@gmail.com

Node applies a previosly computed calibration to imu data

8.35.2 Function Documentation

8.35.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

Definition at line 46 of file apply_calib_node.cpp.

```
{
  ros::init(argc, argv, "apply_calib");

  imu_calib::ApplyCalib calib;
  ros::spin();

  return 0;
}
```

8.36 Smartphone/compensation/src/do_calib.cpp File Reference

```
#include "imu_calib/do_calib.h"
```

Namespaces

- imu_calib

8.36.1 Detailed Description

Author

Daniel Koch daniel.p.koch@gmail.com

Class for performing IMU calibration

8.37 Smartphone/compensation/src/do_calib_node.cpp File Reference

```
#include <ros/ros.h>
#include "imu_calib/do_calib.h"
```

Functions

- int main (int argc, char **argv)

8.37.1 Detailed Description

Author

Daniel Koch danielpkoch@gmail.com

Node performs accelerometer calibration and writes parameters to data file

8.37.2 Function Documentation

8.37.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

Definition at line 46 of file do_calib_node.cpp.

```
{
  ros::init(argc, argv, "do_calib");

  imu_calib::DoCalib calib;
  while (ros::ok() && calib.running())
  {
    ros::spinOnce();
  }

  return 0;
}
```

8.38 Smartphone/smartphone/scripts/clipping.py File Reference

Namespaces

- clipping

Functions

- def clipping.dataFileInitializer ()

  Function used initialize the files in which sensor data will be stored; it will generate three files in the 'output' folder.
- def clipping.lin_acc_compensate (lin_acc_no_g, threshold)

  Function used to filter unwanted minimal incoming data fluctuations, due to noise as well as human operator shake.
- def clipping.storeDataInFiles (fileName, modality, data)

  Function which stores incoming data into .csv files @params fileName name of the generated file @params modality parameter requested by the open() function: 'a' stands for 'open for appending at the end of the file without truncating it' @params data data to be inserted in the .csv file (orientation/linear acceleration/angular velocity)
- def clipping.callback (data)

  This is the callback function: it is invoked every time there is incoming data and has the duty of calling all the previously mentioned functions as well as eulerAnglesToRotationMatrix().
- def clipping.listener ()

  The listener is used to instantiate the homonymous node and to subscribe to the android/imu_corected topic, from which it receives incoming data of smarthpone accelerometers.
- def clipping.talker (msg)

  The talker is used to publish the refined data on the so called 'smartphone' topic.

Variables

- clipping.script_dir = os.path.dirname(__file__)
- string clipping.rel_path1 = "../output/lin_acc.csv"
- clipping.abs_file_path1 = os.path.join(script_dir, rel_path1)
- string clipping.rel_path2 = "../output/orientation.csv"
- clipping.abs_file_path2 = os.path.join(script_dir, rel_path2)
- string clipping.rel_path3 = "../output/angVel.csv"
- clipping.abs_file_path3 = os.path.join(script_dir, rel_path3)
- string clipping.rel_path_gravity = "../calibration/gravity.csv"
- clipping.abs_file_gravity = os.path.join(script_dir, rel_path_gravity)
- int clipping.flagWriteData = 1
- int clipping.index = 1
- float clipping.delta = 0.05
- list clipping.lin_acc_no_g = [0, 0, 0]
- list clipping.angular_velocity = [0, 0, 0]
- list clipping.orientation = [0, 0, 0, 0]
- list clipping.g = [0 ,0 ,0]
- list clipping.max_lin_acc = [0, 0, 0]
- float clipping.safety_coeff = 1.1
- int clipping.counter = 0
- int clipping.maxIteration = 50
- clipping.temp = pd.read_csv(abs_file_gravity, names=['X', 'Y', 'Z'], header=0, decimal=',')

8.39 Smartphone/smartphone/scripts/computeGravity.py File Reference

Namespaces

- computeGravity

Functions

- def computeGravity.dataFileInitializer ()

  Function used initialize the files in which data will be stored; it will be stored in the calibration folder.
- def computeGravity.callback (data)

  This is the callback function: it is invoked every time there is incoming data and has the duty of calling all the previously mentioned functions as well as eulerAnglesToRotationMatrix().
- def computeGravity.listener ()

  The listener is used to instantiate the homonymous node and to subscribe to the android/imu_corrected topic, from which it receives incoming data of smarthpone accelerometers.

Variables

- computeGravity.script_dir = os.path.dirname(__file__)
- string computeGravity.rel_path_gravity = "../calibration/gravity.csv"
- computeGravity.abs_file_path_gravity = os.path.join(script_dir, rel_path_gravity)
- int computeGravity.counter = 0
- float computeGravity.sum_x = 0.0
- float computeGravity.sum_y = 0.0
- float computeGravity.sum_z = 0.0
- float computeGravity.delta = 0.05
- float computeGravity.safety_coeff = 1.1
- list computeGravity.max_lin_acc = [0, 0, 0]
- list computeGravity.lin_acc_no_g = [0, 0, 0]
- list computeGravity.orientation = [0, 0, 0, 0]
- list computeGravity.g = [0, 0, 0]
- int computeGravity.maxIteration = 50

8.40 Smartphone/smartphone/scripts/offlineAnalysis.py File Reference

Namespaces

- offlineAnalysis

Functions

- def offlineAnalysis.plotData (df, subplt, x_axis, y_lable, titlePlot)

  This function simply plots data into a a graph.

Variables

- offlineAnalysis.script_dir = os.path.dirname(__file__)
- string offlineAnalysis.rel_path1 = "../output/lin_acc.csv"
- offlineAnalysis.abs_file_path1 = os.path.join(script_dir, rel_path1)
- string offlineAnalysis.rel_path2 = "../output/orientation.csv"
- offlineAnalysis.abs_file_path2 = os.path.join(script_dir, rel_path2)
- string offlineAnalysis.rel_path3 = "../output/angVel.csv"
- offlineAnalysis.abs_file_path3 = os.path.join(script_dir, rel_path3)
- dictionary offlineAnalysis.font
- offlineAnalysis.df_linacc
- offlineAnalysis.df_rot
- offlineAnalysis.df_angVel
- offlineAnalysis.t = len(df_linacc.X)
- offlineAnalysis.x_axis = np.arange(start=1, stop=t + 1, step=1)
- offlineAnalysis.figsize
- offlineAnalysis.temp = df_linacc.astype(float)

8.41 Smartphone/smartphone/scripts/removeGravity.py File Reference

Namespaces

- removeGravity

Functions

- def removeGravity.removeGravity (lin_acc, Rot_m, g)

8.42 Smartphone/smartphone/scripts/rotationMatrix.py File Reference

Namespaces

- rotationMatrix

Functions

- def rotationMatrix.eulerAnglesToRotationMatrix (angles)

  Function that transforms euler angle coordinates into the rotation matrix.

8.43 Smartphone/smartphone/unused/src/rotation_matrix_server.py File Reference

Namespaces

- rotation_matrix_server

Functions

- def rotation_matrix_server.eulerAnglesToRotationMatrix (angles)

  Function that transforms euler angle coordinates into the rotation matrix.
- def rotation_matrix_server.anglesCompensate (angles)

  Function used to filter unwanted minimal incoming data fluctuations, due to noise as well as human operator shake.
- def rotation_matrix_server.callback (data)

  This is the callback function: it is invoked every time there is incoming data.
- def rotation_matrix_server.serv_callback ()

  This is the server callback function.
- def rotation_matrix_server.smartphone_server_setup ()

  Setup of the server.

Variables

- float rotation_matrix_server.dx = 0.0174
- list rotation_matrix_server.angles = [0, 0, 0]

8.44 V-rep/baxter_scene/logger.txt File Reference

8.45 V-rep/src/coppelialauncher.sh File Reference

8.46 V-rep/src/Interface.cpp File Reference

```
#include "stdlib.h"
#include "ros/ros.h"
#include "std_msgs/Int8.h"
#include "sensor_msgs/JointState.h"
#include "string.h"
#include "sys/types.h"
#include "signal.h"
```

Functions

- int main (int argc, char **argv)

  The main function acts as a user interface for simulation handling thanks to ROS publishers.

8.46.1 Function Documentation

8.46.1.1 main()

```
int main (
            int argc,
            char ** argv )
```

The main function acts as a user interface for simulation handling thanks to ROS publishers.

The command 'help' is for knowing all the possible interface functions

The command 'start' is for starting the simulation

The command 'calibration' is for making the IMU reference system coincide with the human end effector reference system

The command 'pause' is for pausing the simulation in order to restart from the last reached configuration

The command 'stop' is for stopping the simulation leading the robot arm to the default configuration

The command 'set_default' is for setting a desired default configuration for each one of the 7 joints. In the original default configuration each joint is set to 0

The command 'exit' is for closing the user interface, all the running topics and the simulator environment

Definition at line 35 of file Interface.cpp.

```
{

ros::init(argc, argv, "Test"); //initialize the ros sys, giving it a name

ros::NodeHandle n1; //obj of NodeHandle class

ros::Publisher SimPub = n1.advertise<std_msgs::Int8>("handleSimulation", 1000); //publisher advertisement
ros::Publisher SimStatePub = n1.advertise<sensor_msgs::JointState>("default_state", 1000); //publisher advertisement
ros::Publisher ExitPub = n1.advertise<std_msgs::Int8>("exitSimulation", 1000); //publisher advertisement

char string[20];

ros::Rate loop_rate(10); //loop rate set at 10 Hz
printf("To activate an action, digit your command and press enter.\nFor a list of commands use 'help' and press enter\n");

//while loop -> we want to continue to publish messages, as long the node is alive
while (ros::ok())
{


      //Main body
      printf("Digit your command and press enter: ");
      scanf("%s",string);

      if(strcmp(string,"help")==0){
          printf("- start\n- calibration\n- pause\n- stop\n- set_default\n- exit\n");

```

```
       }else if(strcmp(string,"start")==0){
           std_msgs::Int8 msg;
           msg.data=1;
           //command message publishing
           SimPub.publish(msg);

       }else if(strcmp(string,"calibration")==0){
           std_msgs::Int8 msg;
           msg.data=3;
           //command message publishing
           SimPub.publish(msg);

       }else if(strcmp(string,"pause")==0){
           std_msgs::Int8 msg;
           msg.data=2;
           //command message publishing
           SimPub.publish(msg);

       }else if(strcmp(string,"stop")==0){
           std_msgs::Int8 msg;
           msg.data=0;
           //command message publishing
           SimPub.publish(msg);

       }else if(strcmp(string,"set_default")==0){
           sensor_msgs::JointState msg;
           double array[7];
           printf("Print the joint position you want to set as default:\n");
           msg.position.resize(7);
           for(int i=0;i<7;i++){
               printf("joint %d:",i+1);
                scanf("%lf",&msg.position[i]);
            }
            //command message publishing
            SimStatePub.publish(msg);

        }else if(strcmp(string,"exit")==0){
            std_msgs::Int8 msg;
            msg.data=1;
            //command message publishing
            ExitPub.publish(msg);

            int parent=getppid();
            system("rosnode kill -a");
            kill(parent,SIGKILL);
            break;
        }else{
            printf("Error: mistake in command definition, digit 'help' for a command list\n");
        }

        ros::spinOnce();

    //wait for the next cycle
        loop_rate.sleep();

  }

  return 0;

}
```

8.47 V-rep/src/logger.cpp File Reference

```
#include "stdlib.h"
#include "unistd.h"
#include "stdio.h"
#include <string>
#include <fstream>
#include <iostream>
#include "fcntl.h"
#include "math.h"
#include "ros/ros.h"
#include "std_msgs/Float64.h"
#include "sensor_msgs/JointState.h"
```

Functions

- void logtopicCallback (const sensor_msgs::JointState::ConstPtr &msg)

  The Callback function allows to write both a timestamp and the joint configuration values in the log file.
- int main (int argc, char **argv)

  The main function opens the log file, initiates the ROS subscriber to logtopic topic and calls the Callback function whenever some new data are avaialble.

Variables

- FILE * myfile

  The log file identifier is here declared.

8.47.1 Function Documentation

8.47.1.1 logtopicCallback()

```
void logtopicCallback (
            const sensor_msgs::JointState::ConstPtr & msg )
```

The Callback function allows to write both a timestamp and the joint configuration values in the log file.

Definition at line 42 of file logger.cpp.

```
{
    char st[50];
    memset(st,0,sizeof(st));
    double stamp = msg->header.stamp.toSec();
    sprintf(st,"%f",stamp);

    fprintf(myfile, "%s %lf %lf %lf %lf %lf %lf %lf \n", st, msg->position[0], msg->position[1],
    msg->position[2], msg->position[3], msg->position[4], msg->position[5], msg->position[6]);

}
```

8.47.1.2 main()

```
int main (
            int argc,
            char ** argv )
```

The main function opens the log file, initiates the ROS subscriber to logtopic topic and calls the Callback function whenever some new data are avaialble.

Definition at line 54 of file logger.cpp.

```
{

    myfile = fopen("logger.txt", "w");

    ros::init(argc, argv, "logger"); //initialize the ros sys, giving it a name

    ros::NodeHandle n; //obj of NodeHandle class

    ros::Subscriber sub = n.subscribe("logtopic", 1000, logtopicCallback); //subscriber declaration

    ros::spin();

    fclose(myfile);

    return 0;
}
```

8.47.2 Variable Documentation

8.47.2.1 myfile

```
FILE* myfile
```

The log file identifier is here declared.

Definition at line 38 of file logger.cpp.

8.48 V-rep/src/logger_launcher.sh File Reference

8.49 V-rep/src/publisher_ROS_VREP.cpp File Reference

```
#include "stdlib.h"
#include "ros/ros.h"
#include "sensor_msgs/JointState.h"
```

Functions

- int main (int argc, char **argv)

  The main function initiates the ROS node that publishes some DUMMY input data (7 joint positions) on logtopic topic. These values can be constant or randomly generated depending on the commented portion of the code.

8.49.1 Function Documentation

8.49.1.1 main()

```
int main (
            int argc,
            char ** argv )
```

The main function initiates the ROS node that publishes some DUMMY input data (7 joint positions) on logtopic topic. These values can be constant or randomly generated depending on the commented portion of the code.

Definition at line 32 of file publisher_ROS_VREP.cpp.

```
{

ros::init(argc, argv, "publisher_ROS_VREP"); //initialize the ros sys, giving it a name

ros::NodeHandle n; //obj of NodeHandle class

ros::Publisher chatter_pub = n.advertise<sensor_msgs::JointState>("logtopic", 1000); //publisher
  advertisement

```

```
    ros::Rate loop_rate(110);//loop rate set at 110 Hz

    //while loop -> we want to continue to publish messages, as long the node is alive

    while (ros::ok())
    {
          sensor_msgs::JointState msg; //declare our message variable

          msg.position.resize(7); //set to 7 the array size

      //constant command message (values between -1 and 1)

      msg.position[0] = -0.2;
      msg.position[1] = 0.9;
      msg.position[2] = 0.8;
      msg.position[3] = -0.58;
      msg.position[4] = 1;
      msg.position[5] = -0.3;
      msg.position[6] = 0.04;

      //random command message (values between -1 and 1)

      /*msg.position[0] =((rand()%101)/(float)100 - 0.5)*2;
      msg.position[1] =((rand()%101)/(float)100 - 0.5)*2;
      msg.position[2] =((rand()%101)/(float)100 - 0.5)*2;
      msg.position[3] =((rand()%101)/(float)100 - 0.5)*2;
      msg.position[4] =((rand()%101)/(float)100 - 0.5)*2;
      msg.position[5] =((rand()%101)/(float)100 - 0.5)*2;
      msg.position[6] =((rand()%101)/(float)100 - 0.5)*2;*/

      msg.header.stamp = ros::Time::now();

      //command message publishing
          chatter_pub.publish(msg);

          ros::spinOnce();

      //wait for the next cycle
          loop_rate.sleep();
    }

    return 0;

  }
```