Installation and usage manual for the ERL Benchmarking System

Giulio Fontana, Enrico Piazza,
Martino Migliavacca, Matteo Matteucci

October 25, 2016

IMPORTANT NOTE. The contents of this document are preliminary. Changes and additions to such contents will occur as activities of the RockEU2 project proceed.

Contents

List of Figures

IMPORTANT NOTE. The contents of this document are not definitive. Changes to such contents will occur as activities of the RockEU2 project proceed.

1 Introduction

1.1 About this document

This document is an operative description of the ERL Benchmarking System. The ERL Benchmarking System is a tool, composed of several hardware and software elements, used to produce and process data in order to benchmark the performance of autonomous robots in a consistent and scientifically sound way. A key feature of this benchmarking activity is that it occurs in the context of a robot competition, called **European Robotics League Competition**[1]. The ERL Competition is one of the initiatives of the RockEU2 project[2], financed by the European Commission within the Horizon 2020 programme.

The aim of this document is to provide a user with all the information required to understand, set up, configure and use an instance of the ERL Benchmarking System. In order to use such information, a user must have basic experience in autonomous robotics, while previous experience in robot competitions is not necessary.

The document is organized as follows.

Section 2 is a glossary of acronyms and special terms used throughout the rest of this document.

Section 3 provides an overview of the architecture of the ERL Benchmarking System.

Section 4 explains how to set up the acquisition and logging of robot pose data, which are especially important to benchmark robot tasks involving movement.

Section 5 is a short description of a few simple software tools created to ease the execution of the benchmarks.

Section 6 provides general information about how ERL benchmarks are structured and executed. This section is only concerned with the general elements of ERL benchmarks; all details concerning specific benchmarks are left to Section 7.

[1] http://sparc-robotics.eu/the-european-robotics-league/

[2] http://cordis.europa.eu/project/rcn/199873_en.html

Section 7 is an operative description of specific ERL benchmarks. Only the benchmarks where the ERL Benchmarking System had a role in the scoring process are described; those where the system was only used for ground truth logging are not considered.

Project RockEU2, which is currently ongoing, incorporates the legacy of the RoCKIn project. For this reason, the benchmarks described by Section 6 closely correspond to those used for the 2015 RoCKIn Competition. Project RoCKIn and its relation to project RockEU2 are shortly described in Section 1.2.

1.2 European Robotics League and RoCKIn

European Robotics League, i.e. the competition and benchmarking element of the RockEU2 project, descends from the work of a previous European project, called RoCKIn. A consequence of this is that most of what is described in this document is a direct evolution of the results of RoCKIn. For this reason, additional information about the inception and design of the ERL Benchmarking System can be found in Deliverable 2.1.8 of project RoCKIn[3].

The European Robotics League Competition includes two parallel tracks: one (**ERL Service Robots**, or **ERL-SR**) focusing on service robotics, the other (**ERL Industrial Robots**, or **ERL-IR**) on industrial contexts. This subdivision matches the approach of project RoCKIn, which considered the two different contexts of domestic service robotics and shop-floor industrial applications: as a consequence, RoCKIn Competitions were subdivided into two parts, called RoCKIn@Home and RoCKIn@Work[4]. The current ERL-SR and ERL-IR Competitions closesly match RoCKIn@Home and RoCKIn@Work respectively.

The European Robotics League is a distributed competition taking place over a number of venues and occasions. This is, indeed, the main difference between ERL and RoCKIn: the second focused on large competition events, while ERL (while not excluding large events) also aims at setting up smaller events taking place at local testbeds and involving a small number of teams.

The competition activity of the RockEU2 project also includes a third robot competition, called **ERL-Emergency**. ERL-Emergency is focused on outdoor activity and search-and-rescue applications, and is built on the

[3]This document is available at
http://rockinrobotchallenge.eu/rockin_d2.1.8.pdf

[4]A complete description of RoCKIn@Home and RoCKIn@Work can be found in their Rulebooks, available at
http://rockinrobotchallenge.eu/rockin_d2.1.3.pdf and
http://rockinrobotchallenge.eu/rockin_d2.1.6.pdf respectively.

legacy of former European project Eurathlon. As using the ERL Benchmarking System for ERL-Emergency is not foreseen, this document is only concerned with ERL-SR and ERL-IR; the reader will find information about ERL-Emergency in other documents of the RockEU2 project.

2 Special terms

In the following of this document, special terms, acronyms and abbreviations (most of which come from project RoCKIn: see Section 1.2) will be used. Below is a list of the most frequent.

- **BM** = benchmark
- **FBM** = Functionality Benchmark
- **TBM** = Task Benchmark
- **GT** = Ground Truth
- **ERL** = European Robotics League, i.e. the activity of project RockEU2 concerning robot competitions
- **ERL-SR** = ERL Service Robots Competition
- **ERL-IR** = ERL Industrial Robots Competition
- **Referee Box**, **RefBox**, textbfrefbox = computer which coordinates the execution of a benchmark
- **Benchmarking Box**, **BmBox** = computer which processes benchmarking data
- **mocap** = motion capture
- **marker** = IR-reflective object designed to be easily visible to the mocap cameras
- **marker set** = special and fixed configuration of markers designed to be mounted on a robot to track its pose
- **ROS** = the Robot Operating System[5] middleware for robotics
- **OptiTrack** = the OptiTrack motion capture system produced by Natural Point[6]
- **Motive** = the Natural Point software package for the acquisition of motion capture data

[5] http://wiki.ros.org/
[6] http://www.optitrack.com/products/

3 Overview of the ERL Benchmarking System

As already specified in Subsection 1.2, the version of the ERL Benchmarking System described by this document corresponds to the setup used by RoCKIn at the 2015 Competition of Lisbon, Portugal. For this reason, in the following there will be a large number of references to RoCKIn, and to the RoCKIn@Home and RoCKIn@Work Competitions in particular. This is not strange, because -as already explained in Section **??**- the newer ERL-SR and ERL-IR Competitions closely resemble the older RoCKIn ones. More specifically, what follows assumes the presence of separate testbeds, motion capture systems and benchmarks (due to the separation between RoCKIn@Home/ERL-SR and RoCKIn@Work/ERL-IR).

In the interest of precision, even the specific details (e.g., number of motion capture cameras, IP addresses, ...) of the RoCKIn 2015 setup have been preserved in the description that follows. Though it is not required that such details are precisely replicated in every installation of the ERL Benchmarking System, they have been recorded to provide the reader with a more accurate feeling of what is required to obtain satisfactory performance in a critical real-world situation such as a large competition taking place in a general-purpose venue. Of course the environment of most local ERL installations will be more controlled compared to that situation, and consequently some of the constraints on the setup may likely be relaxed.

3.1 System Architecture

Figure 3.1 highlights the main elements of the ERL Benchmarking System and shows that it is composed of several interconnected subsystems. Greyed-out elements represent external systems which interact with the ERL Benchmarking System but are external to it.

As shown in Figure 3.1, for each of the two benchmarking competitions (RoCKIn@Home and RoCKIn@Work) the system includes three main computers. They are:

1. one computer acquiring motion capture data from special cameras (**mocap@H, mocap@W**) and streaming it to other machines;

2. one computer processing mocap data streamed by the former to extract and log *Ground Truth* (*GT*) pose data (**GTlogger**);

3. one computer managing benchmarks (which, in some cases, also re-

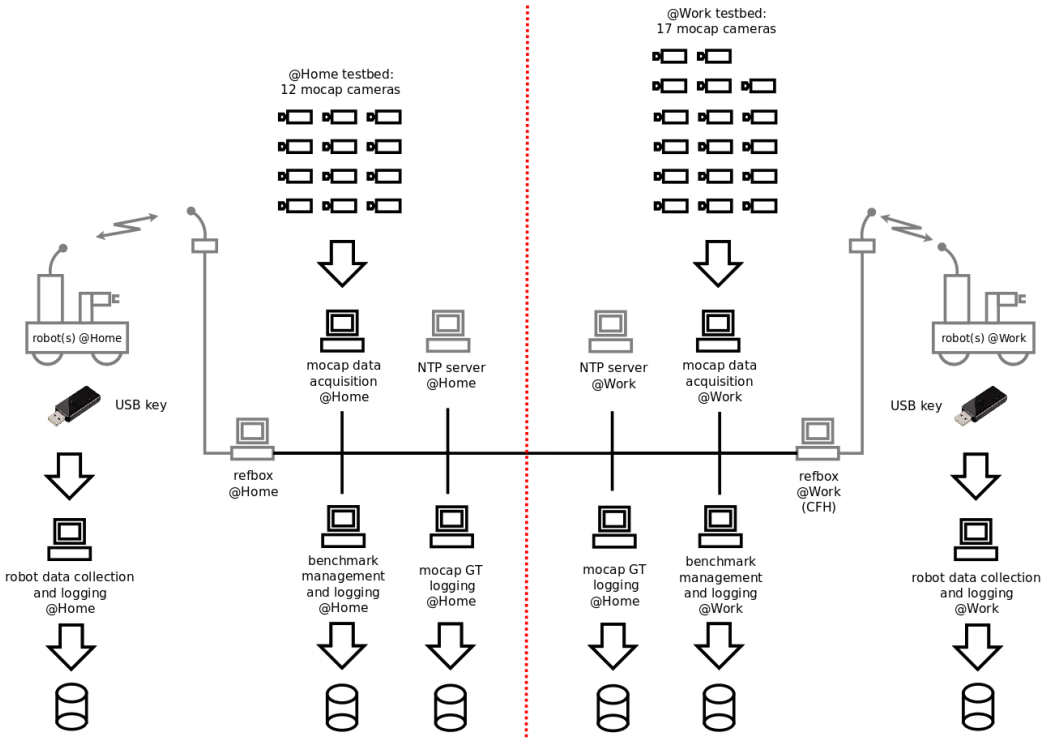


Figure 1: The ERL Benchmarking System. Greyed out elements are external systems that the ERL Benchmarking System interacts with.

quires to receive and process streamed motion capture data) and logging the data related to their execution (**BMmanager**).

One additional computer is used to collect and save robot-generated data, logged by the robots on USB keys (provided by RoCKIn) during the benchmark; these USB keys that are physically brought by the teams to the referees immediately after each benchmark. To operate, the ERL Benchmarking System also needs to interact with external systems. These are:

1. the robot under test;
2. the Referee Box (also called *Central Factory Hub* or *CFH* in RoCKIn-@Work), which organizes competition activities, interfaces with devices belonging to the testbed and interacts with human referees);
3. the NTP server that all other elements synchronize with (see Section 3.4 for additional information).

All the elements of the ERL Benchmarking System interact through TCP/IP networks. These include wireless segments connecting the robots to the rele-

vant Referee Box. If the two networks are connected -as it was at the RoCKIn Competition 2015 in Lisbon- some elements do not need to be duplicated.

At the RoCKIn Competition 2015, the actual machines used were the following:

- mocap@H = Dell Quadcore desktop server
- mocap@W = Shuttle desktop mini-PC
- GTlogger@H, GTlogger@W = HP laptop
- BMmanager@H = Zotac desktop mini-PC
- BMmanager@W = Lenovo laptop provided by the BRSU partner of RoCKIn
- PC used to save robot data = HP laptop
- NTP server for both RoCKIn@Home and RoCKIn@Work = RoCKIn-@Home Referee Box provided by IST[7]
- Referee Box RoCKIn@Home = PC provided by the IST partner of RoCKIn
- Referee Box RoCKIn@Work (Central Factory Hub) = PC provided by the BRSU partner of RoCKIn

3.2 Motion Capture

To be able to benchmark actual robot performance, RoCKIn needs to record exactly what actions the robot actually performs. The information taken as a reference representation of what happened during the benchmarks is generally identified with the name **Ground Truth**, or **GT**. For RoCKIn, the most important category of GT data are those describing robot pose. These are captured by a dedicated commercial *motion capture* (*"mocap"*) system: the OptiTrack system by Natural Point[8]. OptiTrack relies on special infrared cameras to detect the location of IR-reflective markers. A set of at least 3 of these markers having fixed distances between each other can be defined as a *rigid body* in OptiTrack. The system can then detect, track and output the

[7]Though in theory both RoCKIn@Home and RoCKIn@Work were expected to use the same NTP server, for reasons of network topology RoCKIn@Work used the Zotac mini-PC instead, while in turn the Zotac got its time from the RoCKIn@Home Referee Box.

[8]http://www.optitrack.com

6DOF pose of the rigid body: if this is rigidly affixed to a robot component, the same data describe the pose of the component.

A comprehensive description of the RoCKIn motion capture setup, including an in-depth analysis of its real-world performance and limitations, is available in Deliverable 2.1.8 of project RoCKIn (Description of Ground Truth System V2).

To track robots, RoCKIn uses special **marker sets** composed of a base (made of 4mm-thick plywood) fitted with 5 spherical markers, as shown in Figure 3.2. The relative position of the markers on the marker set has been designed chosen to maximize the distances between markers while keeping the marker set compact. Most importantly, marker positions are such that inter-marker distances are all significantly different, in order to minimize ambiguity. Teams are required to mount the marker sets on the top of their robots (to minimize masking by robot parts), in a roughly horizontal orientation, with the arrow-shaped marker set pointing forward according to the robot's own odometry reference system (the shape of the marker set has been chosen to make pointing easy). To facilitate mounting, the marker sets are provided with holes. A CAD file (in *dxf* format) for the shape of the base of the marker sets, useful to manufacture them (e.g., by laser cutting) is available on request.

To get an idea of the dimensions of a marker set, consider that each marker has a diameter of 19 mm, and that the marker set base fits within a circle having a diameter of 170 mm. Additional, special-purpose marker configurations are used for the object recognition Functionality Benchmarks: these will be described in Section 7.

The ERL marker sets just described match those used for the RoCKIn 2015 Competition. It's interesting to point out that these were significantly different from the marker sets used for the 2014 RoCKIn Competitions: the new version, in fact, has all the markers on the same plane. This is deliberate, and has the objective of minimizing masking between markers: in fact, considering the relative positions of mocap cameras and markers during the competition (cameras are significantly higher from the ground than markers) and the fact that teams are required to mount the marker sets on their robots horizontally, critical masking tends to occur only for mocap cameras that are already too far from the marker set to provide useful localization data.

A special marker set (derived from the one used at the RoCKIn Competition 2014) is used for FBM3@Work (Control), and will be described in Section 7.3.

In the setup for an ERL (or RoCKIn) Competition, two separate OptiTrack systems are used. Each of them is dedicated to one of the testbeds used by the Competition, i.e. the ERL-SR (or RoCKIn@Home) testbed and

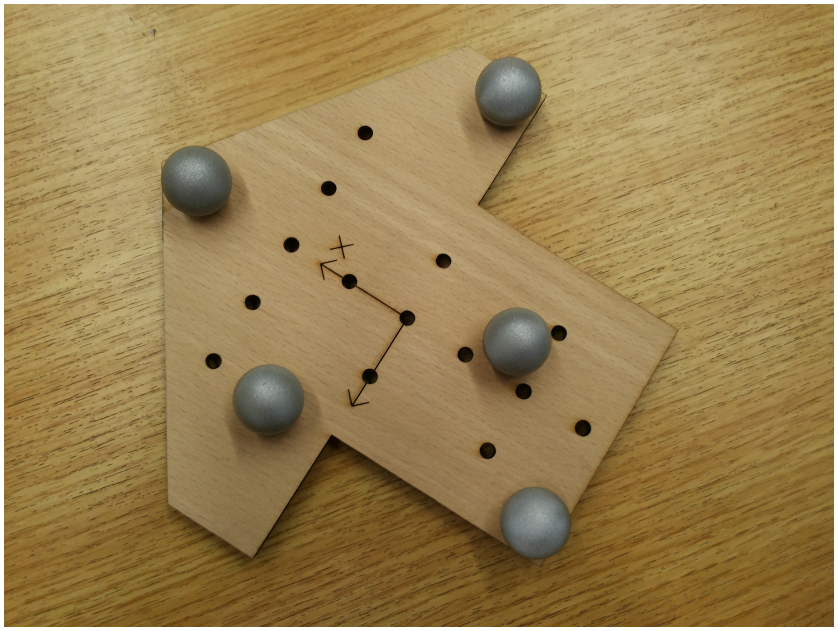


Figure 2: One of the *marker sets* mounted on robots so that they can be tracked by the ERL Benchmarking System.

the ERL-IR (or RoCKIn@Work) testbed. Each OptiTrack motion capture system includes a dedicated Windows PC running the proprietary OptiTrack Motive software, which is required to interface with the cameras and track the marker sets. Motive reconstructs the pose of the marker sets and streams it over UDP to the clients, i.e. the Linux machines that process and log them.

Careful configuration and usage of the Motive software is extremely important for the performance of the ERL Benchmarking System. A guide to optimizing the performance of the mocap system is provided by Section 4.1.

3.3 Networking

Work on this aspect of the ERL Benchmarking System is ongoing. For this reason, no content is currently available for this section.

3.4 Clock synchronization

Clock synchronization among all machines involved in the benchmarking process (including the robots) is required to enable data matching. Such synchronization is provided using the NTP protocol, and by running an NTP

client (chrony [9]) on each machine that require to be synchronized with it. An NTP server provides synchronization data; for the RoCKIn Competition of 2015 the NTP server was the RoCKIn@Home Referee Box, which had IP address 192.168.1.1. Listing 3.4 shows the configuration of Chrony to use on each NTP client in order to sync with the server, supposed to have IP address 192.168.1.1.

```
server 192.168.1.1 minpoll 2 maxpoll 4
initstepslew 2 192.168.1.1

keyfile /etc/chrony/chrony.keys
commandkey 1
driftfile /var/lib/chrony/chrony.drift
maxupdateskew 5
dumponexit
dumpdir /var/lib/chrony
pidfile /var/run/chronyd.pid
logchange 0.5
rtcfile /etc/chrony.rtc
rtconutc
rtcdevice /dev/rtc
sched_priority 1

local stratum 10
allow 127.0.0.1/8

# log measurements statistics tracking rtc
# logdir /var/log/chrony
```

Listing 1: chrony.conf

3.5 Software packages

The GTlogger and BMmanager machines rely on software packages written (except the last one) especially for RoCKIn. These are:

- **rockin_mocap**, which performs logging of motion capture data (on GTlogger and BMmanager)[10];
- **rockin_scoring**, which manages benchmark execution and interfaces with the Referee Box (on BMmanager)[11];

[9] http://chrony.tuxfamily.org/
[10] https://github.com/rockin-robot-challenge/benchmark_and_scoring_mocap
[11] https://github.com/rockin-robot-challenge/benchmark_and_scoring_fbm

- **rockin_acquire_markerset_transform**[12], which acquires the tf transform between the robot's odometric centre and its markerset (on BMmanager);
- **mocap_noise_statistics**[13], provides the measurement mean error on a markerset (on BMmanager);
- **mocap_optitrack**[14], which is used to receive motion capture data streamed by the proprietary Motive software and republish them as ROS *topics* (on GTlogger and BMmanager);

All these packages use the ROS middleware, and have to be available on the both the GTlogger and BMmanager machines. Their location within the filesystem is the directory ~/workspace/ros/src/. Other RoCKIn packages related to benchmarking[15] are available via the repository

https://github.com/rockin-robot-challenge/

In addition to the software packages, a set of *scripts* have been prepared to ease the setup and operation of the ERL Benchmarking System. These will be described in Section 5, and are located in directory ~/workspace/utils/.

[12] https://github.com/rockin-robot-challenge/rockin_acquire_markerset_transform

[13] https://github.com/rockin-robot-challenge/mocap_noise_statistics

[14] http://wiki.ros.org/mocap_optitrack

[15] These additional packages include:

- https://github.com/rockin-robot-challenge/rockin_logger, which contains code and launch files that can be used as examples and templates to implement the logging features required for RoCKIn benchmarking (NOTE: not all the required topics are logged with this code and it must be adapted);
- https://github.com/rockin-robot-challenge/rockin_offline_trajectory_benchmark, which evaluates a score for the FBM2@Home kind of benchmark, comparing the ground truth with the trajectory logged by the robot (i.e. with package https://github.com/rockin-robot-challenge/rockin_logger);
- https://github.com/rockin-robot-challenge/rockin_bags_timeliner, which takes rosbag files in input and outputs the bagfiles ordered by start timestamp, also highlighting the overlaps between bags: this is used to easily associate files produced by different systems having synchronized clocks, such as robots and the benchmarking system;
- https://github.com/rockin-robot-challenge/rockin_test_logs, which include test rosbags, including some that can be useful to test benchmarking without a mocap system.

4 Ground Truth logging

As explained in Section 3, one of the tasks of the ERL Benchmarking System is to log Ground Truth motion capture data. This section is dedicated to describe how these logging activities of GTlogger are performed by the dedicated machines. While these may be more than one (e.g., one per testbed), for simplicity we will suppose that all GT logging is done by a single machine, called GTlogger. Please note that all the filesystem paths in this section refer to this machine. GTlogger includes one or more loggers, each dedicated to a specific category of ground truth information. The loggers can be started and stopped independently.

It's important to note that logged data are *not* those streamed by the OptiTrack motion capture system: such data are received by ROS nodes and the data republished as messages on ROS topics. It's the ROS messages that are logged. For instance, 6DOF pose of the rigid bodies defined in Motive is saved as *tf*[16] data messages. As Section 4.3 will show, actual file preparation and saving is done by the *rosbag*[17] component of ROS, using its own format called "*bag files*" (extension *.bag*).

At the RoCKIn Competition 2014 the loggers were three: one dedicated to the mocap data of the RoCKIn @Home Task Benchmarks, one dedicated to the mocap data of the RoCKIn @Work Task Benchmarks, and the third dedicated to the mocap data of all Functionality Benchmarks, both for RoCKIn @Home and RoCKIn @Work. There is not a one-to-one correspondence between loggers and motion capture systems, as the OptiTrack systems were two while the loggers were three.

At the 2015 RoCKIn Competition the area where motion capture GT data had to be captured and logged were only two: the RoCKIn @Home testbed, for @Home Task and Functionality Benchmarks, and the RoCKIn @Work testbed, for Task and Functionality Benchmarks for @Work. Consequently, as shown by Figure 3.1, two motion capture systems were set up.

It is interesting to note that, beyond the GTlogger machine(s), also the BMmanager machines may log motion capture data. However, this additional logging is focused on the specific needs of the benchmarks: for instance, it may be active only on specific time intervals and/or may capture data related only to some rigid bodies. For each logger the configuration file specifies what elements of the motion capture data will actually be logged.

In order to perform the logging, the user of the GTlogger machine has to execute three subsequent steps:

[16] http://wiki.ros.org/tf
[17] http://wiki.ros.org/rosbag

1. configure the OptiTrack motion capture system;

2. configure the logging PC;

3. run the logging system.

The following of this section describes each of these steps.

4.1 Motion capture optimization

The first step to get stable and high-precision motion capture data to log is to ensure that the OptiTrack system which generates such data is working optimally. For that, it is necessary that the system is configured in the best way for the actual conditions that it must operate in. These conditions depend on environment, installation and system usage; moreover, they have a huge influence on the performance of the OptiTrack system, up to making motion capture impossible. Among the significant aspects are, for instance: lighting, distance between cameras and markers, types and configurations of the markers, ... This section is a short guide to configuring the Motive software to get best performance from it. Please note that these activities also influence the benchmarks that rely on motion capture data for their operation, which will be described in Section 7.

As anticipated, judicious "tweaking" of the operating parameters of the proprietary Motive motion capture software can greatly contribute to make rigid body pose acquisition more precise, stable and reliable. Here follows a list of operations that, in practical experience, proved beneficial:

1. For all cameras, LED lighting should be set at maximum, in order to maximise the ratio of structured-vs-unstructured light in the observed environment: this will usually require that camera exposure is set at a very low value to avoid overexposure. Exposure effect on marker capture can be seen in the camera image ("tracking" view): overexposure tends to inflate and blur marker images (which may also tend to merge for very close markers), while underexposure deforms them by only making visible the brightest regions. Good exposure leads to round and well-defined marker images.

2. Luminance threshold is very critical. It should be set at the highest acceptable level, i.e. the highest which does not deform marker images by obscuring their peripheral (and less bright) regions.

3. If the benchmarks occur in environments where lighting is (partly) natural, expect to have to reset camera exposure (and possibly also

luminance threshold) more than once during a day, according to the criteria listed above.

4. Considering that benchmark environments are quite controlled, no objects which are spuriously similar to the marker sets mounted on the robots should be present. Therefore, it is advisable to increase the probability that the marker set is localized by lowering to 3 the value of parameter "min marker count" that controls how many markers of a rigid body must be localized by the OptiTrack system before it considers to have localized the rigid body.

5. Usually the marker sets include markers that are so close to each other that the peripheral regions of their images tend to overlap (usually this worsens when the marker set is observed from specific angles). In such case, the *circularity filter* of the Motive software may lead to one or more cameras ignoring some of these markers, thus making rigid body identification more difficult. This behaviour is controlled by the value of the "Circularity" parameter (which is found in the "Reconstruction" panel, section "Options"-"2D Object Filter"). This is a threshold: the lower its value, the more "permissive" Motive becomes in recognizing non-perfectly-circular clusters of pixels as marker images. If there is no risk of spurious marker identifications (as should be in the benchmark environments), lowering "Circularity" from its default value of 0.6 to a value as low as 0.2 or 0.15 can make rigid body tracking more reliable.

6. Rigid body properties in Motive include a "Smoothing" parameter, which is usually set to 0 and is best left at such value. Smoothing corresponds to a low-pass filtering of the changes of pose of the rigid body, therefore using a nonzero value leads to smoother trajectories and an increased resilience against spurious oscillations in reconstructed pose of the rigid bodies. This in turn can make rigid body tracking more reliable and stable. However, nonzero values for "Smoothing" should be used with great caution as they lead to Motive interpolating actual data coming from the cameras with computed data which may not correspond with the real motion of the tracked rigid body. In any case, only use very low values for "Smoothing" except for special applications.

7. Carefully monitor tracking performance of the OptiTrack system and repeat its calibration every time it shows signs of degradation. Camera movements (e.g., due to thermal deformation of the supporting structures) can impair reconstructed marker location without catastrophic

disruption of system operation.

Another configuration issue of key importance for motion capture logging, but internal to Motive, is that data streaming is normally deactivated and has to be switched on manually in the Motive software. The correct *multicast* IP to use is 224.0.0.1.

4.2 Preparation and configuration

To prepare the system for logging, the following steps must be followed.

- Via the user interface of the Motive software package, calibrate each of the OptiTrack systems. This includes defining a convenient ground plane and origin (a good choice for the origin is a spot that the robots can easily reach and which is visible to many cameras of the OptiTrack system).
- Via the user interface of the Motive software package, set the working parameters of that software to their optimal value according to the directions of Section 4.1.
- Via the user interface of the Motive software package, define the rigid bodies to be tracked, and assign them unique IDs[18]. These IDs will be used in the configuration files of the logging system to refer to the rigid bodies.
- For each logger, edit the relevant *mocap.yaml* file to match the rigid body IDs to the specific names used in ROS for the relevant reference frames and topics.
- For each logger, edit the relevant *map.yaml* file to configure the map scale and transformation from the *world frame* (i.e., the origin defined in Motive) to the *map frame*. This is necessary to correctly show where the rigid body poses provided by the OptiTrack system are located in the maps.

It is **highly advisable** to assign unique IDs to all rigid bodies used in the RoCKIn Competition, never reusing the same ID twice on different Motive instances running at the same time. This is due to the fact that multiple OptiTrack Motive instances can coexist on the same network and stream data at the same time, but the systems receiving such data have no

[18] In Motive, the IDs assigned to a rigid body corresponds to parameter UserData in section "Advanced" of the rigid body properties.

means to ascertain what instance generated them. Therefore, to avoid any ambiguity it's better to partition the ID space into non-overlapping regions and associate each instance of Motive to one of them. For instance, at the RoCKIn Competition 2015 the IDs assigned to the rigid bodies corresponding to the marker sets and other tracked objects were the following:

rigid body description	ID	smoothing	notes
origin (i.e. table) for FBM1@Work	1	100	physically identical to ID 6
refboard for FBM1@Work	2	20	physically identical to ID 7
robot marker @Home	3	10	physically identical to ID 4
robot marker @Work	4	10	physically identical to ID 3
marker set for FBM3@Work	5	(*)	(*) 15 for line, 20 for spline
origin (i.e. table) for FBM1@Home	6	100	physically identical to ID 1
refboard for FBM1@Home	7	20	physically identical to ID 2

Figure 3: Network configuration for the ERL Benchmarking System.

Column "smoothing" refers to the optional low-pass filtering the Motive can apply to position data. This can be useful to filter out the "trembling" that even stationary objects suffer due to imprecisions in localization, but should be applied very carefully to moving objects as it can introduce overshoots or (even worse) brief time intervals where localization data is still produced for rigid bodies that have actually become untracked.

4.2.1 Configuration files for the acquisition of mocap data

The following listings are the configuration files used for GT logging at the RoCKIn Competition 2015. Please note that - while both files include all the rigid bodies defined for the Competition - those that are not used in the relevant testbed are commented out. Rigid body IDs correspond to those listed in Table 4.2.

4.2.2 Configuration files for the map server

The following listings are the configuration files used by the map servers of the GTlogger machines at the RoCKIn Competition 2015. Setting up a correct map on the GTlogger machine is important mostly for visualization purposes at the time of the Competition: in this way, in fact, the acquired (and logged) mocap poses can be superimposed on the map using the *rviz*[19] visualizer and any problem or inconsistency can be more quickly identified.

[19] wiki.ros.org/rviz

```
rigid_bodies:
#  '1':
#    pose: origin/pose
#    pose2d: origin/pose2d
#    child_frame_id: origin
#    parent_frame_id: world

#  '2':
#    pose: ref_board/pose
#    pose2d: ref_board/pose2d
#    child_frame_id: ref_board
#    parent_frame_id: world

  '3':
    pose: robot_at_home/pose
    pose2d: robot_at_home/pose2d
    child_frame_id: robot_at_home
    parent_frame_id: world

#  '4':
#    pose: robot/pose
#    pose2d: robot/pose2d
#    child_frame_id: robot_at_work
#    parent_frame_id: world

#  '5':
#    pose: robot_at_work/pose
#    pose2d: robot_at_work/pose2d
#    child_frame_id: robot_at_work
#    parent_frame_id: world

  '6':
    pose: origin/pose
    pose2d: origin/pose2d
    child_frame_id: origin
    parent_frame_id: world

  '7':
    pose: ref_board/pose
    pose2d: ref_board/pose2d
    child_frame_id: ref_board
    parent_frame_id: world
```

Listing 2: rockin_mocap/config/all_home_mocap.yaml

```
rigid_bodies:
  '1':
    pose: origin/pose
    pose2d: origin/pose2d
    child_frame_id: origin
    parent_frame_id: world

  '2':
    pose: ref_board/pose
    pose2d: ref_board/pose2d
    child_frame_id: ref_board
    parent_frame_id: world

#  '3':
#    pose: robot_at_home/pose
#    pose2d: robot_at_home/pose2d
#    child_frame_id: robot_at_home
#    parent_frame_id: world

  '4':
    pose: robot/pose
    pose2d: robot/pose2d
    child_frame_id: robot_at_work
    parent_frame_id: world

  '5':
    pose: robot_at_work/pose
    pose2d: robot_at_work/pose2d
    child_frame_id: robot_at_work
    parent_frame_id: world

#  '6':
#    pose: origin/pose
#    pose2d: origin/pose2d
#    child_frame_id: origin
#    parent_frame_id: world

# '7':
#    pose: ref_board/pose
#    pose2d: ref_board/pose2d
#    child_frame_id: ref_board
#    parent_frame_id: world
```

Listing 3: rockin_mocap/config/all_work_mocap.yaml file

```
# Map image file
image: lisbon_home_testbed_map.png

# Resolution of the map, meters / pixel
resolution: 0.01

# The 2-D pose of the lower-left pixel in the map
origin: [-4.03, -4.1, 0]

# Negate white/black free/occupied semantics?
negate: 0

# Occupied/free pixel thresholds
occupied_thresh: 0.65
free_thresh: 0.196
```

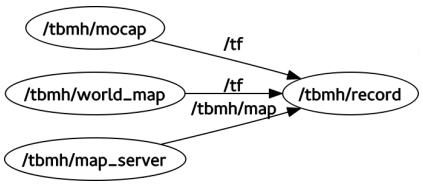
Listing 4: caption

```
# Map image file
image: lisbon_home_testbed_map.png

# Resolution of the map, meters / pixel
resolution: 0.01

# The 2-D pose of the lower-left pixel in the map
origin: [-4.03, -4.1, 0]

# Negate white/black free/occupied semantics?
negate: 0

# Occupied/free pixel thresholds
occupied_thresh: 0.65
free_thresh: 0.196
```

Listing 5: rockin_mocap/config/lisbon_work_testbed_map.yaml

4.3 ROS nodes used for GT logging

Ground truth logging operations on the GTlogger PC are performed by ROS nodes. However, in order to avoid interference among the multiple nodes processing pose data streamed by the two OptiTrack systems, these nodes connect with different instances of *roscore*. Precisely, there is one instance of *roscore* running for each logger. Each instance binds roscore to a different TCP port, to force separation between the different setups. The different environments, with the corresponding roscore instances and TCP ports, are handled by the environment scripts (see Section 5.1).



4.3.1 map_server

This node is a ROS *map_server*[20] node, a C++ node provided by the ROS *map_server* package. It is used to publish a map of the environment, which is described by a bitmap file. In particular, we use this node to stream a map of the testbed, used for inspection with RViz and other ROS tools. The node is started by the ROS launch file for the particular benchmark. The map image name and the corresponding parameters are declared in YAML configuration files stored in the *~/workspace/ros/src/ config* folder.

4.3.2 world_map

This node is a ROS *static_transform_publisher*[21] node, a C++ node provided by the ROS *tf* package. It is used to stream a static transformation between two reference frames. In particular, we use this node to stream the transformation between the absolute reference frame (world) and the map reference frame (map). The node is started by the ROS launch file for the particular benchmark. The transformation between the world reference frame and the map reference frame is fixed to $(0, 0, 0, 0, 0, 0)$ (i.e., null translation and rotation are applied), and the map is located in the correct position by using the origin parameter in the map configuration YAML file.

[20] http://wiki.ros.org/map_server#map_server-1
[21] http://wiki.ros.org/tf#static_transform_publisher

4.3.3 mocap

This node is a ROS *mocap_optitrack*[22] node, a C++ node provided by the ROS *mocap_optitrack* package. It is used to translate motion capture data from an OptiTrack rig to tf transforms, poses and 2D poses. The node receives packets that are streamed by the OptiTrack Motive software, decodes them and broadcasts the poses of configured rigid bodies as tf transforms, poses, and/or 2D poses. The node is started by the ROS launch file for the particular benchmark. The rigid body IDs to be tracked are listed in the YAML configuration files, one for each testbed, named {*tbmh,tbmw,fbm1,fbm3w*}*_mocap.yaml* and stored in the *~/workspace/ros/src/ config* folder.

4.3.4 record

This node is a a ROS *record* node, a C++ node provided by the ROS *rosbag*[23] package. It is used to record data from a running ROS system into a set of .bag files, which are splitted every hour (the split period is specified in the launch files). The node is started by the ROS launch file for the particular benchmark. The list of recored topic depends on the particular benchmark, and it is declared in the ROS launch file. Data is logged using NTP time, so that offline association between GT logs and Teams logs is straightforward.

4.4 Execution of GT logging

This section describes how to run the logging. What follows makes use of the RoCKIn *environments*, which are described in section 5.1. Such section also explains how specific RoCKIn scripts are used to create such environments and to switch among them. The following instructions describe how to run the loggers and their associated monitors (a script that checks that the logfiles increases in size at the expected rate: see 5.2). Usually, two loggers and two monitors will be running at the same time.

To start each logger, first of all prepare the requisite environment. For instance, for the environment used to log RoCKIn @Home GT data you will have to open a new terminal window and -within it- run command

```
$ home_env
```

You can check that the environment is active by observing the system prompt, which now shows an additional string identifying the active environment. In this example, message prompt should now include the string "log@HOME".

[22] http://wiki.ros.org/mocap_optitrack
[23] http://wiki.ros.org/rosbag

Then, in the same terminal window, launch the logging system by calling *roslaunch* and with the relevant *launchfile*, chosen among those provided by ROS package *rockin_mocap*: for instance

```
log@HOME $ roslaunch rockin_mocap home_log_mocap.launch
```

Finally, run the monitor. For this, open a second terminal window and, within it, first launch the command to activate the environment, then run the *log_monitor* script:

```
$ home_env
log@HOME $ log_monitor
```

The log_monitor script automatically adapts its operation (filenames, ...) to the current environment, so there is only one version for all environments.

5 Tools

This short section describes some software tools that have been created to ease the task of the person running the benchmarks. While not necessary, these are handy, especially in situations where speed of execution is important, such as competition events. In the following of this document, descriptions of the execution of benchmarks will make use of the tools introduced here.

5.1 Environments

In order to run multiple logs or benchmarks at the same time, each package must be launched on a different roscore by specifying different roscore's IP address and port. This is done by exporting the variable *ROS_MASTER_URI* before launching the package.

Each environment sets *ROS_MASTER_URI*, *ROCKIN_ENV* (used by log_monitor, see 5.2) and the terminal prefix to different values. The prefix is added to the *bash* shell prompt, so the user can immediately recognize the active environment.

The environments are set by executing the corresponding bash scripts, located in *~/workspace/utils/*. The bash script *~/workspace/utils/ utils/setup.bash* creates an alias for each environment so that they are avilable everywhere. For instance the home environment's alias is set with the command
`alias home_env='source ~/workspace/utils/home_env.sh'`

By adding command *source ~/workspace/utils/ utils/setup.bash* to the *.bashrc* file, these aliases are made available in any terminal window as soon as it is opened.

The commands made available by *~/workspace/utils/ utils/setup.bash* are the following:

- `home_env`, adds the `log@HOME` prefix to bash and exports the following environmental variables:
  `ROCKIN_ENV="log@HOME"`
  `ROS_MASTER_URI="http://localhost:11311"`
- `work_env`, adds the `log@WORK` prefix to bash and exports the following environmental variables:
  `ROCKIN_ENV="log@WORK"`
  `ROS_MASTER_URI="http://localhost:11312"`

During the RoCKIn Competition 2015 there was no need to run multiple benchmarks on the same machine. For this reason, environments where not used on the BMmanagers.

5.2 Log monitor

To make sure that log files are growing as expected, i.e., that the motion capture systems and the ROS nodes are producing data and running without problems, a bash script monitors the log directory and warns the user if something is not going as expected. This includes, in particular, a check on the rate at which the logfiles increase their size, which is required to be higher than a configurable threshold. Whenever the monitor script detects an anomaly, it prints an error message and emits a sound.

The script is called *log_monitor* and is available in the *~/workspace/utils/ utils/* folder. There is a single script covering all the logging tasks of RoCKIn: the script includes several sections (one for each logging task) and executes only the one among them that matches the current environment (see 5.1). Matching depends on the value of environment variable *ROCKIN_ENV*, which must be set beforehand.

The behaviour of the script in correspondence to each environment can be configured by changing the relevant section. For instance, in correspondence to environment log@HOME, the log_monitor script includes the following lines:

```
if os.environ['ROCKIN_ENV'] == 'log@HOME':
    print colors.INFO + "log@HOME environment" + colors.END
    env_prefix = "log@HOME "
    log_prefix = LOG_DIR + "home_log_mocap_"
    log_size_growth = 50 * 1000
```

which define:

- the string to be prepended to the system prompt (as a reminder of the active environment);
- the path and name of the logfile;
- the expected growth rate for the logfile.

To run the monitor script a new terminal has to be opened, the correct environment has to be activated (see section 5.1) and, finally, command *log_monitor* must be run:

```
$ home_env
log@HOME $ log_monitor
```

6 Benchmark structure

European Robotics League benchmarks are subdivided into two categories:

- benchmarks that do not use the ERL Benchmarking System at all (where performance evaluation is performed by human referees, possibly aided by the Referee Box);
- benchmarks that rely on the ERL Benchmarking System to evaluate robot performance and/or to define scoring.

Section 6 and Section 7 are only concerned with the second category. Section 6, in particular, describes the general structure of an ERL benchmark. At the 2015 RoCKIn Competition all benchmarks relying on the ERL Benchmarking System (more precisely, on the similar RoCKIn Benchmarking System) were Functionality Benchmarks: for this reason, in the following the two categories of "benchmarks based on the ERL Benchmarking System " and "Functionality Benchmarks" will sometimes be treated as equivalent. It must be clear to the reader, however, that this is not a general rule: there is no reason, for instance, why future Task Benchmarks may not rely on the ERL Benchmarking System as well.

As already said, the European Robotics League Competition directly descends from the RoCKIn Competition, and closely matches it. In particular, while some ERL benchmarks are expected -in the end- to differ from their RoCKIn counterparts, at the time of writing ERL benchmarks and RoCKIn benchmarks coincide. For this reason, this document will usually refer to the benchmarks with their original RoCKIn-related names. The ERL/RoCKIn Functionality Benchmarks that depend on the motion capture system to be executed and to compute scores are the following:

- FBM1@Home (Object Perception)
- FBM1@Work (Object Perception)
- FBM2@Home (Navigation)
- FBM3@Work (Trajectory Following)

For these benchmarks, the Benchmarking System has to interact with the Referee Box while the benchmark is executed. Data exchange with the Robots is handled by the Referee Box, which is in charge of forwarding messages from and to the Benchmarking System (e.g., to send goals and to receive results).

Actually, the Referee Box has additional duties that do not directly involve the benchmarking system. For instance, right before the execution of each benchmark, the Referee Box authenticates the robot and checks for clock synchronization to guarantee that all data (including those logged by the robot) share the very same time base.

6.1 Benchmark actors

Specific information about how each benchmark is designed and how it is executed will be provided by Section 7. Here we will provide an overview of the execution process of a generic benchmark. As this process is almost the same for all benchmarks, this description is valid for all the benchmarks described by Section 7: any difference or specificity, where present, will be highlighted there.

The following of this section describes the process governing the execution of a generic benchmark. This process requires interaction between three actors:

1. the **BMmanager** computer, also called *Benchmarking Box* or sometimes *BmBox*;

2. the **Referee Box** computer (taking the name of Central Factory Hub in RoCKIn@Work), also called *refbox*;

3. the **Robot** executing the benchmark.

To successfully cooperate to perform the benchmark, the actors need to communicate. As shown by Figure 3.1, the physical medium for such communications is provided by suitable TCP/IP networks.

The BMmanager and the Referee Box use the network(s) to communicate via the ROS middleware, by publishing and receiving messages on ROS topics. To do that, they share a single instance of *roscore*, running on the Referee Box. This is done by configuring the ROS environment variables on the BMManager PC in such a way that ROS nodes launched on this machine connect to the rosnode running on the Referee Box.

For what concerns communications between refbox and robot, it relies on *protobuf*; for this reason, developers of the refbox chose to avoid RPC-style interactions between refbox and BMmanager to employ only the publish/-subscribe messaging paradigm typical of node-level communications in ROS. A module inside the refbox takes care of exchanging and converting data from ROS to protobuf and vice versa whenever interactions between robot and BMmanager is required.

6.2 Benchmark state

The behavior of each of the three actors described in Section 6.1 is governed by a finite state machine. For each machine, state transitions are driven by state updates of the other state machines. In the following, the term **benchmark state** will be used to identify the joint states of the above three interacting finite state machines. The benchmark state is constantly published on the ROS Topic *rockin_benchmark_name/state*, by sending *rockin_scoring/BenchmarkState.msg* messages, at a frequency sufficient to prevent dangerous situations (e.g., the robot not stopping in time). As already specified, these messages are composed of three elements, i.e.:

- state of the Benchmarking Box state machine;
- state of the Referee Box state machine;
- state of the Robot state machine.

The following of this section is dedicated to describing the state machines associated to the benchmarks actors.

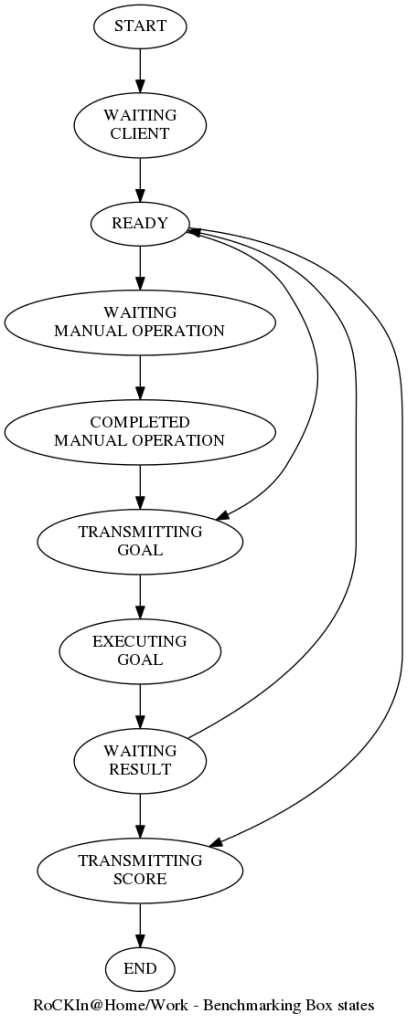
6.2.1 Benchmarking Box state machine

The Benchmarking Box finite state machine has the following states:

- ***WAITING CLIENT***: waiting for a connection from the Referee Box
- ***READY***: everything ready to start the benchmark
- ***WAITING MANUAL OPERATION***: a manual operation from the human referee is needed
  payload contains the requested operation
- ***COMPLETED MANUAL OPERATION***: the referee confirmed that the manual operation was completed
- ***TRANSMITTING GOAL***: a new goal is beeing transmitted by the Benchmarking Box to the robot
  payload: the description of the goal
- ***EXECUTING GOAL***: waiting for the robot to complete the goal
- ***WAITING RESULT***: waiting for the robot to transmit the result to the Benchmarking Box

- ***TRANSMITTING SCORE***: transmtting the final score
  payload: the computed score
- ***END***: the benchmark is concluded

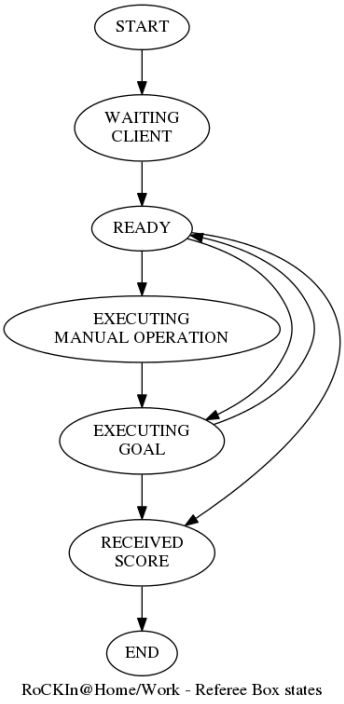
The current state is published on the *fbm_name/bmbox_state* topic, while message content is described by *BmBox.msg* (see Listing 6.2.1). Figure 6.2.1 is a flux diagram showing the transitions between the states.



RoCKIn@Home/Work - Benchmarking Box states

```
uint8 START = 0
uint8 WAITING_CLIENT = 1
uint8 READY = 2
uint8 WAITING_MANUAL_OPERATION = 3
uint8 COMPLETED_MANUAL_OPERATION = 4
uint8 TRANSMITTING_GOAL = 5
uint8 EXECUTING_GOAL = 6
uint8 WAITING_RESULT = 7
uint8 TRANSMITTING_SCORE = 8
uint8 END = 9

uint8 state
string payload
```

Listing 6: BmBoxState.msg



6.2.2 Referee Box state machine

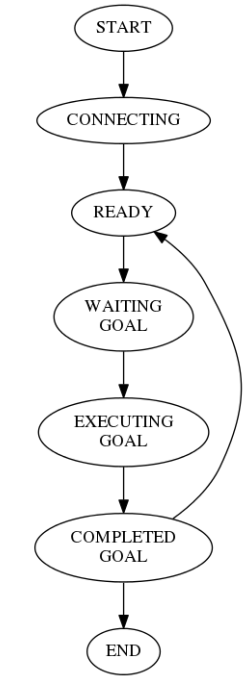
The Referee Box run finite state machine has the following states:

- ***WAITING CLIENT***: waiting for a connection from the Robot
- ***READY***: everything ready to start the benchmark
- ***EXECUTING MANUAL OPERATION***: the manual operation is being executed by a human referee
- ***EXECUTING GOAL***: the robot is executing the goal
- ***RECEIVED SCORE***: received the final score from the robot
- ***END***: benchmark concluded
  payload: if the state transition has been issued by the referee box, payload specifies the reason (timeout / emergency halt)

The current state is published on the *fbm_name/refbox_state* topic, while message content is described by *RefBoxState.msg* (see Listing 6.2.2). Figure 6.2.2 is a flux diagram showing the transitions between the states.

```
uint8 START = 0
uint8 WAITING_CLIENT = 1
uint8 READY = 2
uint8 EXECUTING_MANUAL_OPERATION = 3
uint8 EXECUTING_GOAL = 4
uint8 RECEIVED_SCORE = 5
uint8 END = 6

uint8 state
string payload
```

Listing 7: RefBoxState.msg



RoCKIn@Home/Work - Robot states

6.2.3 Robot state machine

The Robot finite state machine, also called Client, has the following states:

- ***CONNECTING***: attempting to connect to the Referee Box
- ***READY***: everything ready to start the benchmark
- ***WAITING GOAL***: waiting for a goal from the Benchmarking Box
- ***EXECUTING GOAL***: executing the goal
- ***COMPLETED GOAL***: goal completed; the payload contains the result

As will be explained in Section 6.3, the Client state machine does *not* run on the robot; instead, it is a model of the robot's behavior internal to the Referee Box. The current state of the Client is published on the *fbm_name/client_state* topic, while message content is described by *ClientState.msg* (see Listing 6.2.3). Figure 6.2.3 is a flux diagram showing the transitions between the states.

```
uint8 START = 0
uint8 CONNECTING = 1
uint8 READY = 2
uint8 WAITING_GOAL = 3
uint8 EXECUTING_GOAL = 4
uint8 COMPLETED_GOAL = 5
uint8 END = 6

uint8 state
string payload
```

Listing 8: ClientState.msg

6.3 State machine implementation

As already explained, the behavior of the benchmark actors is governed by the finite state machines described in Subsection 6.2. Each of the three state machines is implemented in the form of a ROS node running on one of the PCs managing the benchmarks. Precisely:

1. the state machine of the Referee Box runs on the Referee Box PC;
2. the state machine of the BMmanager runs on the BMmanager PC;
3. the state machine of the Robot runs on the Referee Box PC.

The third state machine is implemented by a ROS node called *Client*. The Client is run on the refbox to maintain control over its evolution, and changes its state according to the communications between robot and refbox. This machine represents the expected behavior of the robot while executing a benchmark. More precisely, the state of the Client represents the information about the actual state of the Robot that the refbox presents to the BMmanager in order to let the latter proceed with the execution of the benchmark. From the point of view of the BMmanager, the client appears as an integral part of the Referee Box system: the state that the BMmanager considers when evaluating if its own state should changes is therefore the joint state of the Referee Box and Robot machines. In principle, the only interactions between state machines that are required to run any benchmark are: interactions between the Referee Box machine and the BMmanager, and interactions between Referee Box machine and Client machine. In practice, for historical reason linked to the way the system evolved over time, each of the three state machines interacts with both the others.

During the development process of new benchmarks on the BMmanager PC, it would be cumbersome to force the presence and participation to the

development process also of the Referee Box and/or the Robot. In order to avoid this need, modified ("dummy") versions of the Referee Box and Robot state machines have been developed, which can be run on the BMmanager. In this way, this is the only machine required for development (for details, please refer to the descriptions of specific benchmarks in Section 7).

6.4 Generic Benchmark Workflow

What follows is a description of the way a generic benchmark is executed, in terms of states of the three benchmark actors, as defined by Section 6.1.

As soon as the benchmark is started (i.e., by launching the corresponding roslaunch file), the Benchmarking Box goes into *WAITING CLIENT* state. When a client connects, the Referee Box authenticates it and checks that clocks are synced; if everything is correct, the Referee Box state updates to *READY*. When the Referee Box state becomes *READY*, bot the Robot state and the Benchmarking Box state are updated to *READY* too.

The robot then can ask for a goal, and its state is updated to *WAITING GOAL*.

If a manual operation is requested (e.g., a human referee must put an object in front of the robot), the Benchmarking Box updates its state to *WAITING MANUAL OPERATION*, sending a description of the required operation as payload of the state message. When the manual operation has been concluded, the Referee Box updates its state to *EXECUTING GOAL*, and the Benchmarking Box goes into *COMPLETED MANUAL OPERATION* state.

The Benchmarking Box can now send the goal to the Client (which represents the robot), going to state *SENDING GOAL* and transmitting the description of the goal as payload of the state message. When the Client receives the goal, it updates its state to *EXECUTING GOAL*, and the Benchmarking Box goes into state *WAITING FOR RESULT* as a consequence.

When the robot completes the goal, it updates its state to *TRANSMITTING RESULT*, with the result as payload of the state message; the Benchmarking Box saves the received results and, if there are additional goals, waits for the robot to request a new goal (i.e., the state is updated to *READY*); otherwise, the state of the Benchmarking Box is updated to *TRANSMITTING SCORE*, with the final score as payload, and the benchmark ends.

At any time during the benchmark the state can be updated to *END* by the Referee Box, by sending either payload "timeout" or payload "halt". The first is used when the timeout for the benchmark is reached; the second when the benchmark is interrupted due to a request by a human referee. In

fact the Referee Box incorporates a timer, and provides ways to let human referees stop an ongoing benchmark if necessary.

7 Benchmark execution

This Section is dedicated to describing in detail specific benchmarks that rely on the ERL Benchmarking System for their execution. As already explained in Section 1.2 and elsewhere in this document, current ERL benchmarks correspond to those used by the 2015 RoCKIn Competition which took place in Lisbon, Portugal: in particular, ERL-SR benchmarks correspond to RoCKIn@Home benchmarks, while ERL-IR benchmarks correspond to RoCKIn@Work benchmarks. Rulebooks with a complete description of the RoCKIn benchmarks are available at

`http://rockinrobotchallenge.eu/rockin_d2.1.3.pdf` and
`http://rockinrobotchallenge.eu/rockin_d2.1.6.pdf`

for RoCKIn@Home and RoCKIn@Work respectively. For the reason stated above, while describing procedures for benchmark execution we will often make references to RoCKIn: the reader must be aware that such references also apply directly to ERL. As project RockEU2 proceeds with its activities, ERL benchmarks may diverge from RoCKIn benchmarks.

As explained in Section 3, the ERL Benchmarking System has two different tasks:

- **logging** Ground Truth motion capture data;
- **benchmarking**, i.e. managing some of the RoCKIn benchmarks (which includes logging the data associated to the benchmark).

The second activity is performed independently from the first, by a dedicated PC (*BMmanager* in the scheme of Figure 3.1). This section is dedicated to describe how the benchmarking activities of BMmanager are performed, and also acts as a "benchmark manual" for operators that have to execute it.

As explained in Section 6, at the RoCKIn Competition 2015 not all the benchmarks were managed in this way: precisely, all of the Task Benchmarks and some of the Functionality Benchmarks were not. This Section focuses only on the benchmarks that strictly required the BMmanager computer for their execution.

Please note that all filesystem paths cited in the following refer to the filesystem of the BMmanager.

7.1 FBM1@Home and FBM1@Work: object recognition

In this benchmark, the robot is required to identify the class, the instance, and the pose of a set of objects. The benchmark works as follows (see the Rule Book for further details):

1. an object of unknown class and unknown instance is placed on a table in front of the robot
2. the robot must determine the objects class, its instance within that class as well as the 2D pose of the object w.r.t. the reference system specified on the table
3. the preceding steps are repeated until time runs out or 10 objects have been processed

For each presented object, the robot must produce the result consisting of:

- object class name [string]
- object instance name [string]
- object pose (x [m], y [m], theta [rad])

This benchmark takes place on a special surface called **FBM1 table**, fitted with AR tags that define the reference system that the robot must use when communicating reconstructed object poses to the Referee Box. Figure 4 shows the FBM1 table: alongside it are set the objects used for FBM1. The figure also shows the reference frame of the FBM1 table as defined by the AR tags.

In order to run the benchmark, the BMmanager computer needs to compare the Ground Truth pose of the object observed by the robot (expressed in the reference frame of the FBM1 table) with the estimate of that pose provided by the robot. To enable this, two special marker sets (different from those mounted on the robots) are used:

- a marker set fixed to the FBM1 table;
- a marker set fixed to a **reference board**.

Figure 4: FBM1 table and reference frame

The function of the reference board is the following. The objects to be recognized by the robot are not placed on the FBM1 table on their own: they are, instead, set on the reference board which is subsequently placed on the FBM1 table. The location of each object on the reference board is repeatable, as the bottom of each object is glued to with a small specially shaped base, designed to fit into a matching cutout in the reference board.

Using the data coming from the mocap system, the BMmanager is able to determine exactly (i.e., with an error that is much smaller than the precision requested from the robot) the poses of the FBM1 table and reference board. Therefore, the pose of the reference board in the reference frame of the FBM1 table is also known to BMmanager. BMmanager also knows the transform between the reference system of the reference board and the reference frame of the object observed by the robot, because it has been acquired and stored during the preparation of the benchmark. By combining this information, BMmanager obtains the Ground Truth pose of the reference frame of the object when expressed in the reference frame of the FBM1 table: i.e., the localization data to be compared to those coming from the robot.

Additional information will be provided in Section 7.1.3.

7.1.1 ROS nodes

Figure 5 shows the ROS node diagram for FBM1. Below is a description of such nodes.

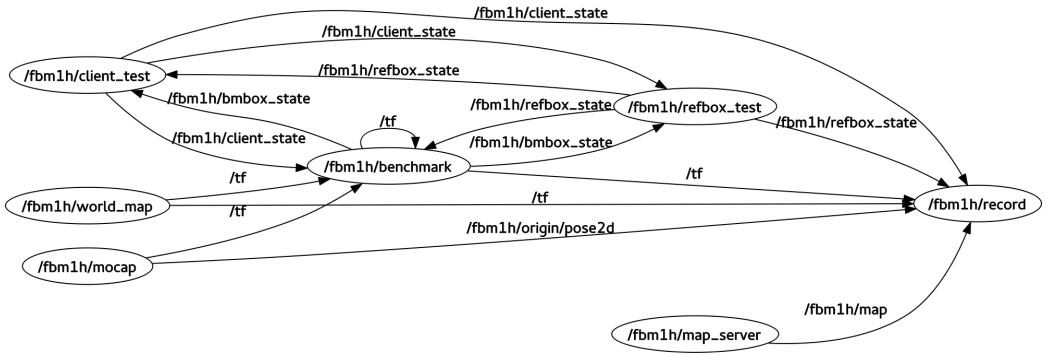


Figure 5: ROS nodes used for FBM1

map_server ROS *map_server* node, as described in Section 4.3.1

world_map ROS *static_transform_publisher* node, as described in Section 4.3.2

mocap ROS *mocap_optitrack* node, as described in Section 4.3.3

record ROS *record* node, as described in Section 4.3.4

benchmark This is the ROS node in charge of running the benchmark. It is developed in Python, and takes care of all the needed tasks:

- runs the benchmark state machine
- loads the object list, with the corresponding transformation from the reference board marker set to the object frame, from a YAML configuration file
- acquires the position of the reference board from the motion capture system
- computes the Ground Truth pose of the object reference frame
- receives the estimate of such pose produced by the robot
- computes the score by comparing Ground Truth and estimate

The only difference between the RoCKIn @Home and the RoCKIn @Work FBM1 is that in @Home the benchmark node is in charge of picking a random object from the object list, while in @Work it is the Referee Box that selects the object.

The script that runs this benchmark is named *fbm1h* (for RoCKIn @Home) or *fbm1w* (for RoCKIn @Work), and the source code is stored in the *rockin_scoring/scripts* folder.

refbox_test [only used for testing] This node simulates the Referee Box state machine. It is developed in Python, and implements the Referee Box state machine to test the interaction with the rest of the system.

The script that simulates the Referee Box is named *fbm1h_refbox_test* and its source code is stored in the *rocking_benchmarking/scripts* folder.

client_test [only used for testing] This node simulates the Robot state machine. It is developed in Python, and implements the Robot state machine to test the interaction with the rest of the system. The script that simulates a Robot is named *fbm1h_client_test* and its source code is stored in the *rocking_benchmarking/scripts* folder.

7.1.2 Object list

The object list is a YAML file representing a Python list of all the objects. The YAML files are named *fbm1h.yaml* (for @Home objects) and *fbm1w.yaml* (for @Work objects), and they are stored in the *rocking_benchmarking/config* folder. The *fbm1h-vanilla.yaml* and *fbm1w-vanilla.yaml* YAML files contain the list of objects without the corresponding translations and rotations, and may be used to acquire the transoformations as described in Section 7.1.4.

For each object, 5 parameters are specified:

- the ID, a unique key starting from 1
- the class
- the instance
- the translation with respect to the reference board frame (expressed in metres)
- the rotation with respect to the reference board frame (expressed in radiants)

```
items:
- class: a
  id: 1
  instance: a1
  rot: [-0.00751, 0.00246, -0.00631, 0.99994]
  trans: [0.01410, -0.25499, -0.01289]
- class: a
  id: 2
  instance: a2
  rot: [-0.00642, 0.00193, -0.00176, 0.99997]
  trans: [0.01642, -0.25131, -0.01429]
```

Listing 9: fbm1h.yaml

7.1.3 Motion capture configuration

The motion capture system has to be carefully configured and calibrated for the correct execution of the benchmark.

First of all, two marker sets (i.e., using the terminology of the Motive software, two *rigid bodies*) have to be defined using Motive:

- the **origin** marker set, composed of the 4 markers fixed to the table (Motive ID 1)
- the **board** marker set, composed of the 5 markers fixed to the reference board (Motive ID 2)

Then, to precisely align the *origin* frame with the AR tags attached to the table, proceed as follows:

1. align the OptiTrack ground plane calibration tool with the AR tags (a replica of the original tool with correct offsets has been made for convenience)
2. in Motive (calibration view), calibrate the ground plane
3. in Motive (creation view), select the *origin* marker set and add an offset to its coordinates so that they coincide with the origin (0, 0, 0, with angle 0)
4. after the calibration, the ground plane is not required to stay on the table any more (i.e., it can be removed, or placed in any other place covered by the Motion Capture System if needed, without compromising the benchmark execution)

7.1.4 Acquisition of objects

To acquire one or more objects, the transformation from the reference board marker set to the frame for each particular object has to be measured and stored. Indeed, the objects may have different frame origins and orientations, depending on their shape. For this reason, when adding objects to the object list (see Section 7.1.2) they must be carefully acquired with the motion capture system to determine the exact transformation.

The recommended setup is to place a camera on the top of the table, pointing at the origin, aligned with the Z axis. Then, display the video stream placing an overlay image showing the 3 axes (e.g., it can be done with VLC[24]), carefully aligning them to the origin. In this way, objects can be precisely placed in the origin, and at the same time images with the superimposed axes can be saved as a reference. A reference frame image is available in the *doc/items* folder. Images of the objects used at the RoCKIn 2015 Competition are stored in the *doc/items/roah* and *doc/items/roaw* folders.

Object acquisition is automated by a ROS script, which cycles the object list and acquires the transoformation for each object. The user is only required to place the objects in the desired position and press ENTER. The result is a YAML configuration file ready to be used for the FBM1; the file is saved in the */home/rockin* folder, and the user is required to overwrite the benchmark config file to use the new one.

The object acquisition script can be launched by calling roslaunch:

```
$ roslaunch rockin_scoring fbm1h_acquire_items.launch
```

7.1.5 Benchmark execution

This section describes how the execution of FBM1 is done in practice. As shown in the following, there are several ways to run FBM1: some of these can be useful while setting up or testing the benchmark.

Executing FBM1 using the Referee Box This is the normal execution mode during an ERL competition. To start the FBM1 using the Referee Box, first of all switch to the correct environment:

```
$ fbmh_env
```

Then, launch the benchmark by calling *roslaunch*:

[24] https://www.vlchelp.com/add-logo-watermarks-over-videos-vlc/

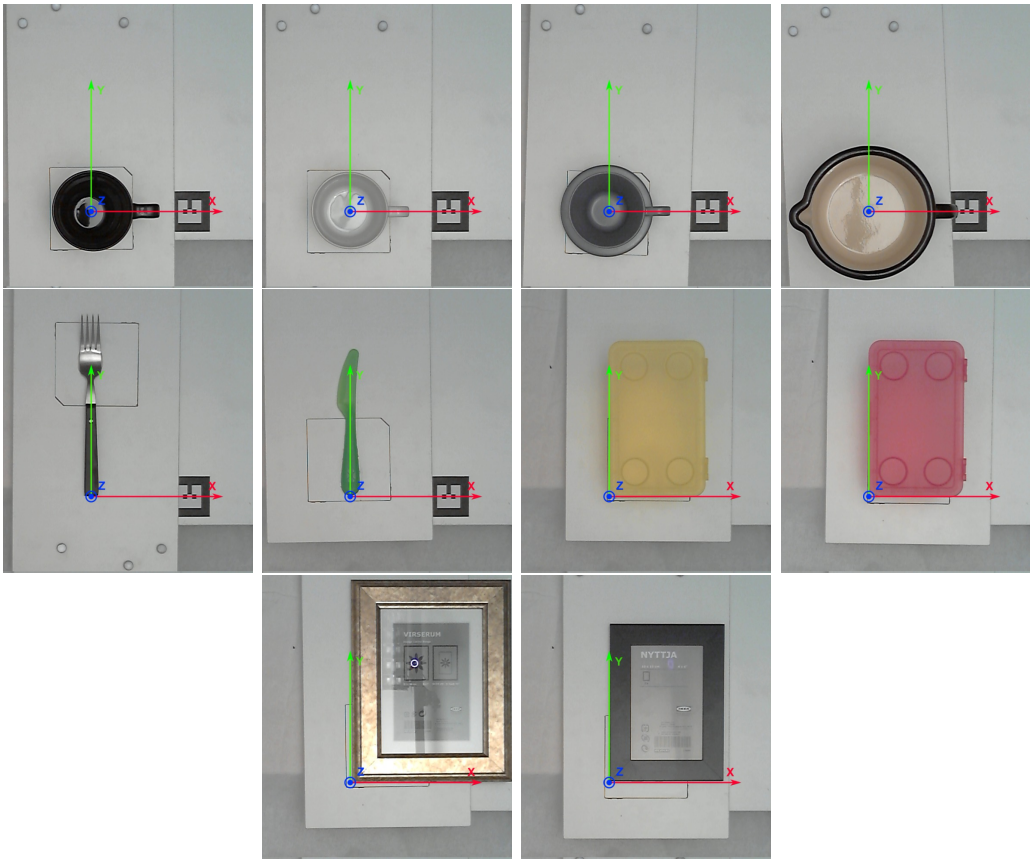


Figure 6: RoCKIn @Home FBM1 objects

```
FBM1@HOME $ roslaunch rockin_scoring fbm1h.launch
```

The benchmark execution is managed by the refbox: whenever a Client (i.e., a robot) is ready, the benchmark node requests a manual operation (i.e., place the given object on the table), sends the goal, and waits for the result. At the end of the runs, the final score is computed and sent to the referee box.

Executing FBM1 without the Referee Box (*manual execution*) It is possible to run FBM1 without requiring that a Referee Box is available. To start FBM1 without the Referee Box, first of all switch to the correct environment:

```
$ fbmh_manual_env
```

Then, launch the benchmark by calling *roslaunch*:

```
FBM1@HOME (MANUAL) $ roslaunch rockin_scoring fbm1h_manual.launch
```

A terminal opens, requesting a manual operation (i.e., place the given object on the table). The referee places the object on the table, and the manual operation is confirmed by pressing ENTER on the terminal. When the robot has recognized the object, press ENTER again to conclude the run. The procedure is repeated for the number of runs, and a report is printed at the end of the benchmark.

Testing A set of python scripts simulating the Referee Box and the Robot state machines can be used to test the FBM1 workflow. To run a test, first of all switch to the correct environment:

```
$ fbmh_manual_env
```

Then, launch the test by calling *roslaunch*:

```
FBM1@HOME (MANUAL) $ roslaunch rockin_scoring fbm1h_test.launch
```

7.1.6 Logging

The benchmark launch files log each execution by launching a rosbag node. The logged topics are:

```
/fbm1h/bmbox_state
/fbm1h/client_state
/fbm1h/refbox_state
/fbm1h/map
/fbm1h/map_metadata
/fbm1h/origin/pose
/fbm1h/origin/pose2d
/fbm1h/ref_board/pose
/fbm1h/ref_board/pose2d
/fbm1h/info
/tf
/rosout
```

Resulting logs are stored in the *˜/logs/* folder.

7.2 FBM2@Home: navigation

This functionality benchmark aims at assessing the capabilities of a robot to correctly and autonomously navigate in a typical apartment, containing furniture and objects spread through the apartments rooms. The benchmark uses the RoCKIn@Home testbed. The robot starts from a predefined starting position; then it receives a list of waypoints that it must visit, in order, to reach finally a goal position. For a complete description of the rules see Deliverable D2.1.3 of project RoCKIn.

During the benchmark, the motion capture system is used to acquire the actual pose of the robot and evaluate its position and orientation accuracy. For this purpose a markerset needs to be mounted on the robot.

7.2.1 ROS nodes

When the benchmark is launched, these ROS nodes are run: *fbm2h*, *world_map*, *map_server*, *mocap*, *record*. Some of these have already been described in Section 7.1.1; among the other, of special interest are the nodes described in the following.

fbm2h This is the ROS node in charge of running the benchmark. Its source code is stored in rockin_scoring/scripts/fbm2h.

The following description defines the operations of the script defining FBM2@Home. In such description, references will be made to the concept of **segment** of the robot's trajectory. The *i-th segment* is defined as the robot's path from waypoint *i-1* to waypoint *i*. Waypoints are numbered, starting from 0, in the order that the robot has to reach them. Segment 0 is the robot's path from the starting pose to waypoint 0.

The benchmark is governed by the script according to the following succession of three phases. Of these, the intermediate one -which refers to one of the trajectory segments- is repeated as required.

It will be specified what states of the Benchmarking Box state machine described in Section 6.2.1 (also called *BmBox*) are involved in each phase.

1. Startup phase. In the start up phase the BmBox goes through the following states:

   - BmBoxState.START
   - BmBoxState.WAITING_CLIENT

   When the benchmark is launched, the robot is in the predefined starting position. The transform from marker set to robot odometric center is

loaded as parameter, for the appropriate team (specified as argument: see Section 7.2.6 for details). This is used to evaluate the robot pose when it reaches a waypoint. The benchmark's parameters are loaded as well (additional information is provided by Section 7.2.3).

2. Execution on each trajectory segment. For each segment, the BmBox goes through the states

   - BmBoxState.READY
   - BmBoxState.TRANSMITTING_GOAL
   - BmBoxState.EXECUTING_GOAL
   - BmBoxState.WAITING_RESULT

   For the first segment of the robot's trajectory (from starting pose to waypoint 0) the payload of the state message BmBoxState.TRANSMITTING_GOAL is the goal list (see Section 7.2.3). For all subsequent segments the payload is empty.

   When the Client state machine (also called Robot: see Section 6.2.3) receives the goal, the state machine transitions to ClientState.EXECUTING_GOAL and then to ClientState.WAITING_RESULT. Once the robot reaches the waypoint, the refbox performs the transition from ClientState.WAITING_RESULT to ClientState.READY. The payload of ClientState.WAITING_RESULT is empty.

   The pose of the marker set on the robot is acquired from the mocap system and the robot pose is computed using the (previously acquired) transform from marker set to odometric center. Finally, the accuracy of the robot pose wrt the waypoint pose is computed according to the metric described in the RoCKIn@Home Rulebook (RoCKIn Deliverable D2.1.3, Section 5.2.7); additionally, segment duration is determined.

3. End of benchmark. In this phase the BmBox goes through the states

   - BmBoxState.READY
   - BmBoxState.TRANSMITTING_SCORE
   - BmBoxState.END

   The BmBoxState.TRANSMITTING_SCORE's payload contains the score, formatted as described in Section 7.2.5.

record When the benchmark is launched the record node logs the following ROS topics:

```
/fbm2h/bmbox_state
/fbm2h/client_state
/fbm2h/refbox_state
/fbm2h/map
/fbm2h/map_metadata
/fbm2h/robot_at_home/pose
/fbm2h/robot_at_home/pose2d
/fbm2h/info
/tf
/rosout
```

Resulting logs are stored in *~/logs/*.

7.2.2 Configuration and data formats

Benchmark parameters consist in the goal list (that is sent by BmBox to RefBox) and the markerset-robot transforms (that are used by BmBox while computing robot pose). The goal list is in files textitrockin_scoring/config/fbm2h.yaml[25]; transforms are in directory *rockin_scoring/transforms/*. The data format used for the goal list is specified in Section 7.2.3.

7.2.3 Goal list data format

The goal list is sent to the refbox as a string in YAML format. The goal list corresponds to the configuration file rockin_scoring/config/fbm2h.yaml. The content of field num_waypoints is computed and added to the data before it is sent to the RefBox. The format of the goal list message is specified in Listing 10.

7.2.4 Transform data format

To run FBM2H, it is needed to acquire and use the transform between the pose of the marker set and the pose of the odometric center. This must be done for each of the robots (i.e., of the teams) executing the benchmark. When the benchmark is run, choice of the right transform is done by giving the correct value to the <team_name> argument when launching the benchmark (see Section 7.2.6). The transform is acquired as described in Section 7.2.7, and saved in a YAML file called transform-<team_name>.yaml. The format for such file is specified in Listing 11.

[25] See also and *rockin_scoring/config/fbm2h_test.yaml*.

```
goal:
  waypoints:       # list of Pose2D poses
  - [-2.12524914742, -1.60717535019, 1.55841375286]
  - [2.32184529305, -3.40717220306, 2.37683575141]
  - [1.73485660553, 3.14053320885, -0.0238841612192]
  - [-3.10801386833, 4.88111925125, -1.52614450226]
  - [4.30852985382, 3.19435501099, 2.20723667763]
  num_waypoints: 5 # computed by the benchmark
  starting_pose:
    [0.0, 0.0, 0.0]
  penalty_time:    # not used in the benchmark, needed by refbox
    120.0
  timeout_time:    # not used in the benchmark, needed by refbox
    120.0
```

Listing 10: example of goal list

```
robot_info: <team_name>     # team name or other team identifier
transform_timestamp: <%Y-%m-%d\_%H:%M:%S> # acquisition timestamp
marker_to_robot_transform:
- [<x>, <y>, <z>]           # translation from markerset to robot
- [<qx>, <qy>, <qz>, <qw>]  # orientation of robot wrt markerset
robot_to_marker_transform:
- [<x>, <y>, <z>]           # translation from robot to markerset
- [<qx>, <qy>, <qz>, <qw>]  # orientation of markerset wrt robot
```

Listing 11: Contents of *transform-<teamname>.yaml* file

During testing it may be useful to use an identity transform. Such transform is described in Listing 12.

```
robot_info: identity
transform_timestamp: 0
marker_to_robot_transform:
- [0.0, 0.0, 0.0]
- [0.0, 0.0, 0.0, 1.0]
robot_to_marker_transform:
- [0.0, 0.0, 0.0]
- [0.0, 0.0, 0.0, 1.0]
```

Listing 12: Contents of *transform-identity.yaml* file

7.2.5 Score data format

The score is sent to the refbox at the end of the benchmark as a string in YAML format. The format used for such string is described by Listing 13.

```
position_accuracy
orientation_accuracy
execution_time
hits                          # number of hits, input by human referee
timeout_segments: 0           # always 0
details:
  marker-robot transform      # transform used during the benchmark
segments_details:             # details for i-th segment of trajectory
  i:                          # i goes from 0 to (number of waypoints - 1)
    dinstance_error
    end_segment_time
    orientation_error
    refbox_timeout: false     # always false
    result_payload: {}        # always empty
    robot_pose:               # robot pose acquired when it reached waypoint i
    segment_time
    start_segment_time
    target_pose:              # pose of waypoint i
    timeout: false            # always false
```

Listing 13: Data format for FBM2@Home scores

7.2.6 Benchmark execution

Before executing the benchmark, it is necessary to execute the following preparatory operations:

1. for each robot, acquire the transform between marker set and odometry center;
2. in the Motive mocap software, define the rigid body corresponding to the marker set on the robot;
3. check the benchmark's configuration.

Note that the ID of the rigid body in Motive[26] has to match the value specified in *rockin_mocap/config/all_home_mocap.yaml*.

[26]The ID corresponds to the User Data field found in the Advanced part of the Rigid Body Properties form.

Executing FBM2@Home with the Referee Box This is the normal execution mode during an ERL competition. To run the benchmark, execute the command

```
$ roslaunch rockin_scoring fbm2h.launch team_name:=<team_name>
```

Note that <team_name>must be the same used in the name of the file containing the transform, i.e. the transform file must be named *transform-<team_name>.yaml*

A terminal opens, where the output of the *fbm2h* script is displayed. The benchmark is completely autonomous and doesn't require any user input during the execution. Only at the end of the benchmark the human referee is asked to input the number of hits (see the RoCKIn Rulebooks for details).

Testing with mocap system This mode of execution is used to test the benchmark. Both the refbox and the robot are simulated: the scripts *rockin_scoring/scripts/fbm2h_client_test* and *rockin_scoring/scripts/fbm2h_refbox_test* are used to manage and publish the states of the finite state machines associated to them (see Section 6.2 for details). To run the benchmark execute this command:

```
$ roslaunch rockin_scoring fbm2h_test.launch team_name:=identity
```

Three terminal windows open: one for *fbm2h_refbox_test*, one for *fbm2h_client_test* and the last for *fbm2h*.

In the Client terminal (the one associated to the robot) you have to confirm when to send the result to the refbox: this act corresponds to the robot reaching a waypoint.

Testing without mocap system This mode of execution is used to test the benchmark. The test consists in running the test described above without actually using the OptiTrack motion capture system. The output of the mocap system is simulated, which can be useful if such a system is not available or not set up yet. To do this, ROS nodes *mocap_optitrack*, *rosbag record*, *map_server* and tf static transform publisher are not launched: instead, a *bagfile* created with *rosbag* is played. The bagfile provides the ROS messages that the nodes (if they were running) would publish on their associated topics. As already described in Section 7.2.6, the scripts *rockin_scoring/scripts/fbm2h_client_test* and *rockin_scoring/scripts/fbm2h_refbox_test* simulate the robot and the refbox respectively.

To run the benchmark test execute these commands, in this order and each in a different terminal:

```
$ roscore
$ rosparam set use_sim_time true
$ rockin_mocap/test_logs$ rosbag play --clock <bagfile_name>.bag
$ roslaunch rockin_scoring fbm2h_test_optitrackless.launch team_name:=identity
```

Note that it's better to wait a second before running the last command, in order to leave time to rosbag and tf to publish some transforms.

A readymade bag is already available as file

rockin_mocap/test_logs/log_fbm2h_mocap_2015-10-23-16-15-40_0.bag

File

rockin_mocap/test_logs/log_fbm2h_mocap_2015-10-23-16-15-40_0_reached_waypoint_timing.yam

contains the *reached_waypoint* "timestamps" of the recorded robot. You have to send these manually to the refbox using the *fmb2h_client_test* terminal.

7.2.7 Acquisition of marker set-to-odometric centre transforms

To compare actual robot pose (i.e., pose of its odometry reference system) with waypoint pose, BMmanager needs to know both. For what concerns waypoint pose, it is part of the configuration of the benchmark; robot pose, instead, requires some processing. The mocap system providing the robot's Ground Truth pose does not, in fact, output the pose of the robot's odometry reference system: instead, it outputs the pose of the reference system associated to the marker set fitted to the robot. To obtain the first from the second, it is necessary to know the transform between them. To ease this activity, a software package (*rockin_acquire_markerset_transform*) has been prepared to acquire the *tf transform* (directly in ROS format) from the marker set to the odometry system of the robot, using the mocap system.

To acquire a transform:

1. set the robot at (x=0, y=0, θ=0) coordinates in the reference system of the testbed
2. run the command

   ```
   $ roslaunch rockin_acquire_markerset_transform acquire.launch
   ```

rockin_acquire_markerset_transform asks the user for a team name to associate to the transform. The acquired transform is saved as YAML file in */logs/transform-<team_name>.yaml*.

In rockin_acquire_markerset_transform/config/mocap.yaml there is the mocap cofiguration. It is not needed to configure this normally.

7.3 FBM3@Work: trajectory following

This functionality benchmark assesses the capability of a robot of precisely controlling the manipulator (and the mobile platform) motion while following a continuous path. The path is assigned to the robot by the Referee Box (which for RoCKIn@Work is also called *Central Factory Hub*). The path and the reference frame are also printed on a sheet of paper positioned on the table over which the end effector moves: this is for visualization by human observers, as the path of the end effector is tracked using the motion capture system.

The mocap system is used to measure deviations between the assigned path (*target_path*) and the actual path followed by the end effector (*robot_path*). In order to make the end effector trackable by the mocap system, a marker set is affixed to it by means that may depend on the specific robot executing the benchmark: for instance by having the end effector grip the marker set. The marker set used for FBM3@Work is shown in Figure 7.3[27].

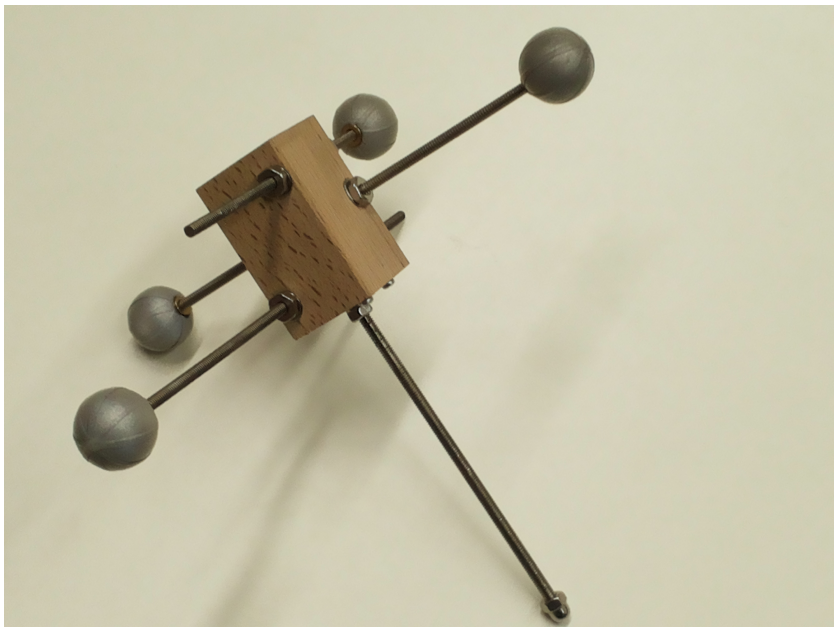


Figure 7: The marker set used for FBM3@Work.

The wooden element of the marker set of Figure 7.3 measures 30 mm x 30 mm x 50 mm. The protruding rod with the cap nut at its end extends for 100 mm out of the wooden element. In order to execute the benchmark, the

[27]This marker set is a modified version of the one used to track robots at the 2014 RoCKIn Competition.

robot has to follow the assigned path with the center point of the spherical element of the nut. During the benchmark, only the position of this point of the end-effector is considered, while the orientation is not: in practice, the benchmark requires the robot to move a (physical) point along a given trajectory. This leaves the robot free to adjust the position of the joints of its arm during the benchmark if geometrical constraints require it, while keeping the tracked point on the trajectory.

Currently no constraints are set on the velocity and/or acceleration of the trajectory tracking. Also, assigned trajectories are currently planar (2D).

The benchmark is composed by 5 runs. For each run a different path is selected as target path. Target paths are selected from a previously defined set. The overall score of the robot corresponds to its tracking accuracy in the best run.

7.3.1 Accuracy evaluation

Let the target path assigned to the robot be identified by $t(l)$, and the actual (Ground Truth) path followed by the robot be identified by $r(l)$, where l is a parameter in the range $[0 : 1]$. Considering that the assigned path is 2D and that only the (x, y) components of the trajectory of the end effector are used to compute robot accuracy,

- $r(l) = (x_{r(l)}, y_{r(l)})$ the parametric representation of the robot path
- $t(l) = (x_{t(l)}, y_{t(l)})$ the parametric representation of the target path

In particular

- $r(0)$ and $r(1)$ are the initial and final points of the robot path;
- $t(0)$ and $r(1)$ are the initial and final points of the target path.

Then robot accuracy is computed by

$$\frac{1}{N} * \sum_{l \in L_{sampled}} d(r(l), t(l)) \tag{1}$$

where $L_{sampled}$ is a subset of L_{gt} and L_{gt} are the values of l for which are available actual measurements of the position of the end effector (from the ground truth system), $N = |L_{sampled}|$ and $d()$ is the Euclidean distance. See Section 7.3.2 for additional information about the sampling of measurements that produces $L_{sampled}$.

7.3.2 Measuring the path of the end effector

The motion capture system provides a large number of measurements of the position of the end effector. As the number of available measurements is much larger than required to precisely capture the trajectory of the end effector, a subsampling process is useful. Subsampling does not only reduce data quantity: on the contrary, its most important function (if suitably performed) is to reduce the problems introduced by the unavoidable uncertainty in GT (Ground Truth) data. The following of this section describes the problems and explains how they are tackled for FBM3@Work.

The first problem concerns accuracy of measurement. Every measurement coming from the GT system is affected by a measurement error. Therefore, if robot path is simply defined as the composition of all measured positions, robot path has the form of a tangled and "fuzzy" yarn: very different from the (mathematically defined) target path. This makes comparison between the two paths to assess robot accuracy less straightforward than apparent. The problem remains if measurements are subsampled simply by decimation, i.e. by reading measurements periodically at a given frequency. In particular, robot path during time intervals when the end effector does not move take the form of "tangles" formed by tightly packed small movements, actually due to measurement errors; when the end effector moves, such "tangles" are deformed along the actual trajectory of the end effector. Only very rapid motion (i.e., motion where the end effector changes its position significantly during the sampling period) tends to transform these "tangles" into lines.

A consequence of the "motion jitter" introduced by measurement errors is is that the length of the robot path is very different from the length of the target path, and grows over time even when the end effector is stationary. If the distribution of the measurement error is constant over time and has a mean error of ϵ, robot path grows in length by ϵ for each sample. This means that path length is affected by a constant growth summed to the real one (due to the actual motion of the robot). If the mean error varies in time (which is likely as localization accuracy changes according to the visibility of the single markers of the marker set by mocap cameras), the length error does too. This introduces an additional problem, since, to compute the benchmark score, the robot path is "indexed" by its (normalised) length. In practice, lengthwise, the weight of any small portion of the path would depend on the measurement error distribution along that segment.

The solution to this problem is based on the fact that the benchmark focuses on the capability of the robot to correctly track a path (sequence of positions) with the end effector. Actual trajectory (position, speed, acceleration) of the end effector is not of interest: only its deviations from the

assigned path are significant. Measurement errors due to the motion capture system are much smaller than minimum significant deviation[28]: therefore they can be easily filtered by considering that the end effector "moved" only when the distance between its current and previous measured positions exceeds a predefined threshold d_min. By sampling measured positions in this way, measurement error does not affect anymore trajectory length, while of course it continues to affect the accuracy of each measurement. The value chosen for distance threshold d_min can be expressed as $c \cdot \epsilon$, where $c > 1$. The value of c should be experimentally chosen according to the specific benchmark setup (lighting, ...), and large enough that measurement error never exceeds d_min. As larger values for c decrease the number of elements of the set $L_{sampled}$ of sampled end effector positions, unnecessarily large values should be avoided as they make the computations described in Section 7.3.2 less precise.

A second problem that can affect the computation of robot accuracy is the occasional loss of tracking of the marker set by the mocap system. However, if the mocap system is correctly installed and calibrated and the benchmark location is suitably chosen (e.g., by positioning the robot in such a way that it does not overly mask visibility of the marker set by the mocap cameras) this problem can be confined to very small portions (if any) of the trajectory of the end effector. Then, the only consequence of the loss of tracking on the benchmark is that such short portions of the trajectories are described by a single straight segment instead of a succession of shorter straight segments: while computing robot accuracy this will make little difference on the result.

7.3.3 ROS nodes

When the benchmark is launched, these nodes are run: *fbm3w*, *world_map*, *map_server*, *mocap*, *record*. Some of these have already been described in Section 7.1.1; among the other, of special interest are the nodes described in the following.

fbm3w This is the ROS node in charge of running the benchmark. The source is stored in *rockin_scoring/scripts/fbm3w*. The script works as follows:

1. Startup phase. In the start up phase the BmBox goes through the following states:
   - BmBoxState.START
   - BmBoxState.WAITING_CLIENT

[28] This is, indeed, a requirement on the performance of the GT capture system.

- BmBoxState.READY

When the benchmark is launched, the robot is in front of the table where the benchmark will take place. The RoCKIn@Work Rulebook describes the procedure to align the robot's and BMmanager reference systems, and to correctly position on the table the sheet of paper where a facsimile of the assigned trajectory is printed. The configuration is loaded from *rockin_scoring/config/fbm3w.yaml*.

2. Execution on each run. The trajectory following task is executed by the robot multiple times. On each run, the BmBox goes through the following states:

   - BmBoxState.READY
   - BmBoxState.WAITING_MANUAL_OPERATION
   - BmBoxState.COMPLETED_MANUAL_OPERATION
   - BmBoxState.TRANSMITTING_GOAL
   - BmBoxState.EXECUTING_GOAL
   - BmBoxState.WAITING_RESULT

   The trajectory specification for the current run is read from the configuration (see 7.3.6). This only consist of a string indicating whether the target path is a line or a spline.

   The script requests a manual operation transitioning to BmBoxState.WAITING_MANUAL_
   The robot is told by the refbox to position the end-effector in reference position. Once the robot has finished positioning (and the reference point of the paper with the printed path has been manually positioned), the manual operation terminates and the state machine transitions to BmBoxState.COMPLETED_MANUAL_OPERATION.

   Then, the robot reference position is acquired.

   The script sends a goal request to the refbox transitioning to BmBoxState.TRANSMITTING_GOAL. The robot is told by the refbox to position the end-effector in the starting position and to start following the target path. Once the robot has reached the starting position (and the paper with the printed path has been manually oriented), goal request terminates and the state machine transitions to BmBoxState.EXECUTING_GOAL.

   The acquisition of the robot path (measured by the Ground Truth motion capture system) then starts. Acquisition exploits a callback

function collecting the poses as they arrive from the mocap system. The robot's starting position will be considered as the first pose in the robot path.

The script then calls WaitResult and transitions to BmBoxState.WAITING_RESULT. Once the robot has finished following the path, WaitResult returns and the state machine transitions to BmBoxState.READY or -if the current one is the last run- to BmBoxState.TRANSMITTING_SCORE.

Finally, acquisition of the robot path terminates, the robot's reference system is computed (from the reference position and starting position), and finally accuracy is computed by comparing target path and robot path.

3. End of benchmark. In this phase the BmBox goes through the states

   - BmBoxState.READY
   - BmBoxState.TRANSMITTING_SCORE
   - BmBoxState.END

   For state BmBoxState.TRANSMITTING_SCORE, payload of the state messagee includes the score[29], formatted as specified in Section 7.3.8.

record When the benchmark is launched the record node logs the following ROS topics:

```
/fbm3w/bmbox_state
/fbm3w/client_state
/fbm3w/refbox_state
/fbm3w/map
/fbm3w/map_metadata
/fbm3w/robot_at_work/pose
/fbm3w/robot_at_work/pose2d
/fbm3w/robot_path_in_frame
/fbm3w/target_path_in_frame
/fbm3w/info
/tf
/rosout
```

Resulting logs are stored in *~/logs/*.

[29]The Central Factory Hub (i.e., the Referee Box for RoCKIn@Work) actually discards payloads: the score is saved manually or written by hand. It is useful nonetheless to include the score in the payload, because this ensures that it get logged.

7.3.4 Configuration and data formats

The configuration for this banchmark consists in the list of target paths that the robot must follow over the set of runs composing the benchnmark, the actual shapes of the paths and the value of parameter D_MIN, defined in Section 7.3.2. Of these configuration elements, only the specifications for the paths are actually read from file (see Section 7.3.6; both the value of D_MIN parameter and the shapes are hardcoded in the script.

7.3.5 Setting position threshold

The value of D_MIN in the script can be changed if necessary, but this should only occur after the mocap system has been accurately calibrated. In this way the smallest value compatible with system performance can be chosen. Such value is one that filters gross inaccuracies due to momentary anomalies (if they exist) but does not completely smother "jitter" in the reconstructed marker set pose. The mean position error ϵ (see Section Section 7.3.2) that D_MIN is based upon can be measured using the package mocap_noise_statistics (see 7.3.13).

Note that the mean error ϵ is affected by the *smoothing* parameter of the Motive motion capture software. By increasing the smoothing, the measurement error decreases (thus D_MIN can be lowered), but ground truth measurements of the pose of the end effector become less consistent with real robot motion, up to the point where -with excessive smoothing- the actual robot path can get completely masked. For this reason, smoothing in Motive should be set as low as possible for this benchmark.

7.3.6 Path specifications

As already said, the only parameter that the benchmark loads from file is the list of path specifications for each run. A path specification only consists of a string indicating which type of path is selected. Available selections are defined as specified in Section 7.3.7. Path specifications are stored in YAML file *rockin_scoring/config/fbm3w.yaml* according to the format specified in Listing 14.

The number of elements in the list of path specifications is the number of runs for each execution of the benchmark. This number should be equal to the value assigned to the variable *BENCHMARK_RUNS.* in the script.

```
runs_specifications:
  - 'line'
  - 'spline'
  - 'line'
  - 'spline'
  - 'line'
```

Listing 14: fbm3w.yaml

7.3.7 Target path specifications

For FBM3@Work, the goal consists in the parametric description of a target path. Different paths shapes are hardcoded in the script in the form of mathematically defined curves. To each shape is associated a name; such names can be used in the path specification file (see Section 7.3.6) to select, for each run, one of the curves as the target path.

Note that the goal is not sent to the refbox, since the paths are predefined and known to the refbox as well.

Listing 15 provides an example of this method to define goals: four shapes are defined, namely "line_path", "semi_ellipse", "sine" and "spline_path":

```
def line_path(l):
  l = float(l)
  return Pose2D(0.15*l, 0.045*l, 0)
  # straight line from (0, 0) to (30, 10) [cm]

def semi_ellipse(l):
  l = float(l)
  return Pose2D(0.11 - 0.11*cos(pi*l), 0.11*sin(pi*l), 0)
  # semi ellipse (0,0),(5, 4.8),(10, 0) [cm]

def sine(l):
  l = float(l)
  return Pose2D(0.15*l, 0.075*sin(12.5*pi*0.15*l), 0)
  # semi ellipse (0,0),(5, 4.8),(10, 0) [cm]

spline_path = sine
```

Listing 15: example of functions defining curve shapes

7.3.8 Score data format

The format used for the score of FBM3@Work is a YAML string formatted as shown in Listing 16.

```
best_distance_error
best_run
execution_time
runs_details:
  i:                  # with i from 0 to number of runs - 1
    spec              # what path was selected as target_path for this run
    distance_error    # the distance error for this run
    execution_time    # the execution time for this run
```

Listing 16: Format used for the score of FBM3@Work

7.3.9 Benchmark execution

Before executing the benchmark, it is necessary to execute the following preparatory operations:

- in the Motive mocap software, define the rigid body corresponding to the marker set fitted to the end effector of the robot
- Measure the position's noise;
- check the benchmark's configuration.

Note that the ID of the rigid body in Motive has to match the value specified in *rockin_mocap/config/all_home_mocap.yaml*[30].

7.3.10 Executing FBM3@Work with the Referee Box

This is the normal execution mode during an ERL competition. To run the benchmark, execute the command

```
$ roslaunch rockin_scoring fbm3w.launch
```

A terminal opens, where the output of the *fbm3w* script is displayed. The benchmark is completely autonomous and doesn't require any user input during the execution.

It is useful to also open *rviz* (ROS' data visualizer) and observe the robot path and target path as they are published after the accuracy has been computed. To do this, before launching the benchmark execute the following commands in different terminals

```
$ roscore
$ rviz
```

[30]The ID corresponds to the User Data field found in the Advanced part of the Rigid Body Properties form.

When the benchmark starts, the topics to visualize using rviz are "/fbm3w/robot_path_in_frame" and "/fbm3w/target_path_in_frame".

7.3.11 Testing with mocap system

This mode of execution is used to test the benchmark. Both the refbox and the robot are simulated: the scripts *rockin_scoring/scripts/fbm3w_client_test* and *rockin_scoring/scripts/fbm3w_refbox_test* are used to manage and publish the states of the finite state machines associated to them (see Section 6.2 for details). To run the benchmark execute this command:

```
$ roslaunch rockin_scoring fbm3w\_test.launch
```

Three terminals open: one for *fbm3w_refbox_test*, one for *fbm3w_client_test* and the last for *fbm3w*. In the three terminals you have to manually confirm when to proceed.

7.3.12 Testing without mocap system

This mode of execution is used to test the benchmark. The test consists in running the test described above without actually using the OptiTrack motion capture system. The output of the mocap system is simulated, which can be useful if such a system is not available or not set up yet. To do this, ROS nodes mocap_optitrack, rosbag_record, map_server and tf static transform publisher are not launched: instead, a bagfile created with rosbag is played. The bagfile provides the ROS messages that the nodes (if they were running) would publish on their associated topics. As already described in Section 7.3.11, the scripts *rockin_scoring/scripts/fbm3w_client_test* and *rockin_scoring/scripts/fbm3w_refbox_test* simulate the robot and the refbox respectively.

To run the benchmark test execute these commands, in this order and each in a different terminal:

```
$ roscore
$ rosparam set use_sim_time true
$ rockin_mocap/test_logs$ rosbag play --clock log_fbm3w_mocap_2015-11-03-17-39-4
$ roslaunch rockin_scoring fbm3w_test_optitrackless.launch # it's better to wait
```

A readymade bag is already available as file *rockin_mocap/test_logs/log_fbm3w_mocap_2015-11-03-17-39-42_0.bag*.

7.3.13 Noise analysis

To determine the measurement error of the mocap system, set the Motive smoothing parameter to a reasonable value (guidelines to choose such value can be found in Section 7.3.5) and set the FBM3@Work marker set in a stable position, as still as possible. Then launch *mocap_noise_statistics* with command

```
$ roslaunch mocap_noise_statistics mocap_noise_statistics.launch
```

The terminal will show a graph with the distribution of the difference between two subsequent measurements of the position of the marker set as they are provided by the motion capture system. On the vertical axis is shown the distance in metres, grouped in quanta. On the horizontal axis is shown the normalised amount of samples that fell in that quantum.

When setting D_MIN, it is advisable to choose a value that stays far from the peak visible in the distribution.