The RoCKIn Benchmarking System

Martino Migliavacca,
Giulio Fontana,
Enrico Piazza

October 30, 2015

Contents

1 Introduction

This document is an operative description and user manual of the RoCKIn Benchmarking System. Additional information about the design, performance and limitations of the system can be found in Deliverable 2.1.8 of project RoCKIn(Description of Ground Truth System V2). The document is organized as follows.

Section 2 provides an overview of the system architecture, with details about the involved subsystems and the network configuration.

Section 3 is intended as a user manual: it explains how to set up and operate the system without entering into the details of the specific benchmarks.

Section 5 explains how the benchmarks that are managed by the RoCKIn Benchmarking Systemoperate, and describes the interaction between the system and the Referee Box.

Section 6 is a detailed description of the benchmarks. This description necessarily refers to a specific version of the benchmarks, which can differ from one RoCKInevent to another.

NOTE. This version of the document refers to the setup used for the 2015 RoCKInCompetition of Lisbon, Portugal.

1.1 Special terms

In the following, sometimes RoCKIn-specific terms, acronyms and abbreviations will be used. Here is a list of the most frequent.

- **BM** = benchmark
- **FBM** = Functionality Benchmark
- **TBM** = Task Benchmark
- **H**, **@H** = referred to the RoCKIn@Home Competition
- **W**, **@W** = referred to the RoCKIn@Work Competition
- **GT** = Ground Truth
- **mocap** = motion capture
- **marker set** = configuration of IR-reflective markers mounted on a robot to track its pose
- **ROS** = Robot Operating System[1]
- **OptiTrack** = the Natural Point OptiTrack motion capture system[2]
- **Motive** = Natural Point software package for the acquisition of motion capture data

[1] http://wiki.ros.org/
[2] http://www.optitrack.com/products/

2 Overview of the RoCKIn Benchmarking System

2.1 System Architecture

Figure 2.1 highlights the main elements of the RoCKIn Benchmarking Systemand shows that it is composed of several interconnected subsystems. Greyed elements represent external systems which interact with the RoCKIn Benchmarking Systembut are external to it.

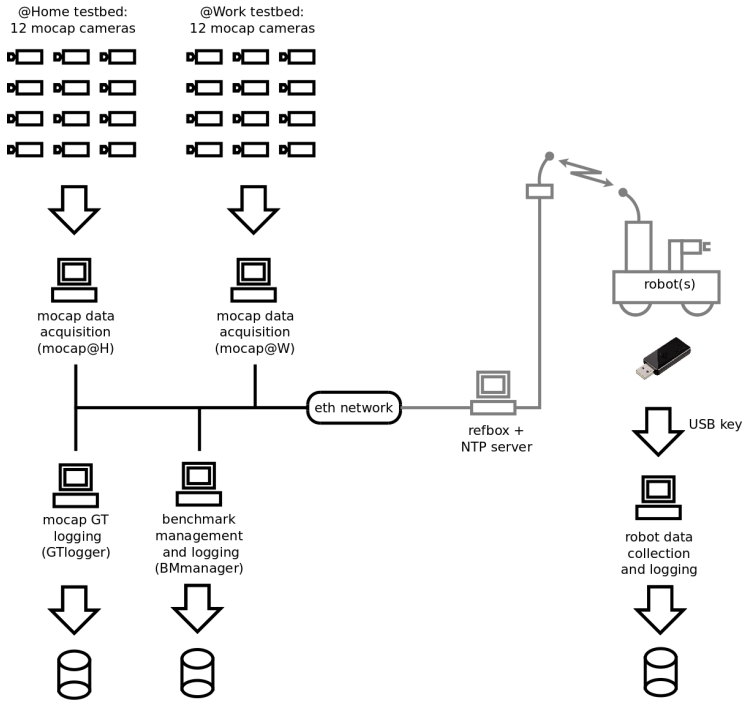


Figure 1: The RoCKIn Benchmarking System. Greyed elements are external systems that the RoCKIn Benchmarking Systeminteracts with.

Some of the machines shown in Figure 2.1 need to exchange data during the execution of the benchmark. They are:

1. two PCs acquiring motion capture (**mocap@H, mocap@W**) data from special cameras;
2. one PC (**GTlogger**) dedicated to processing mocap data to extract and log ground truth (GT) pose data;
3. one PC (**BMmanager**) dedicated to managing benchmarks and logging the data describing their execution.

These machines are interconnected via an ethernet wired network (*eth network*). One additional PC is used to collect and save robot-generated data, logged on USB keys (provided by RoCKIn) by the robot subjected to the benchmarks.

External systems interacting with the RoCKIn Benchmarking Systeminclude the Referee Box (which organizes competition activities, interfaces with devices belonging to the testbed and interacts with human referees) and the robot itself.

2.2 Motion Capture

RoCKInis a project focusing on robot benchmarking through competitions. To be able to benchmark actual robot performance, it is necessary to be able 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 [3]. 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 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 RoCKInmotion 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, RoCKInuses special *marker sets* composed of a laminated wood base fitted with 5 markers, as shown in Figure 2.2. 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 6.

In the setup for the RoCKInCompetition, two separate OptiTrack systems are used. Each of these is dedicated to one of the testbeds used by the RoCKIn-Competition. The RoCKIn@Home testbed is used for the RoCKIn@Home Task Benchmarks and Functionality Benchmarks; the RoCKIn@Work testbed is used for the RoCKIn@Work Task Benchmarks and Functionality Benchmarks.

Each motion capture system includes a dedicated data acquisition PC: a Windows PC running the OptiTrack Motive software [4], which is required to interface with the cameras and track the marker sets. The PCs used are called **mocap@H** (*Dell Quad-core*) for @Home and **mocap@W** (*Shuttle PC*) for @Work. Acquired mocap data is streamed over UDP to the clients, i.e. the Linux machines that process and log them.

Careful configuration of the Motive software is crucial for the performance of the RoCKIn Benchmarking System. A short guide to configure the system is provided by Section 3.1.1.

2.3 Networking

The ethernet network connecting the elements of the RoCKIn Benchmarking Systemis subdivided into two logical subnets: the benchmarking systems are connected to the 10.0.0.0/24 subnet, while the Referee Box and other appliances

[3] http://www.optitrack.com/products/
[4] http://www.optitrack.com/products/motive/

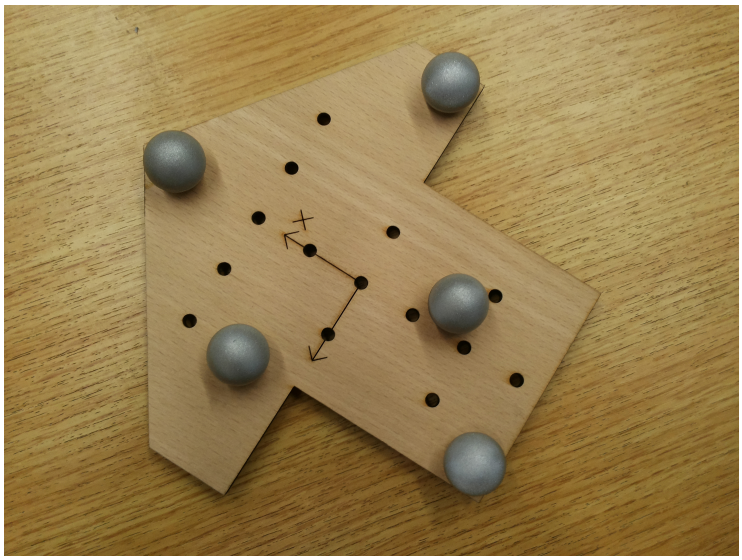


Figure 2: One of the *marker sets* mounted on robots so that they can be tracked by the RoCKIn Benchmarking System.

not related to benchmarking are on the 192.168.1.0/24 subnet. A router acts as a gateway between the 10.0.0.0/24 subnet (connected to one of the "LAN" ports) and the 192.168.1.0/24 network (connected to the "WAN" port).

For ground thruth (GT) mocap logging, a Linux PC running ROS is used, called **GTlogger** (*HP laptop*). A ROS node receives the UDP packets broadcasted by the Motive software and publishes, for each tracked marker set, the corresponding *tf* reference frame, the 3D pose, and the 2D pose. The GT is logged by a set of other ROS nodes, as described in Section 3.

To manage the benchmarks, a Linux PC running ROS is used, called **BMmanager** (*Zotac*). This PC receives data both from the Referee Box and from the motion capture systems (the benchmarks require, in fact, to compare them), and runs a set of ROS nodes to interact with the Referee Box, manage the execution of the benchmarks, and evaluate the final score, as described in Section 6.

The motion capture system is configured to use the network 10.0.0.0/24. All the PC have static IP addresses, and a router has been used to provide connection to the Internet. As the Functional Benchmarks need to interact with the Referee Box, the BMmanager PC is configured to use two IP addresses: 10.0.0.14 to communicate with the motion capture system, and 192.168.1.2 to communicate with the refbox. For this reason, the FBM management box, which needs to communicate with both the OptiTrack System and the Referee Box, has two IP addresses configured (by defining an alias for eth0). Table 2.3 summarizes the network configuration of the RoCKIn Benchmarking System.

TODO: add details about network setup: switches, routers, ...
TODO: why does the GTlogger have an alias?

PC	IP address	Netmask	Gateway
mocap@H	10.0.0.11	255.255.255.0	10.0.0.254
mocap@W	10.0.0.13	255.255.255.0	10.0.0.254
BMmanager	10.0.0.14	255.255.255.0	10.0.0.254
BMmanager (alias)	192.168.1.2	255.255.255.0	
GTlogger	10.0.0.15	255.255.255.0	10.0.0.254
GTlogger (alias)	192.168.1.3	255.255.255.0	
Refbox/NTP server	192.168.1.1		

Figure 3: Network configuration for the RoCKIn Benchmarking System.

2.4 Software packages

The GTlogger and BMmanager machines rely on custom software packages written for RoCKIn. These are:

- **rockin_mocap**, which manages acquisition of motion capture data;
- **rockin_logging**, which performs logging of motion capture data;
- **rockin_scoring**, which manages benchmark execution and interfaces with the Referee Box.

Both are packages written for the ROS middleware[5]. Their operation relies on other ROS packages not specific to RoCKIn: especially important within these is *mocap_optitrack*, which is used by rockin_mocap to receive motion capture data streamed by the Motive software and republish them as ROS *topics*.

All of the above packages (both the nonspecific ones and the ones written for RoCKIn) have to be present on the GTlogger and BMmanager machines. Their location within the filesystem is the directory ~/workspace/ros/src/.

In addition to the software packages, a set scripts have been prepared to ease the setup and operation of the RoCKIn Benchmarking System. These will be described in Section 4, and are located in directory ~/workspace/utils/.

2.5 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. The GT logging PC and the FBM PC run a NTP client (chrony [6]), configured to use the NTP server running on the RoCKIn@Home refbox (IP address 192.168.1.1). Listing 2.5 shows the configuration to use on NTP clients to sync with the server.

[5] http://wiki.ros.org/
[6] http://chrony.tuxfamily.org/

```
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 Ground Truth logging

As explained in Section 2, the RoCKIn Benchmarking Systemhas two different tasks:

- logging Ground Truth motion capture data;
- managing some of the RoCKInbenchmarks (which includes logging the associated data).

The first activity is performed independently from the second, on a dedicated PC (*GTlogger* in Figure 2.1). This section is dedicated to describe how the logging activities of GTlogger are performed. Please note that **all the filesystem paths in this section refer to the GTlogger machine.**

GTlogger includes three separate loggers, each dedicated to a specific category of ground truth information. Precisely, one is dedicated to the mocap data of the RoCKIn@Home Task Benchmarks, one is dedicated to the mocap data of the RoCKIn@Work Task Benchmarks, and the third is dedicated to the mocap data of all Functionality Benchmarks, both for RoCKIn@Home and RoCKIn@Work. The loggers can be started and stopped independently. There is not a one-to-one correspondence between loggers and motion capture systems, as the OptiTrack systems are two while the loggers are three.

At the 2015 RoCKInCompetition the area where the Functionality Benchmark take place is located within the RoCKIn@Home testbed, so the very same OptiTrack motion capture system is used for RoCKIn@Home Task Benchmarks, RoCKIn@Home Functionality Benchmarks and RoCKIn@Work Functionality Benchmarks (the second OptiTrack system is used for RoCKIn@Work Task Benchmarks). For this reason, the first and last of the loggers listed above use the very same data, i.e. those produced by the OptiTrack system of the RoCKIn@Home testbed. However, the logfiles produced by the two loggers differ, as the information that they extract from the data are not the same. For instance, the relevant rigid bodies are not the same for the two loggers. These differences are a consequence of the different configuration files of the two loggers.

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

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.

3.1 Configuration

To configure the mocap acquisition system and the loggers, 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, define the rigid bodies to be tracked, and assign them unique IDs[7]. 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 *~/workspace/ros/src/mocap.yaml* file to match the rigid body IDs to the required ROS frames and topics.
- For each logger, edit the relevant *~/workspace/ros/src/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.

3.1.1 Configuring the Motive software

Judicious "tweaking" of the operating parameters of Motive can greatly contribute to make rigid body pose acquisition more precise, stable and reliable. Here follows a list of suggested operations:

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

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

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.

3.1.2 Configuring logging for Task benchmarks, RoCKIn@Home (TBMH)

This logger acquires the pose (generated by the OptiTrack system) of a single rigid body: the robot marker set, which is common to all the robots participating to the RoCKInCompetition. For this rigid body, the corresponding 3D pose, 2D pose, and tf frame are acquired, published on ROS topics and logged by the nodes described in Section 3.2. Configuration of these operations is specified in the *~/workspace/ros/src/tbmh_mocap.yaml* file. Note that the numeric ID assigned, in Motive, to the rigid body corresponding to the marker set must match the one used in this file.

```
rigid_bodies:
  '3':
    pose: robot/pose
    pose2d: robot/pose2d
    child_frame_id: robot_at_home
    parent_frame_id: world
```

Listing 2: tbmh_mocap.yaml

The map image file is *~/workspace/ros/src/tbmh_map.pgm* and its configuration is *~/workspace/ros/src/tbmh_map.yaml*, both stored in the *~/workspace/ros/src/config* folder.

```
# Map image file
image: tbmh_map.pgm

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

# The 2-D pose of the lower-left pixel in the map
origin: [-5.02, -4.02, 0]

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

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

Listing 3: tbmh_map.yaml

3.1.3 Configuring logging for Task benchmarks, RoCKIn@Work (TBMW)

This logger acquires the pose (generated by the OptiTrack system) of a single rigid body: the robot marker set, which is common to all the robots participating to the RoCKInCompetition. For this rigid body, the corresponding 3D pose, 2D pose, and tf frame are acquired, published on ROS topics and logged by the nodes described in Section 3.2. Configuration of these operations is specified in the *~/workspace/ros/src/tbmw_mocap.yaml* file. Note that the numeric ID assigned, in Motive, to the rigid body corresponding to the marker set must match the one used in this file.

```
rigid_bodies:
  '5':
    pose: robot/pose
    pose2d: robot/pose2d
    frame_id: robot_at_work
```

Listing 4: tbmw_mocap.yaml

The map image file is *~/workspace/ros/src/tbmw_map.pgm* and its configuration is *~/workspace/ros/src/tbmw_map.yaml*, both stored in the *~/workspace/ros/src/config* folder.

```
# Map image file
image: tbmw_map.pgm

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

# The 2-D pose of the lower-left pixel in the map
origin: [-3.48, -1.34, 0]

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

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

Listing 5: tbmw_map.yaml

3.1.4 Configuring logging for Functionality benchmarks, RoCKIn@Home and RoCKIn@Work (FBM)

This logger acquires the pose (generated by the OptiTrack system) of multiple single rigid bodies: the robot marker set, which is common to all the robots participating to the RoCKInCompetition, and the rigid bodies used for the execution of the Functional Benchmarks. For this rigid bodies, the corresponding 3D pose, 2D pose, and tf frame are acquired, published on ROS topics and logged by the nodes described in Section 3.2. Configuration of these operations is specified in the *~/workspace/ros/src/fbm_mocap.yaml* file. Note that the numeric IDs assigned, in Motive, to the rigid bodies match the ones used in this file.

The FBM1 configuration is intended to log the Functional Benchmark 1 (Object perception) for both RoCKIn@Home and RoCKIn@Work. It acquires two rigid bodies: the table origin marker set (Motive ID: 1) and the reference board marker set (Motive ID: 2). For each rigid body, the corresponding 3D pose, 2D pose, and tf frame are published to ROS, as specified in the *fbm1_mocap.yaml* configuration file.

```
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
    '4':
      pose: robot/pose
      pose2d: robot/pose2d
      child_frame_id: robot_at_home
      parent_frame_id: world
```

Listing 6: fbm1_mocap.yaml

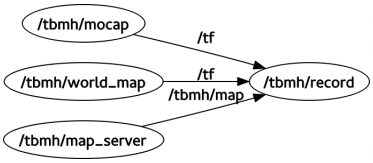
The map image file is *~/workspace/ros/src/fbm_map.pgm* and its configuration is *~/workspace/ros/src/fbm_map.yaml*, both stored in the *~/workspace/ros/src/config* folder.

```
# Map image file
image: fbm_map.pgm

# Resolution of the map, meters / pixel
resolution: 0.001

# The 2-D pose of the lower-left pixel in the map
origin: [-0.1, -0.1, 0]

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

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

Listing 7: fbm_map.yaml

3.2 ROS nodes used for 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 4.1).



3.2.1 map_server

This node is a ROS *map_server*[8] 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.

3.2.2 world_map

This node is a ROS *static_transform_publisher*[9] 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.

3.2.3 mocap

This node is a ROS *mocap_optitrack*[10] 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[11] software, decodes them and broadcasts the poses of configured rigid bodies as tf transforms, poses, and/or

[8] http://wiki.ros.org/map_server#map_server-1
[9] http://wiki.ros.org/tf#static_transform_publisher
[10] http://wiki.ros.org/mocap_optitrack
[11] http://www.optitrack.com/products/motive/

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.

3.2.4 record

This node is a a ROS *record* node, a C++ node provided by the ROS *rosbag*[12] 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.

3.3 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 4.1. Such section also explains how specific RoCKInscripts are used to create such environments and to switch among them. The following instructions describe how to run each of the loggers and its associated monitor (a script that checks that the logfiles increases in size at the expected rate: see 4.2). Usually, more than one couple logger+monitor 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

```
$ tbmh_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 "TBM@HOME".

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

```
TBM@HOME $ roslaunch rockin_mocap tbmh_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, the n run the *log_monitor* script:

```
$ tbmh_env
TBM@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.

[12] http://wiki.ros.org/rosbag

4 Tools

4.1 Environments

TODO: update with the Lisbon environment set

To perform correctly and adapt their operation to specific circumstances, many components of the RoCKIn Benchmarking Systemrequire that a suitable *environment* is set up beforehand. Such environments use environment variables to define working parameters for the RoCKInsystem, such as the port used by the *roscore* component of ROS, relevant IP addresses, and so on. In order to ease the setup of such environments and to allow a quick switch from one to the other, a few bash scripts are provided in the *~/workspace/utils/* directory. Beyond defining the required environment variables, each of these scripts adds a specific prefix to the *bash* shell prompt, so the user can immediately recognize the active environment.

Using environments, multiple benchmarking systems can run on the same machine at the same time. Each of them should be run in a separate terminal window, after having set up the required environment using the scripts. The scripts are available through a series of bash commands that become available by sourcing file *~/workspace/ros/src/utils/setup.bash*. This can be done by running command

source ~/workspace/ros/src/utils/setup.bash

or by adding the setup file to the *.bashrc* file. The commands made available by *~/workspace/ros/src/utils/setup.bash* are the following:

- `tbmh_env` adds the `TBM@HOME` prefix to bash and exports the following environmental variables:
  `\ro_ENV="TBMH"`
  `ROS_MASTER_URI="http://localhost:11312"`
- `tbmw_env` adds the `TBM@WORK` prefix to bash and exports the following environmental variables:
  `\ro_ENV="TBMW"`
  `ROS_MASTER_URI="http://localhost:11313"`

4.2 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/ros/src/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 4.1). 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 TBM@HOME, the log_monitor script includes the following lines:

```
if os.environ['ROCKIN_ENV'] == 'TBMH':
    print colors.INFO + "TBM@HOME environment" + colors.END
    env_prefix = "TBM@HOME "
    log_prefix = LOG_DIR + "log_tbmh_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 (using one of the the helpers from section 4.1) and, finally, command *log_monitor* must be run:

```
$ tbmh_env
TBM@HOME $ log_monitor
```

5 Benchmark structure

RoCKInbenchmarks are subdivided into two categories:

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

For obvious reasons, Section 5 and Section 6 are only concerned with the first category. This section, in particular, describes the general structure of a RoCKInbenchmarks. At the moment when this document has been written, benchmarks relying on the RoCKIn Benchmarking Systemonly included Functionality Benchmarks: for this reason, in the following the two categories will sometimes be treated as equivalent. There is no reason, of course, why Task Benchmarks may not rely on the RoCKIn Benchmarking Systemas well.

For the 2015 RoCKInCompetition, Some of the RoCKInFunctionality Benchmarks depends on the motion capture system to be executed and to compute scores. In particular, the Benchmarking System is involved in the execution of the following FBMs:

- FBM1@Home (Object Perception)
- FBM1@Work (Object Perception)
- FBM2@Home (Navigation)
- FBM2@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).

5.1 Actors

Specific information about how each benchmark is designed and how it is executed will be provided by Section 6. 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 6: 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 PC, also called *Benchmarking Box*;
2. the Referee Box PC, also called *refbox*;
3. the robot.

Each of them has a behaviour that is governed by a finite state machine. Changes of state are triggered by events. The next part of Section 5 provides a description of these finite state machines, and details about the communication means connecting them. In the following, the term **benchmark state** will be used to identify the joint states of the above three interacting finite state machines.

5.2 Communication

To successfully cooperate to perform the benchmark, the actors described in Section 5.1 need to communicate. As shown by Figure 2.1, these communications pass through suitable networks. This section provide more information about the way they occur.

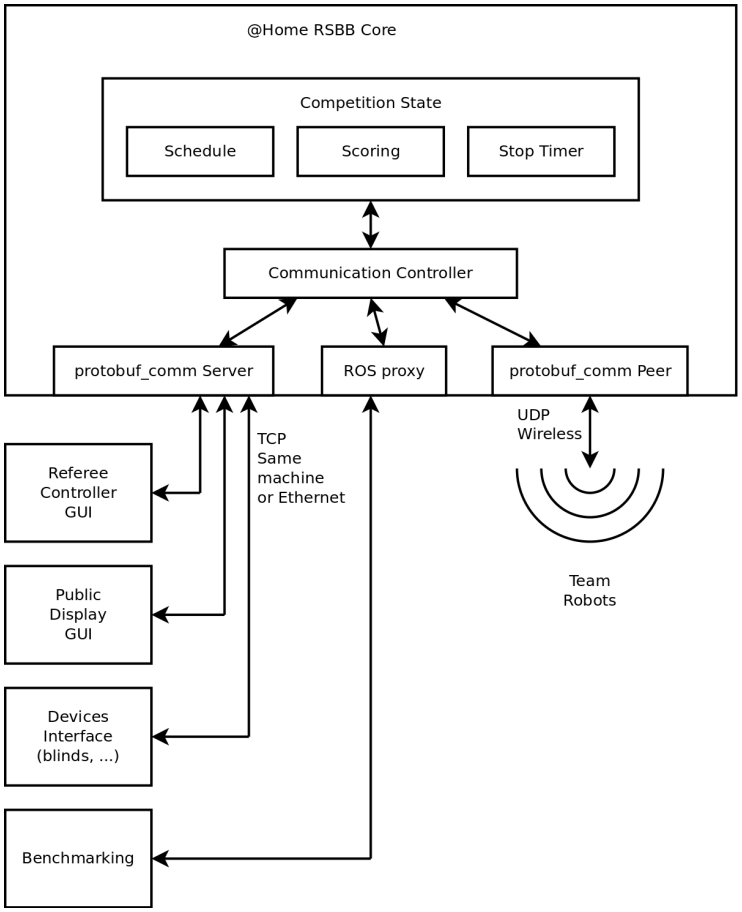


Figure 5.2 shows the Referee Box, and especially its communication channels with external systems. The whole system shown by figure 5.2 is sometimes called **RoCKInScoring and Benchmarking Box**, or **RSBB**. The "Benchmarking" element of the RSBB is the BMmanager PC; the "RSBB Core" element is the Referee Box.

The BMmanager PC (Benchmarking Box) and the @Home Referee Box (RSBB Core) communicate through ROS middleware, by publishing and receiv-

ing messages on ROS Topics. To do that, they share a single *roscore*, running on the Referee Box. As the RSBB relies on protobuf to communicate with the robots, the RSBB developers asked to avoid RPC-style interactions and use only the publish/subscribe messaging paradigm. A ROS Proxy module inside the RSBB Core takes care of converting data from ROS to protobuf and vice versa. As said, a single instance of *roscore* is executed on the computer running the RSBB Core: all other ROS systems of the RSBB connect to this instance of *roscore*.

The Benchmark State describes the current aggregated state of Referee Box, BMmanager and robot. It 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).

5.3 Benchmark States

The overall state of a benchmark is descriped by 3 different states:

- the Benchmarking Box state;
- the Referee Box state;
- the Robot state;

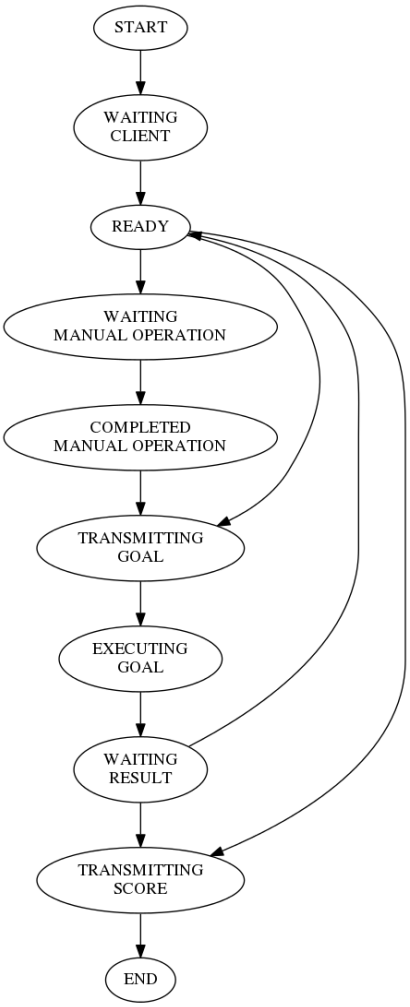
State transitions for a finite state machine are driven by state updates of the other state machines.

5.3.1 Benchmarking Box States

The Benchmarking Box runs a finite state machine with 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 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

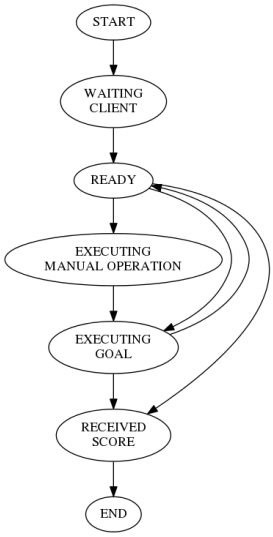
The current state is published on the *fbm_name/bmbox_state* topic, while message content is described by *BmBox.msg* (see Listing 8).



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 8: BmBoxState.msg



5.3.2 Referee Box States

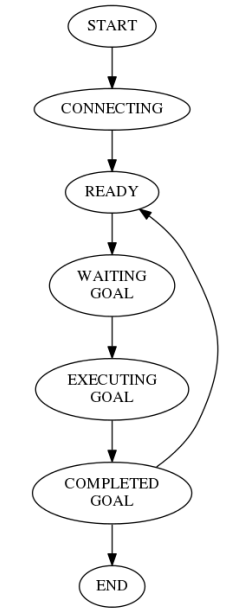
The Referee Box runs a finite states machine with the following states:

- ***WAITING CLIENT***: waiting for a connection from the Robot
- ***READY***: everything ready to start the benchmark
- ***EXECUTING MANUAL OPERATION***: the manual operation is beeing executed
- ***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*.

```
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 9: RefBoxState.msg



RoCKIn@Home/Work - Robot states

5.3.3 Robot States

The Robot runs a finite states machine with 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

The current state is published on the *fbm_name/client_state* topic, while message content is described by *ClientState.msg*.

```
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 10: ClientState.msg

5.4 Generic Benchmark Workflow

What follows is a description of the way a generic benchmark is executed, in terms of states of the three actors (see Section 5.1).

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., the referee must put an object in front of the robot), the Benchmarking Box updates its state to *WAITING MANUAL OPERATION*, sending 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, going to state *SENDING GOAL* and transmitting the description of the goal as payload of the state message. When the Robot 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 completed 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 more goals, waits for the robot requesting a new goal (i.e., the state is updated to *READY*), otherwise, the state is updated to *TRANSMITTING SCORE*, with the final score as payload, and the benchmark is concluded.

At any time, the state can be updated to *END* by the Referee Box, with payload "timeout" if it runs out of time, or payload "halt" if the benchmark was halted by a human request. The RSBB Core runs a stop timer, and provides a way to halt the benchmark by a human.

6 Benchmarks

As explained in Section 2, the RoCKIn Benchmarking Systemhas two different tasks:

- logging Ground Truth motion capture data;
- managing some of the RoCKInbenchmarks (which includes logging the associated data).

The second activity is performed independently from the first, on a dedicated PC (*BMmanager* in Figure 2.1). This section is dedicated to describe how the benchmarking activities of BMmanager are performed. Please note that **all the filesystem paths in this section refer to the BMmanager machine.**

TODO: REVIEW THE REST OF THE DOCUMENT

6.1 Functional Benchmark 1 for RoCKIn@Home and RoCKIn@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:

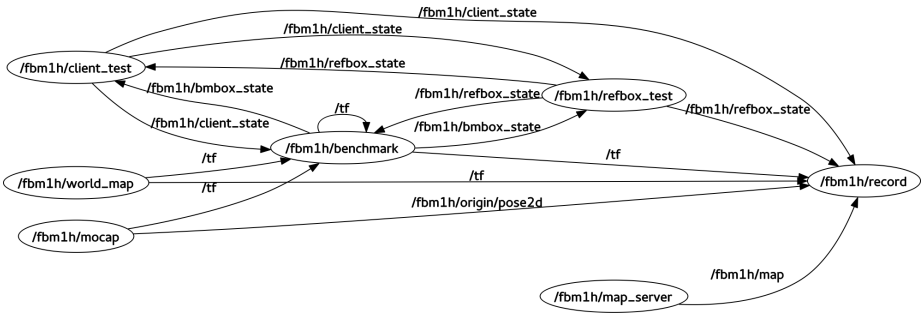
- object class name [string]
- object instance name [string]
- object pose (x [m], y [m], theta [rad])

The motion capture system is used to acquire the actual pose of the object. For this purpose, two marker sets are defined:

- one fixed to the table, which specifies the origin
- one fixed to a reference board, which has a slot to accomodate the objects

In this way, given the transformations between each object and the reference board marker set, we can acquire the pose of the object with respect to the origin.

6.1.1 ROS nodes



6.1.2 map_server

ROS *map_server* node, as described in Section 3.2.1

6.1.3 world_map

ROS *static_transform_publisher* node, as described in Section 3.2.2

6.1.4 mocap

ROS *mocap_optitrack* node, as described in Section 3.2.3

6.1.5 record

ROS *record* node, as described in Section 3.2.4

6.1.6 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 actual position of the object reference frame
- computes the score

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.

6.1.7 refbox_test

This node simulates the Referee Box. 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.

6.1.8 client_test

This node simulates the Robot. 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.

6.1.9 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 6.1.11.

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 meters)
- 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 11: fbm1h.yaml

6.1.10 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 set have to be defined in the motive OptiTrack system:

- the *origin* marker set, composed of the 4 markers fixed to the table (Motive ID 1)
- the *ref_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 codes attached to the table, proceed as follows:

1. align the Optirack ground plane calibration tool with the AR codes (a replica of the original tool with correct offsets has been made, and should be taped to the back of the table)

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)

6.1.11 Acquisition of new objects

To acquire the object pose, the transformation from the reference board marker set to the frame of the particular object has to be known. Indeed, the objects may have different frame origins and orientations, depending on their shape. For this reason, when adding new object to the list, they must be carefully acquired with the motion capture system to find the exact transformation.

The reccomended 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 [13]), 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 RoCKIn2014 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
```

6.1.12 Execution

6.1.13 Using the Referee Box

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*:

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

The benchmark execution is managed by the refbox: whenever a client 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.

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

Figure 4: rockin FBM1 table and reference frame

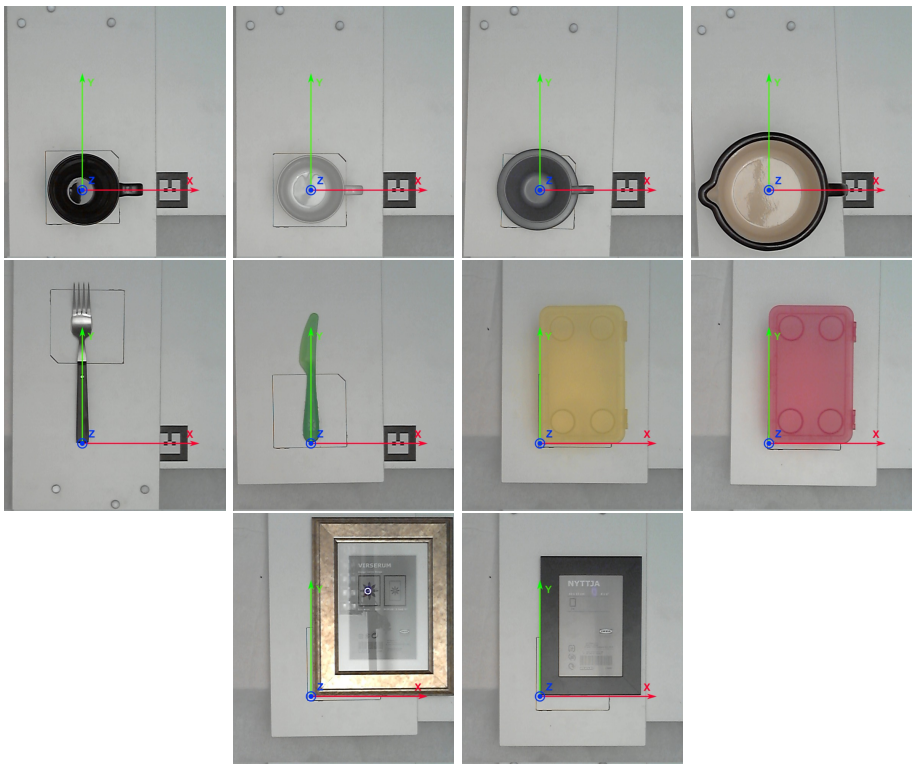


Figure 5: RoCKIn@Home FBM1 objects

6.1.14 Manual execution

To start the FBM1 using 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.

6.1.15 Testing

A set of python scripts simulating the referee box and the robot can be used to test the FBM1 workflow. To run a test, firt 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
```

6.1.16 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.

6.2 Functional Benchmark 2 for RoCKIn@Home: navigation

TODO

6.3 Functional Benchmark 2 for RoCKIn@Work: trajectory following

TODO