Master in Artificial Intelligence and Robotics (MARR)

Robot Programming

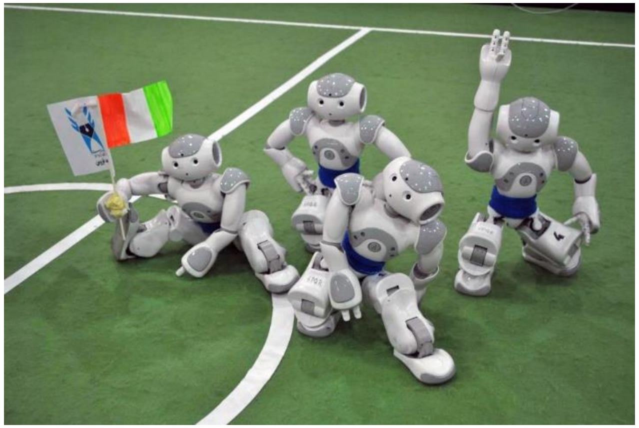
2015/2016

SAPIENZA
UNIVERSITÀ DI ROMA

Programming Nao-Robots

Francesco Riccio
riccio@diag.uniroma1.it

SPL – Standard Platform League



S.P.Q.R.

(Soccer Player Quadruped Robots) is the RoboCup team of the Department of Computer, Control, and Management Engineering "Antonio Ruberti" at Sapienza university of Rome

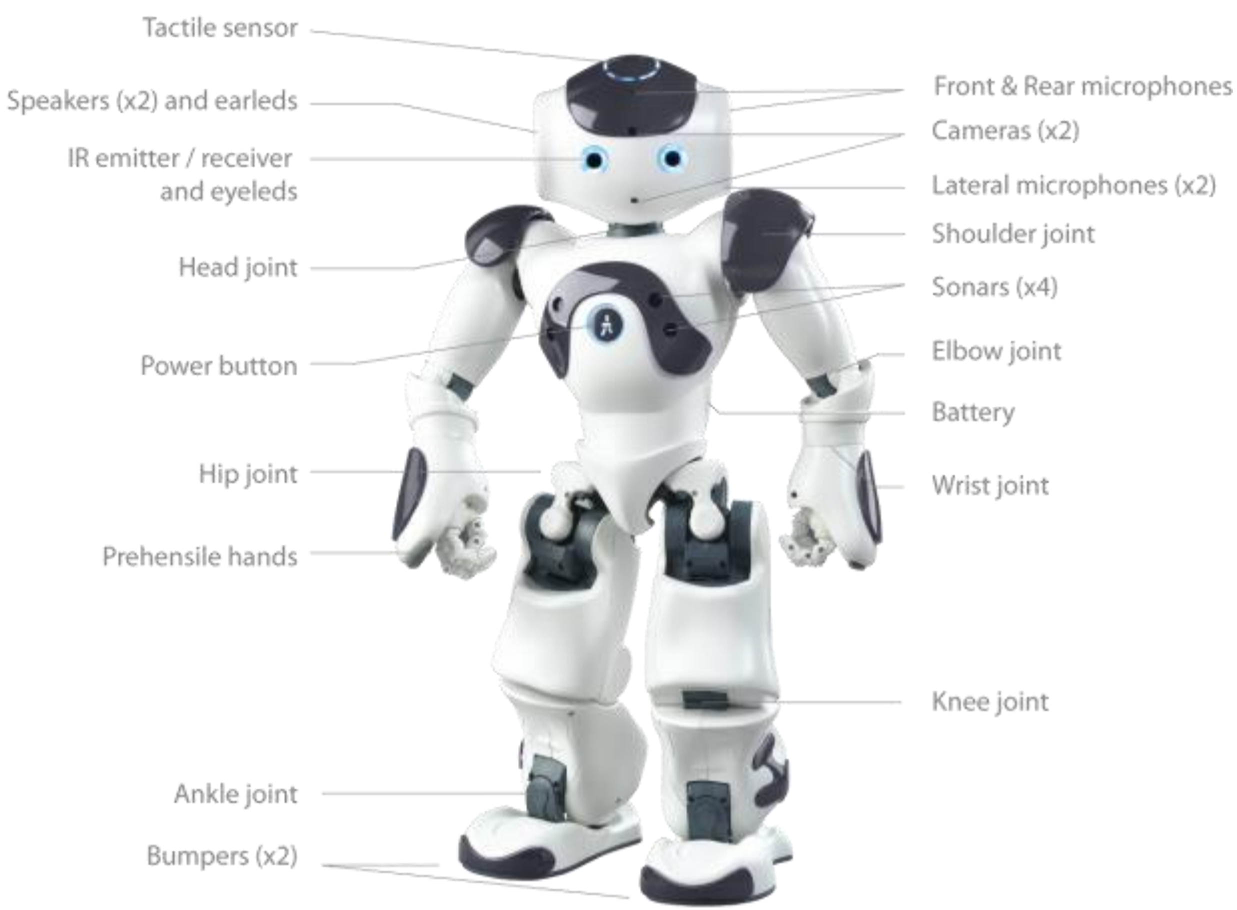
- Middle-size 1998-2002;
- Four-legged 2000-2007;
- Real-Rescue robots since 2003;
- Virtual-Rescue robots since 2006;
- Standard Platform League since 2008;

http://spqr.diag.uniroma1.it

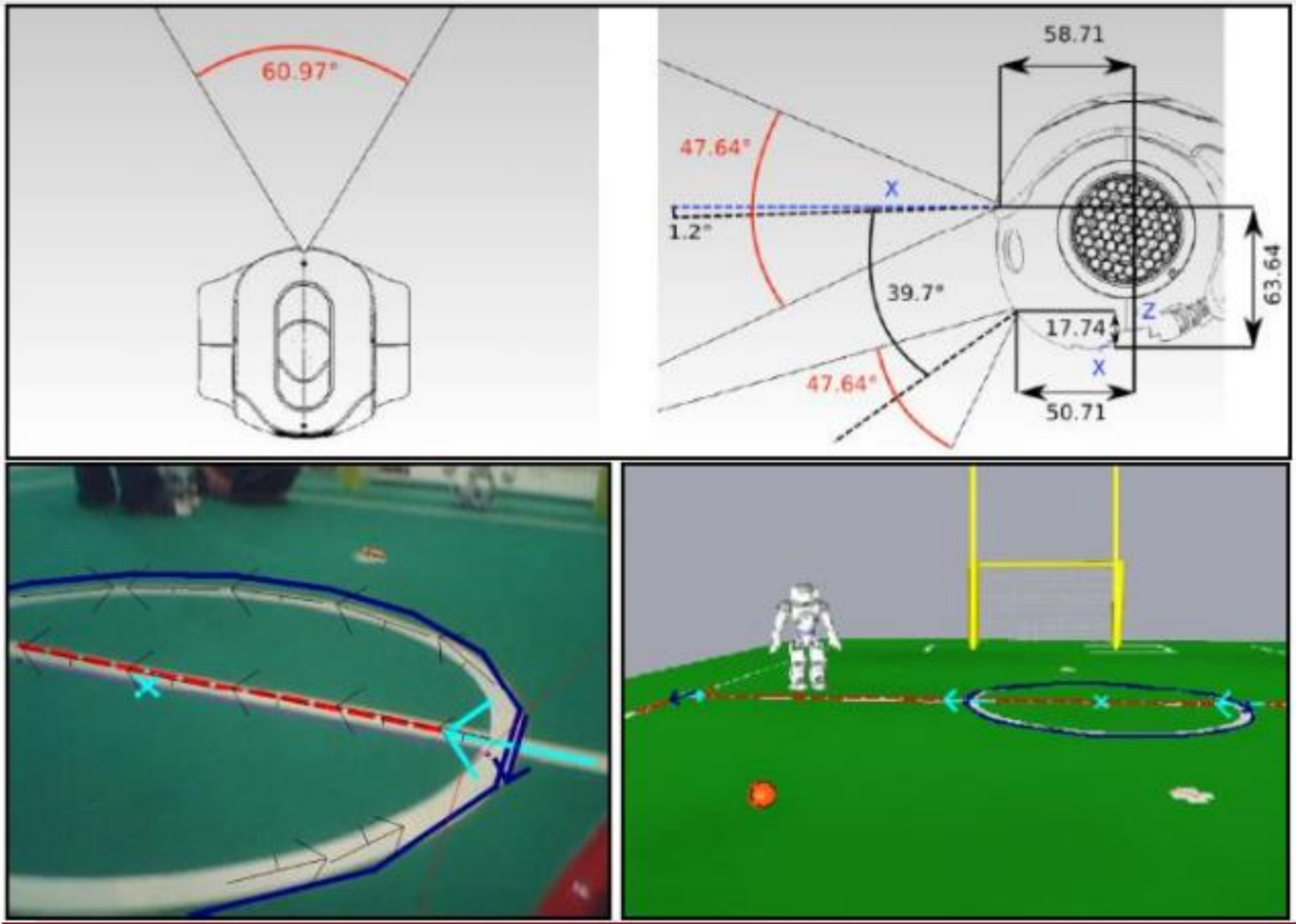
The Aldebaran Nao robot

Nao is an autonomous, programmable, medium-sized humanoid robot.

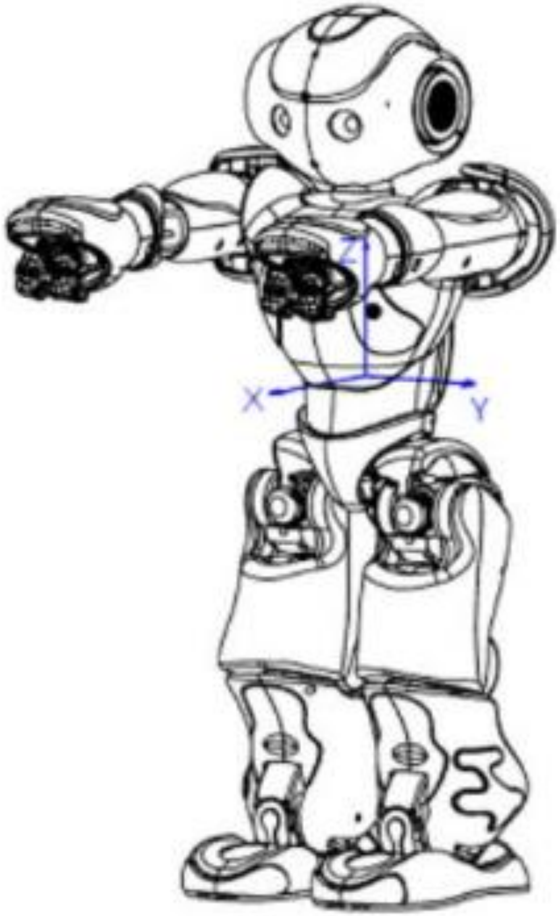
ATOM Z530 1.6GHz CPU 1 GB RAM / 2 GB flash memory / 4 to 8 GB flash memory dedicated



Camera Nao



Inertial Unit

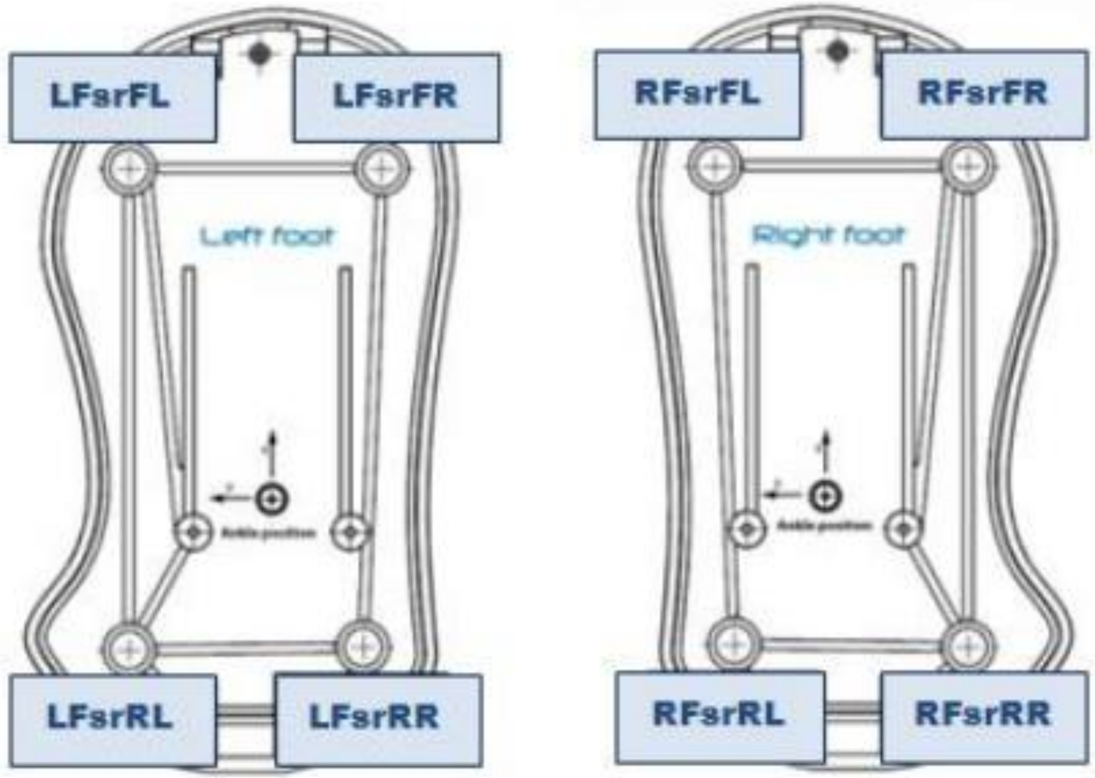


- 2 axes gyrometers
- 1 axis accelerometers

The Inertial unit is located in the torso

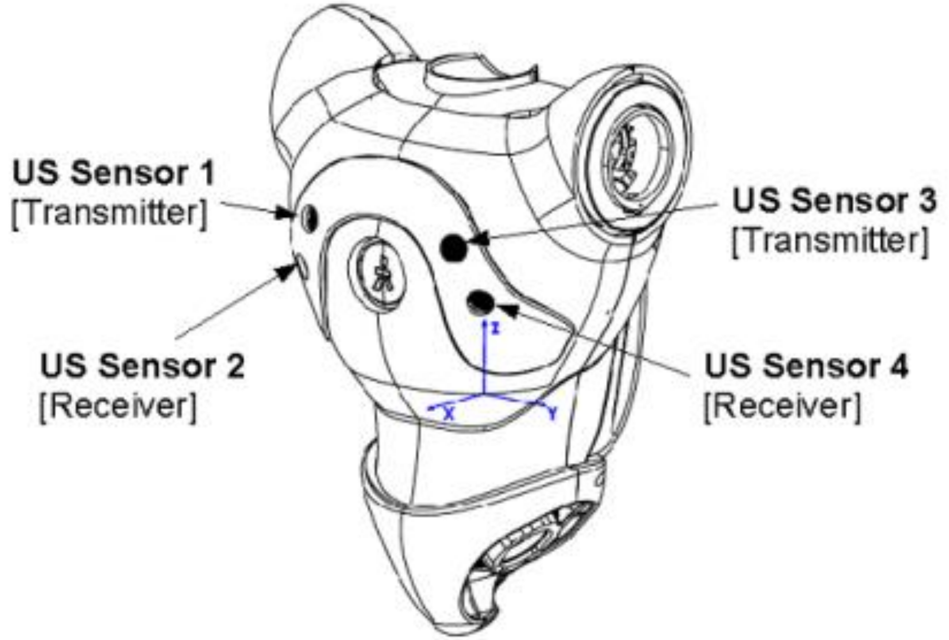
FSR – Force Sensitive Resistor

These sensors measure a resistance change according to the pressure applied.

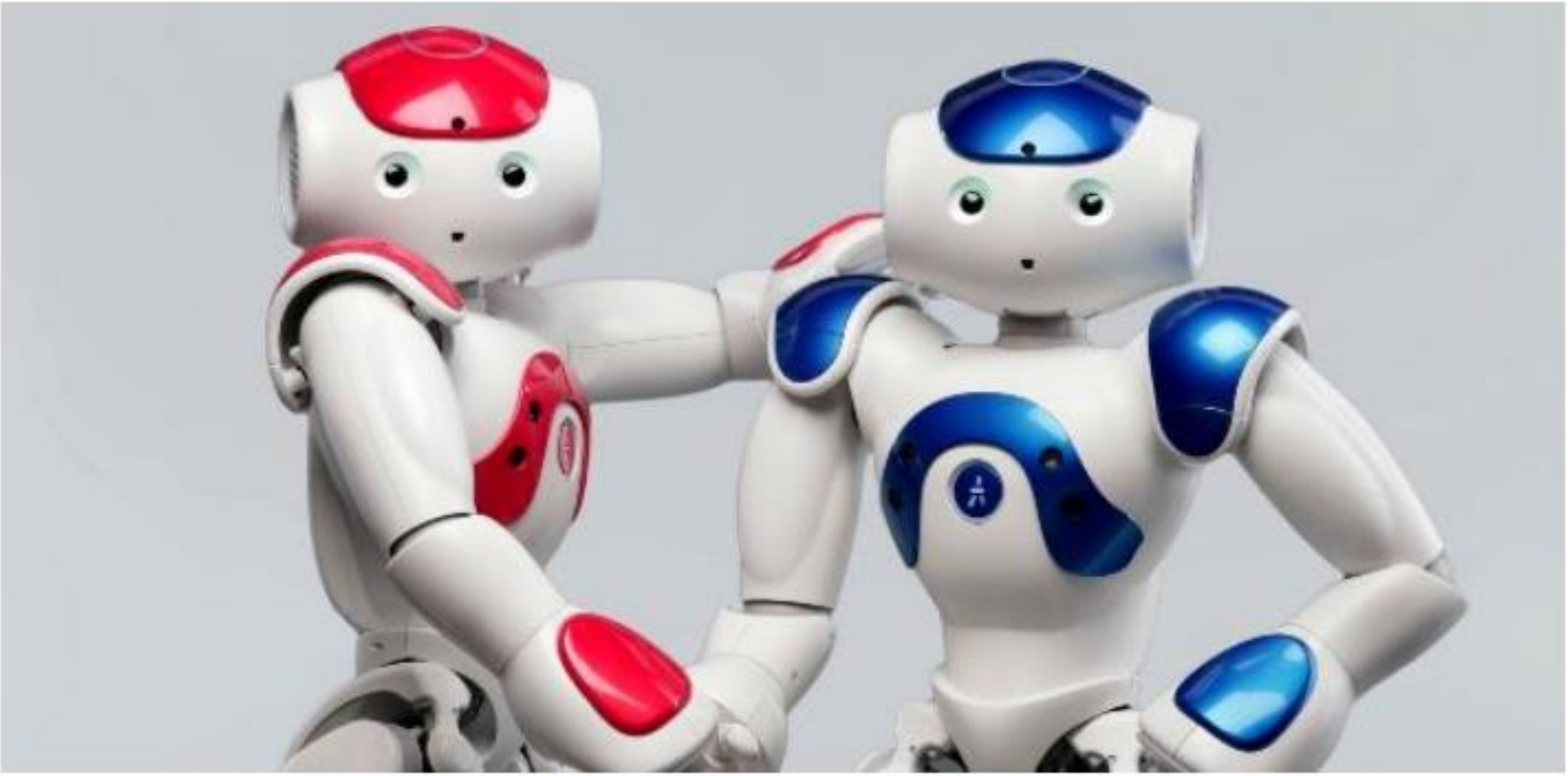


Sonars

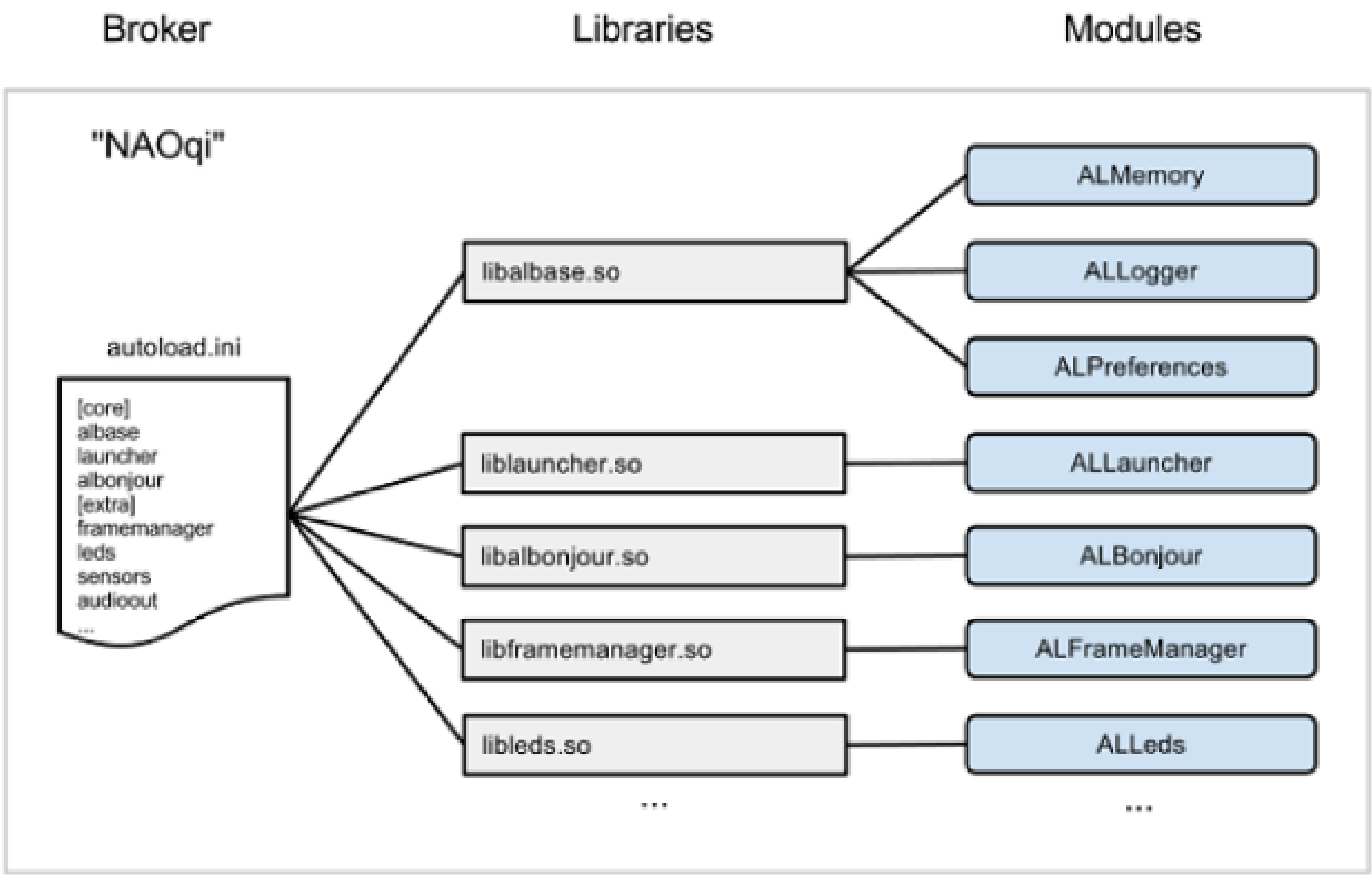
- Resolution: 1cm
- Frequency: 40kHz
- Detection range: 0.25m -2.55m
- Effective cone: 60°



Nao Robot Software Support

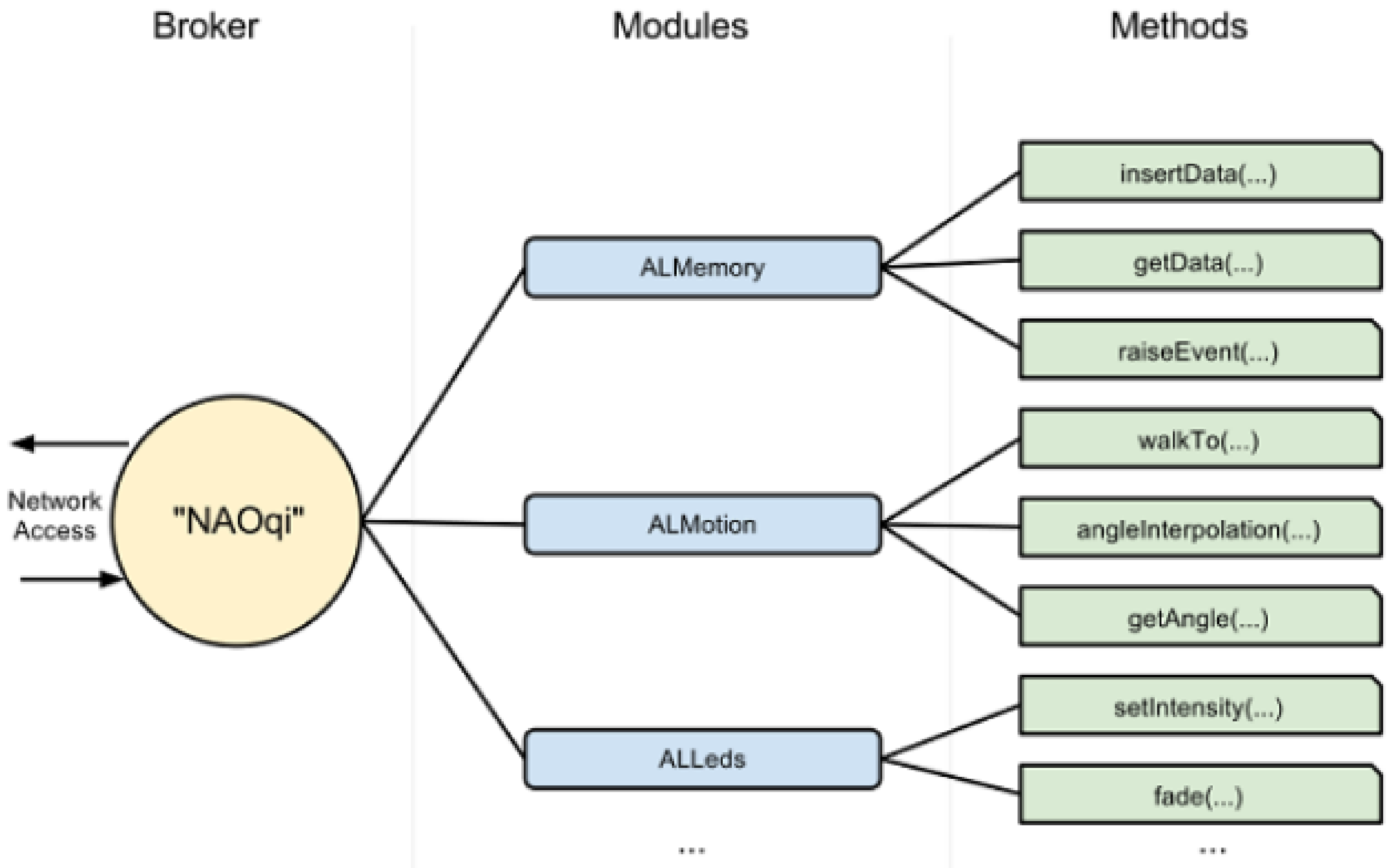


Naoqi API



https://community.aldebaran-robotics.com/doc/

Naoqi API



https://community.aldebaran-robotics.com/doc/

Naoqi API

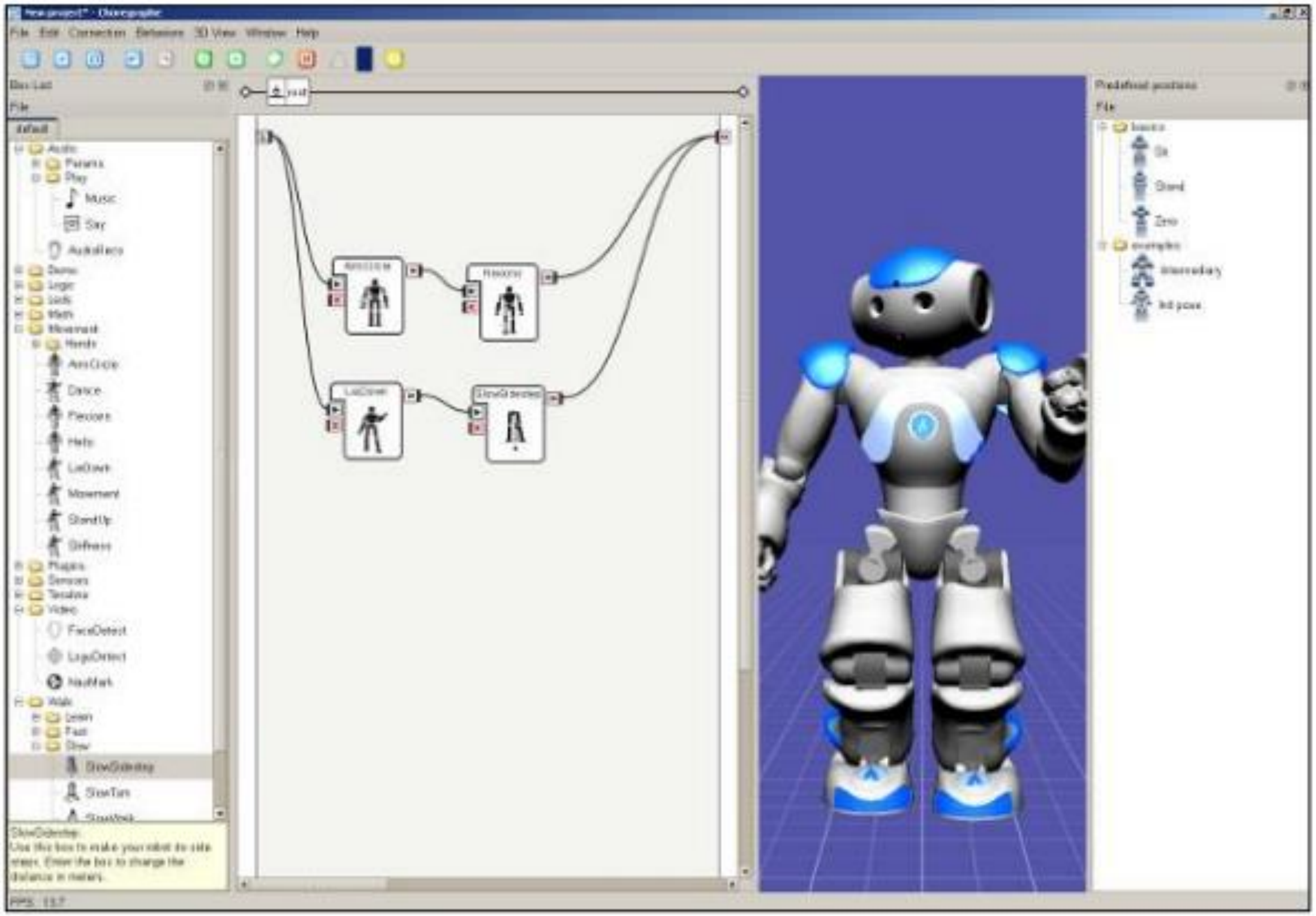
A **broker**:
provides directory services allowing you to find modules and methods.
provides network access allowing the methods within modules to be called from outside the process.

A **proxy** is an object that will behave as the module it represents.
For instance, if you create a proxy to the ALMotion module, you will get an object containing all the ALMotion methods.

A **Module** is a class within a library. When the library is loaded from the autoload.ini, it automatically instantiates the module class.

Choregraphe

Naoqi API



B-Human Framework Architecture

Based on the original framework of the GermanTeam, developed by:

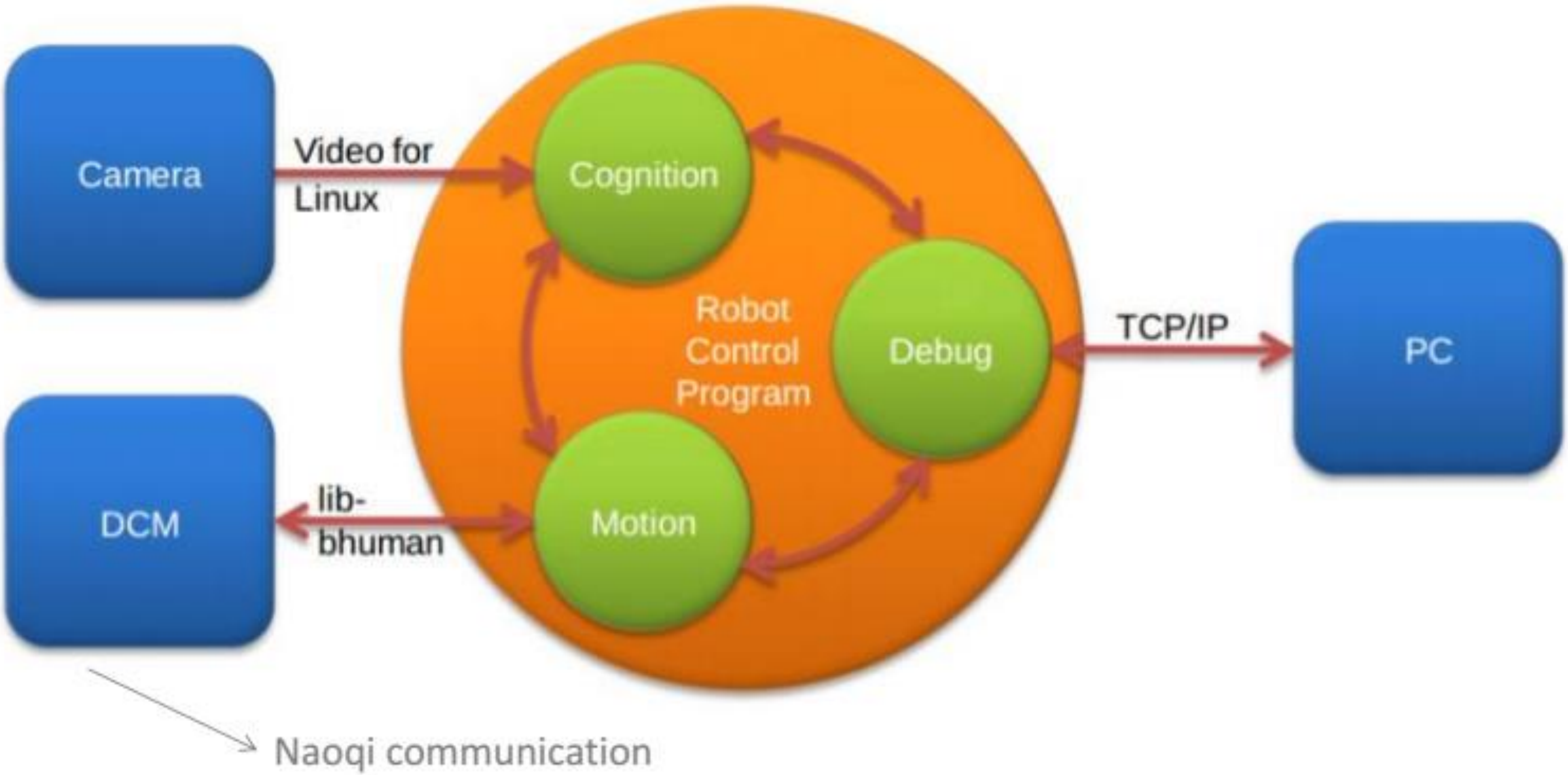
- University of Bremen;
- German Research Center for Artificial Intelligence (DFKI).

Since 2009 used in the Standard Platform League by many teams as a base framework.

Documentation:

http://www.b-human.de/downloads/publications/2015/CodeRelease2015.pdf
http://www.b-human.de/downloads/publications/2014/CodeRelease2014.pdf
http://www.b-human.de/downloads/publications/2013/CodeRelease2013.pdf

B-Human Framework Architecture



Processes

- **Cognition**:

  Inputs: Camera images, Sensor data;

  Outputs: High-level motion commands.

- **Motion**:

  Process high-level motion commands and generates the target vector $\boldsymbol{q}$ for the 25 joints of the Nao.

- **Debug**:

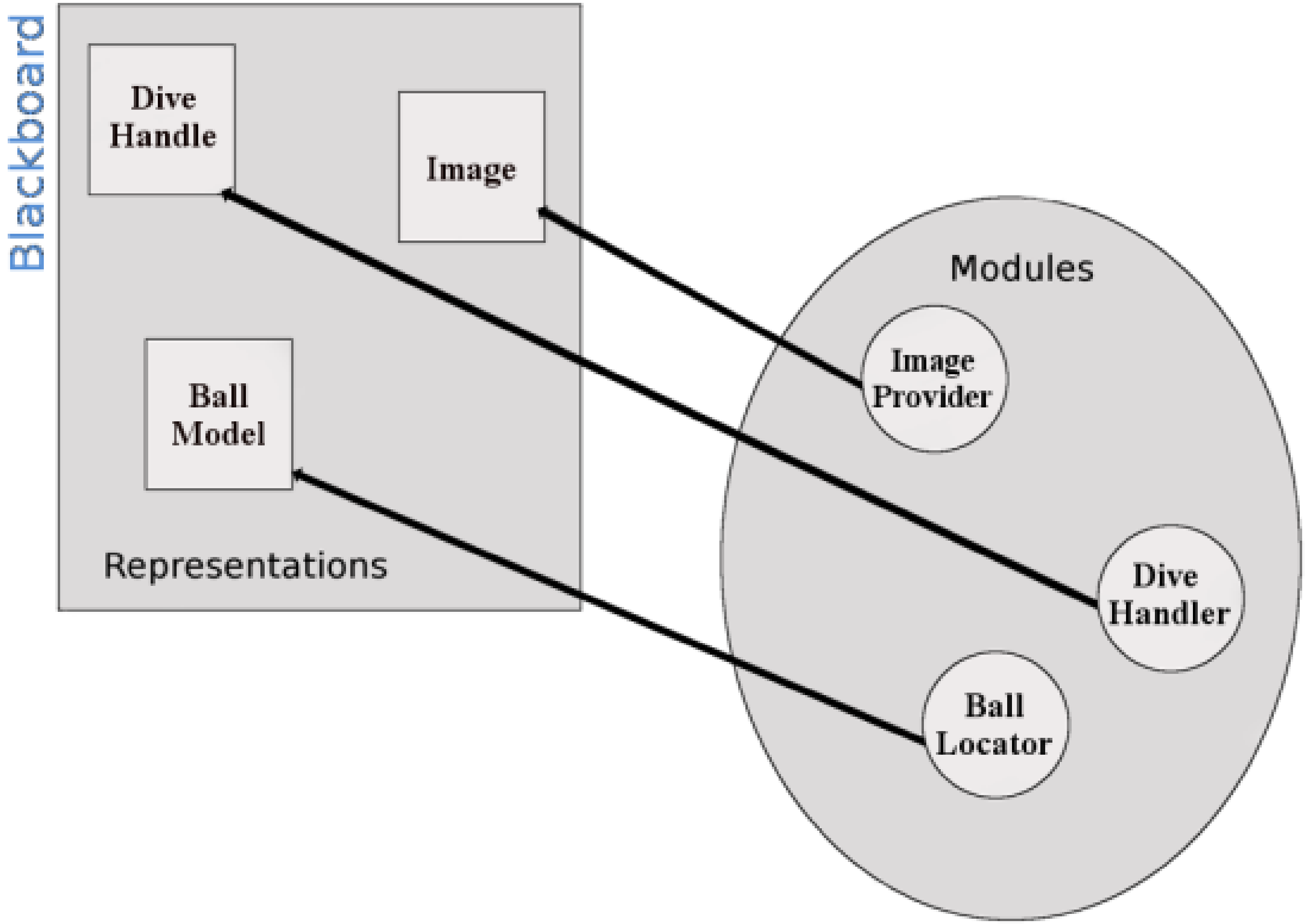
  Communicates with the host PC providing debug information **(e.g. raw image, segmented image, robot pose, etc.)**

Modules and Representations

- The robot control program consists of several modules, each performing a certain task.
- Modules usually require inputs and produce one or more outputs, i.e. representations.

The framework uses a Scheduler to automatically determines the right execution sequence, which depends on the inputs and the outputs of the modules.

Modules and Representations



Representation template

Path to **representations** :

/spqrnao2016/Src/Representations/

```
class Foo :  public Streamable
{
  private:
    void serialize(In* in, Out* out)
    {
      STREAM_REGISTER_BEGIN;
      STREAM(a);
      STREAM(b);
      STREAM_REGISTER_FINISH;
    }

  public:
    float a;
    int b;

    Foo() :  a(0.0), b(0) {;}
};
```

Update modules.cfg

Path to **config** files:
/spqrnao2016/Config/Locations/<location>/

```
representationProviders = [
  ...
  {representation = RobotInfo; provider = GameDataProvider;},
  ...
  {representation = Coordination; provider = Coordinator;},
  {representation = Foo; provider = FooModule;}
];
```

Module template

Path to **modules**: /spqrnao2016/Src/Modules/

Modules performs a certain task requiring specific inputs and providing specific outputs:

- **0...n** Inputs (REQUIRES or USES)
- **1...m** Outputs (PROVIDES)

It must defines an update function for each provided representation.

Module template

```
#include "Tools/Module/Module.h"
#include "Representations/Perception/BallPercept.h"
#include "Representations/SPQR-Representations/Foo.h"

MODULE(FooModule)
  REQUIRES(BallPercept)
  PROVIDES(Foo)
END_MODULE

class FooModule :  public FooModuleBase
{
  private:
    //...
  public:
    FooModule();
    void update(Foo& foo);
};
```

Module template

```
#include "FooModule.h"

MAKE_MODULE(FooModule, SPQR-Modules)

FooModule::FooModule() {;}

void FooModule::update(Foo& foo)
{ if (theBallPercept.wasSeen)
  {
    foo.a = 1.0;
    foo.b = 10;
  }
  else
  {
    foo.a = 2.0;
    foo.b = 5;
  }
}
```

Scheduler

```
MODULE(A)
  PROVIDES(Foo1)
END_MODULE

MODULE(B)
  REQUIRES(Foo1)
  PROVIDES(Foo2)
END_MODULE
```

The execution order is defined by the required representations. In this case module ***B*** cannot be executed before ***A***.

Therefore the order is ***A*** and then ***B***

Scheduler

```
MODULE(C)
  REQUIRES(Foo3)
  PROVIDES(Foo1)
END_MODULE
```

```
MODULE(B)
  REQUIRES(Foo1)
  PROVIDES(Foo2)
END_MODULE
```

Considering input Foo3 as available:

the order is ***C*** and then ***B***

Scheduler

```
MODULE(D)
  REQUIRES(Foo2)
  PROVIDES(Foo1)
END_MODULE
```

```
MODULE(B)
  REQUIRES(Foo1)
  PROVIDES(Foo2)
END_MODULE
```

- ***D*** cannot be executed before ***B***.
- ***B*** cannot be executed before ***D***.

=> Deadlock, the code compiles but it does not execute.
How can we discover deadlock in the structure?

Scheduler

```
MODULE(D)
  USES(Foo2)
  PROVIDES(Foo1)
END_MODULE
```

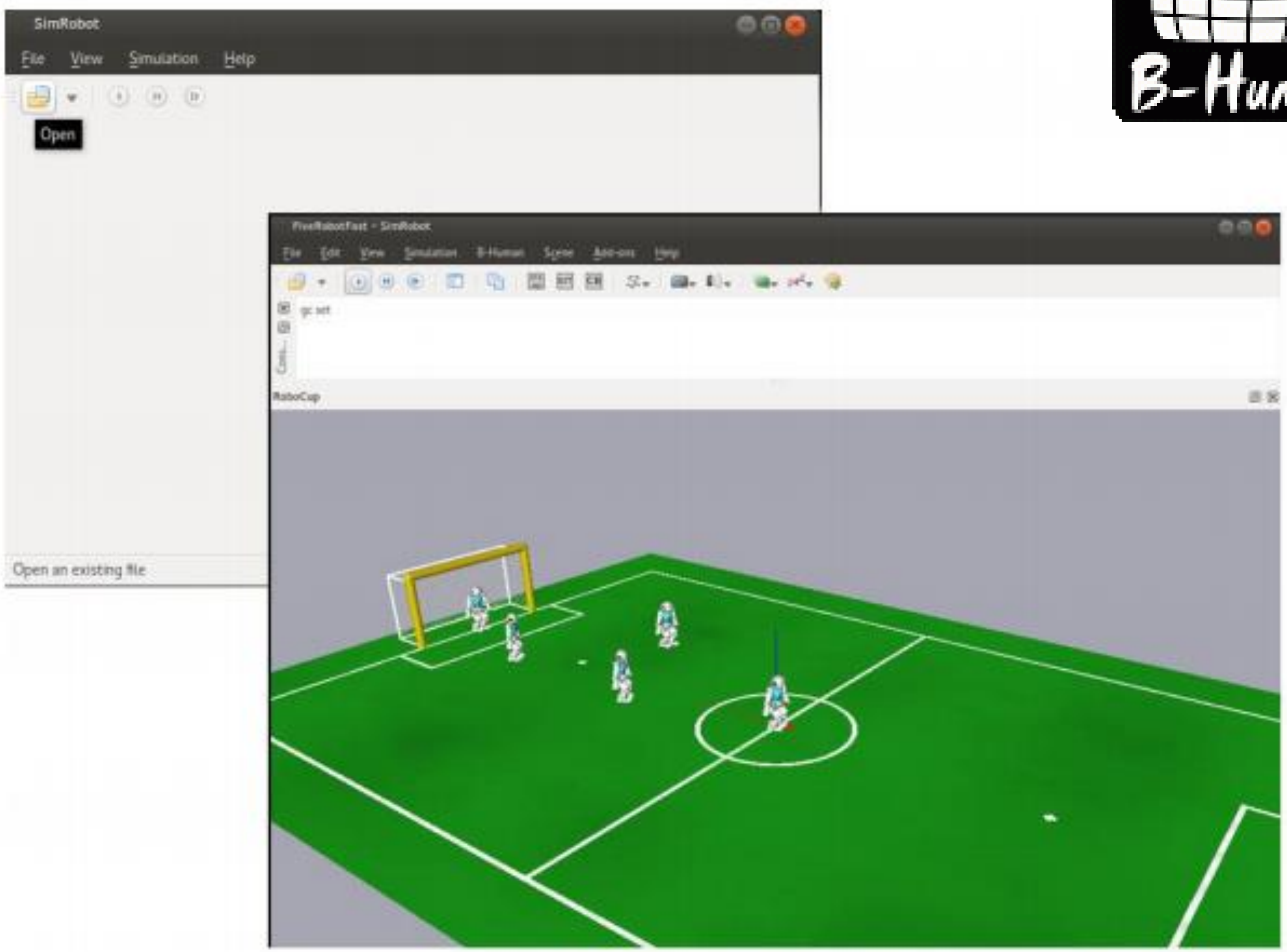
```
MODULE(B)
  REQUIRES(Foo1)
  PROVIDES(Foo2)
END_MODULE
```

D can be executed before **B**.

Warning: USES macro does not guarantees that the representation Foo2 is updated up to the last value.

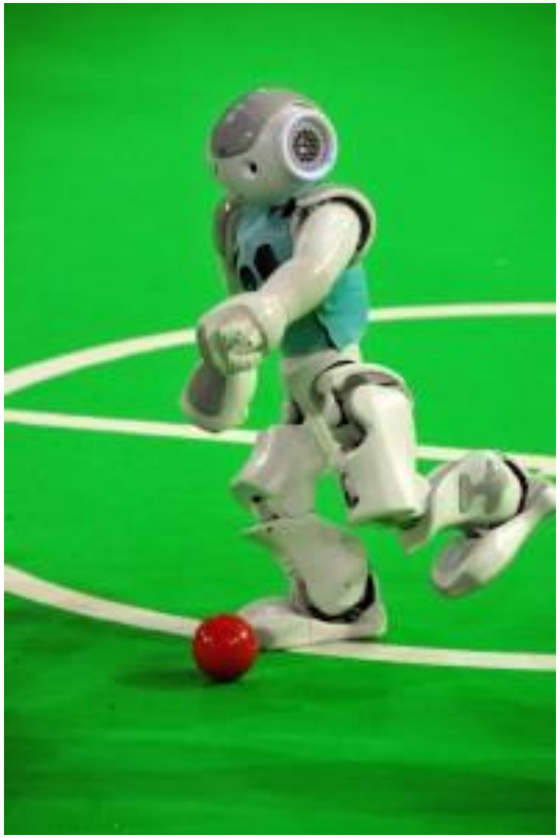
Tip: pay attention to the initialization of the “used” representations

SimRobot

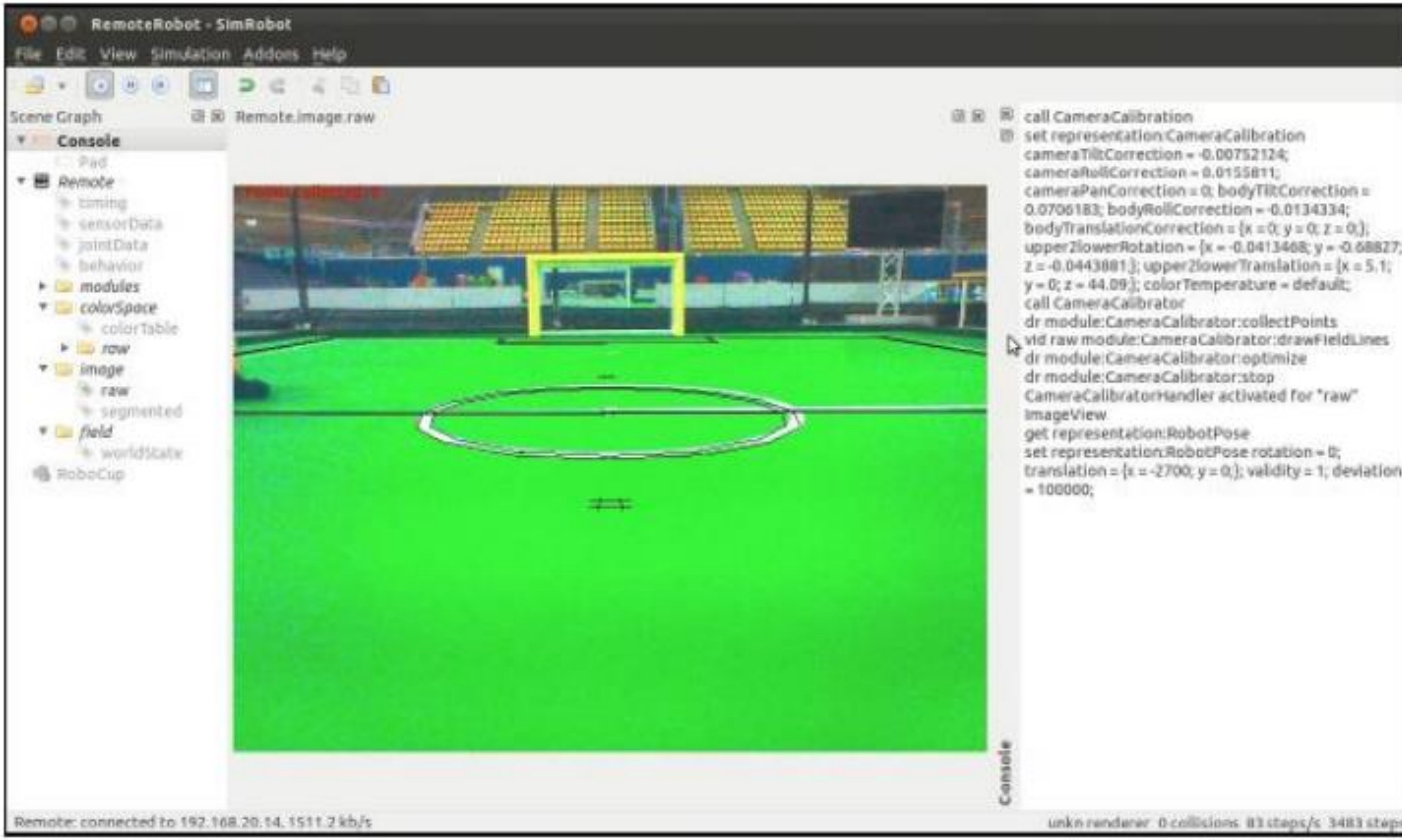

SimRobot
File
View
Simulation
Help
Open
Open an existing file
RoboCup

SimRobot

- ✓ Simulate the code;
- ✓ Connect the robot;
- ✓ Calibrate the color table;
- ✓ Calibrate the camera parameters;
- ✓ Calibrate sensors;



SimRobot


RemoteRobot - SimRobot
File Edit View Simulation Addons Help
Scene Graph
Remote.image.raw
Console
Remote
modules
colorSpace
image
raw
field
call CameraCalibration
set representation:CameraCalibration
cameraTiltCorrection = -0.00752124;
cameraRollCorrection = 0.0155811;
cameraPanCorrection = 0; bodyTiltCorrection =
0.0706183; bodyRollCorrection = -0.0134334;
bodyTranslationCorrection = {x = 0; y = 0; z = 0;};
upper2lowerRotation = {x = -0.0413468; y = -0.68827;
z = -0.0443881;}; upper2lowerTranslation = {x = 5.1;
y = 0; z = -44.09;}; colorTemperature = default;
call CameraCalibrator
dr module:CameraCalibrator:collectPoints
vid raw module:CameraCalibrator:drawFieldLines
dr module:CameraCalibrator:optimize
dr module:CameraCalibrator:stop
CameraCalibratorHandler activated for "raw"
ImageView
get representation:RobotPose
set representation:RobotPose rotation = 0;
translation = {x = -2700; y = 0;}; validity = 1; deviation
= 100000;
Console
Remote: connected to 192.168.20.14, 1511.2 kb/s
unkn renderer 0 collisions 83 steps/s 3483 steps

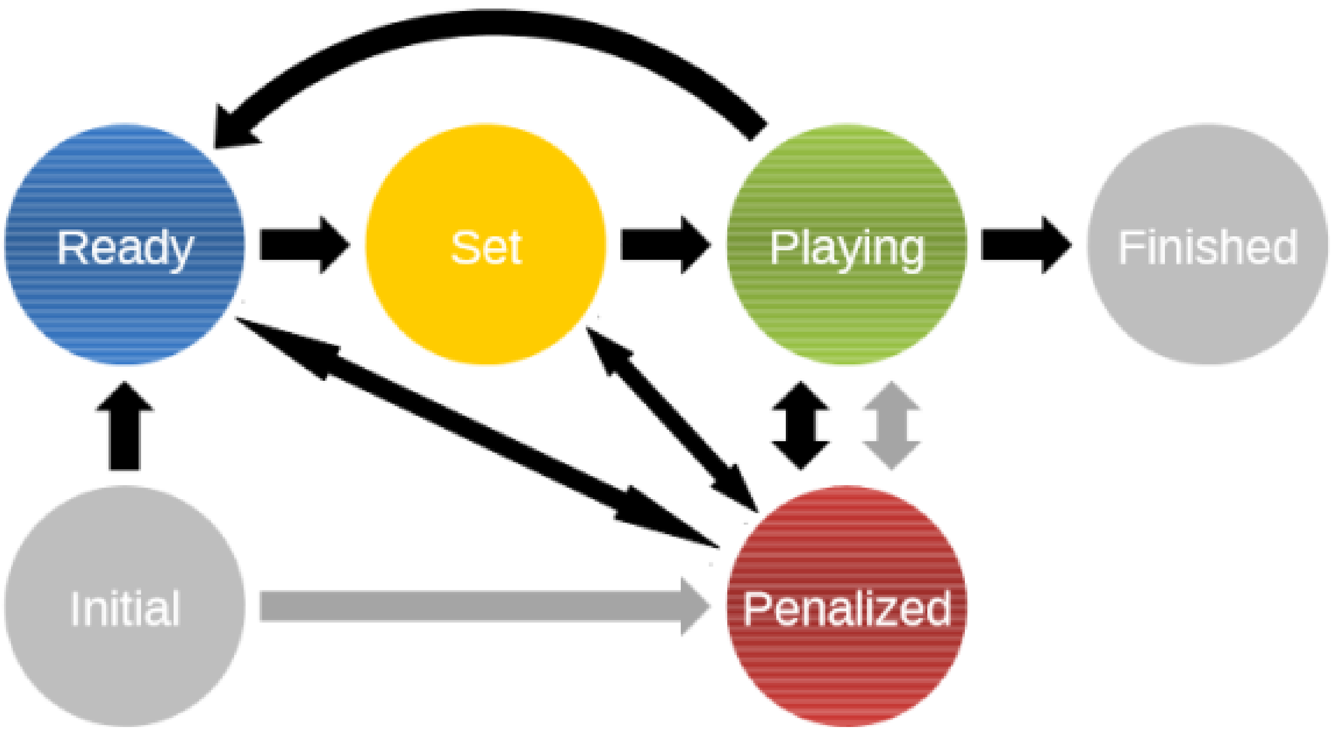
SPQR code: tips and useful paths

✓ bash_aliases;
✓ compile in Develop;
✓ Use grep: $ grep –r "<string>" .*

Paths (move to the **RoboCup/spqrnao2016/** folder)

- SimRobot: Build/ SimRobot/Linux/<Debug/Release/Develop/>
- Make: Make/Linux/
- Install: install/
- Scenes: Config/Scenes/
- Locations: Config/Locations/
- Behaviours: Src/Modules/BehaviorControl/
- Options.h: Src/Modules/BehaviorControl/BehaviorControl2015/Options.h
  Look at this file if you want to add options

Game States



SimRobot console commands

gc ready: the robot runs the ready behavior and gets into their default position;

gc set: places the robot into the default set positions;

gc playing: starts the game;

mr RobotPose OracledWorldModelProvider: if you want to provide a perfect localization.

10 mins break?

C-based Agent Behavior Specification Language (CABSL)

It is a derivative of **XABSL: eXtensible Agent Behavior Specification Language**

It is designed to describe and develop an agent's behavior as a **hierarchy of state machines**.

CABSL solely consists of C++ preprocessor macros and can be compiled with a normal C++ compiler.

A behavior consists of a set of **options** that are arranged in an **option graph**.

CABSL

Adopted by the German Team since the RoboCup 2002

Good choice to describe behaviors for autonomous robots or NPCs in computer games.

http://www.xabsl.de

CABSL:
General structure

CABSL comprises few basic elements: options, states, transitions, actions.

Each option is a **finite state machine** that describes a specific part of the behavior such as a skill or a head motion of the robot, or it combines such basic features.

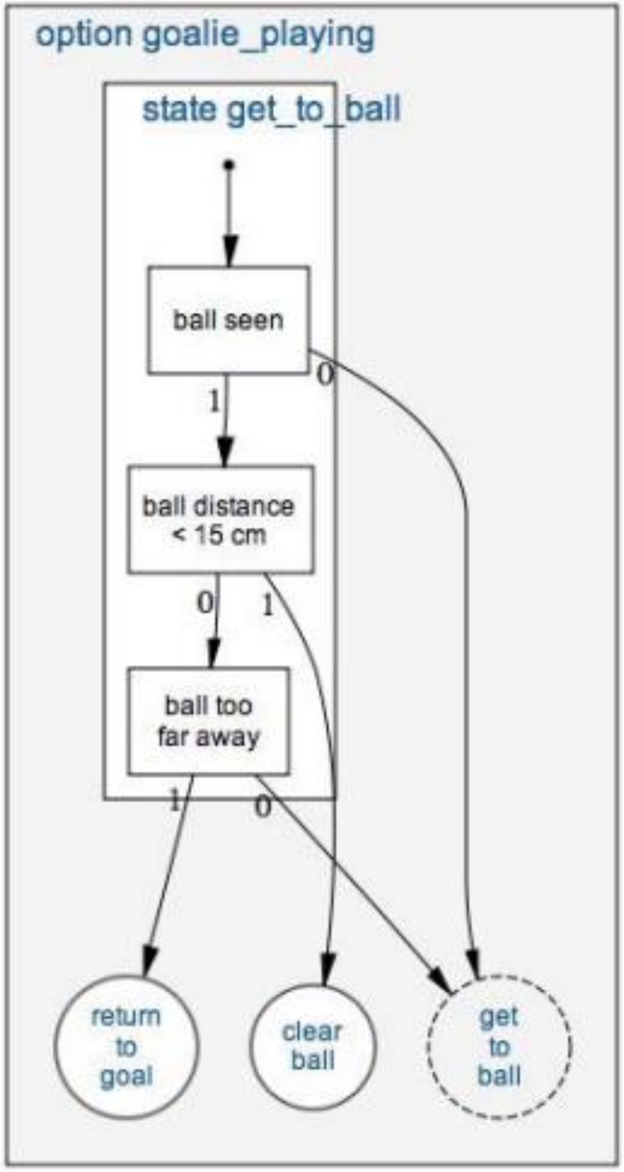
Tip: Deeply debug the inner state machine in order to avoid loops.

CABSL: Options

Each **state** has a decision tree with transitions to other states.

For the decisions, other sensory information (representations) can be used.

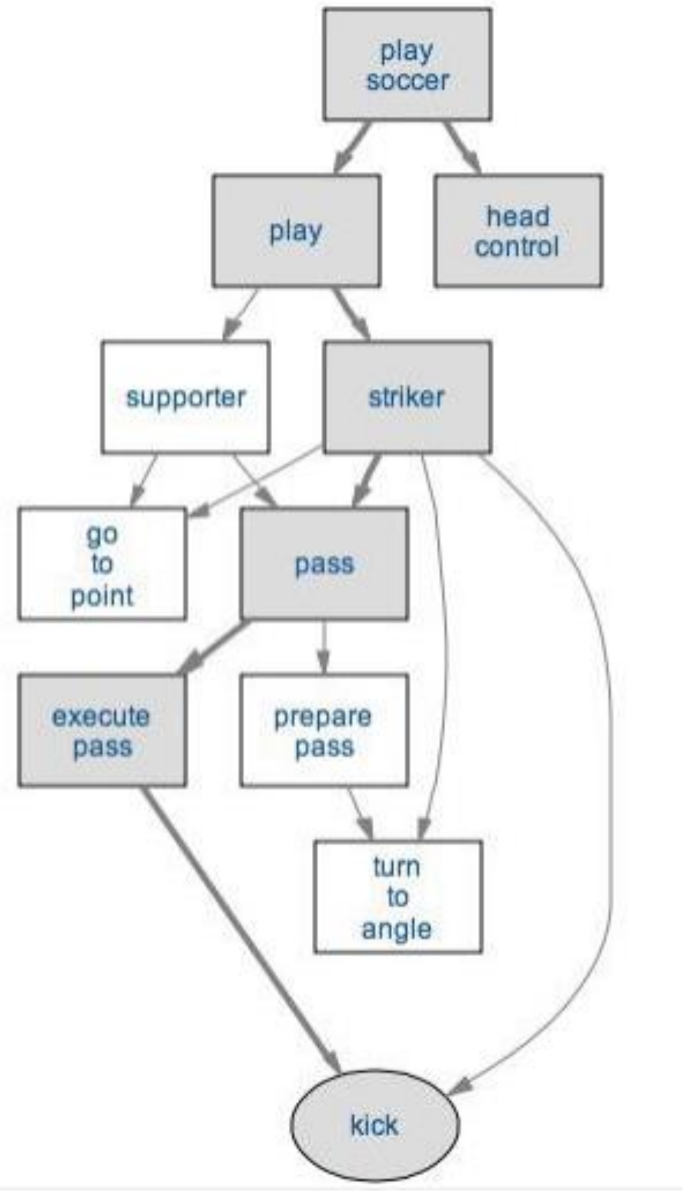
Tip: take into account how long the state has been active



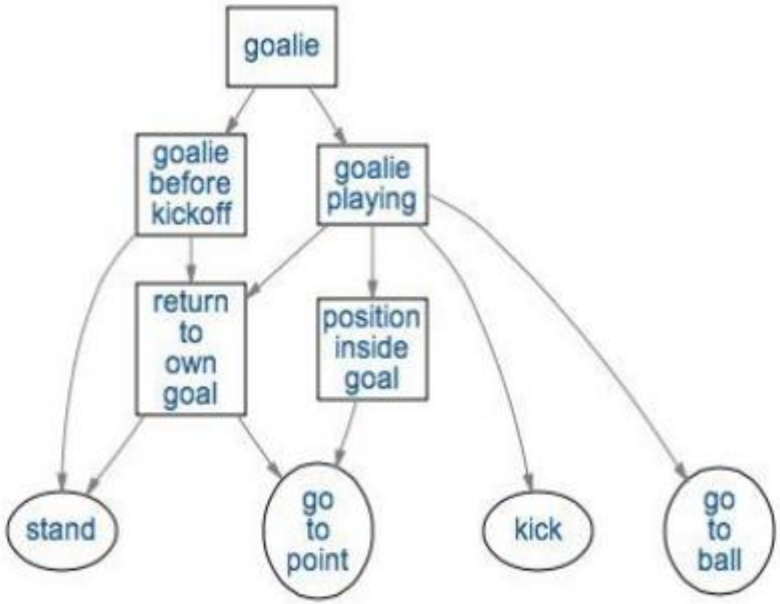
CABSL: Options

Options are activated at a specific time step from a rooted tree.

Such tree is a sub-tree of the more general option graph and it's called **option activation tree**.



CABSL: Options



Pseudo-code:

Foreach iteration

{

the execution of the tree **starts** from the root and controls the flux of the option graph top-down;

do

{

if the transition is within the current node **continue** the execution;

else jump to the lower level;

} **until** current node is a leaf node;

}

Task of the option graph:

activate one of the leaf behaviors (proceeding top-down), which is then executed.

CABSL: Libraries

```
class LibExample : public LibraryBase
{
public:
  LibExample();
  void preProcess() override;
  void postProcess() override;
  bool boolFunction(); // Sample method
};
```

CABSL examples and templates

CABSL: Options

```
option(exampleOption)
{
  initial_state(firstState)
  {
    transition
    {
      if(booleanExpression)
        goto secondState;
      else if(libExample.boolFunction())
        goto thirdState;
    }
    action
    {
      providedRepresentation.value = requiredRepresentation.value * 3;
    }
  }
```

CABSL: Options

```
state(secondState)
{
  action
  {
    SecondOption();
  }
}
```

Warning: Pay attention to this kind of states.

CABSL: Options

```
state(thirdState)
{
  transition
  {
    if(booleanExpression)
      goto firstState;
  }
  action
  {
    providedRepresentation.value = RequiredRepresentation::someEnumValue;
    ThirdOption();
  }
}
```

Parallelism through the activation graph.

CABSL: Options

```
option(OptionWithParameters, int i, bool b, int j = 0)
{
  initial_state(firstState)
  {
    action
    {
      providedRepresentation.intValue = b ? i : j;
    }
  }
}
```

Arguments can generalize the options.

CABSL: Options

```
common_transition
{
  if(booleanExpression)
    goto firstState;
  else if(booleanExpression)
    goto secondState;
}
```

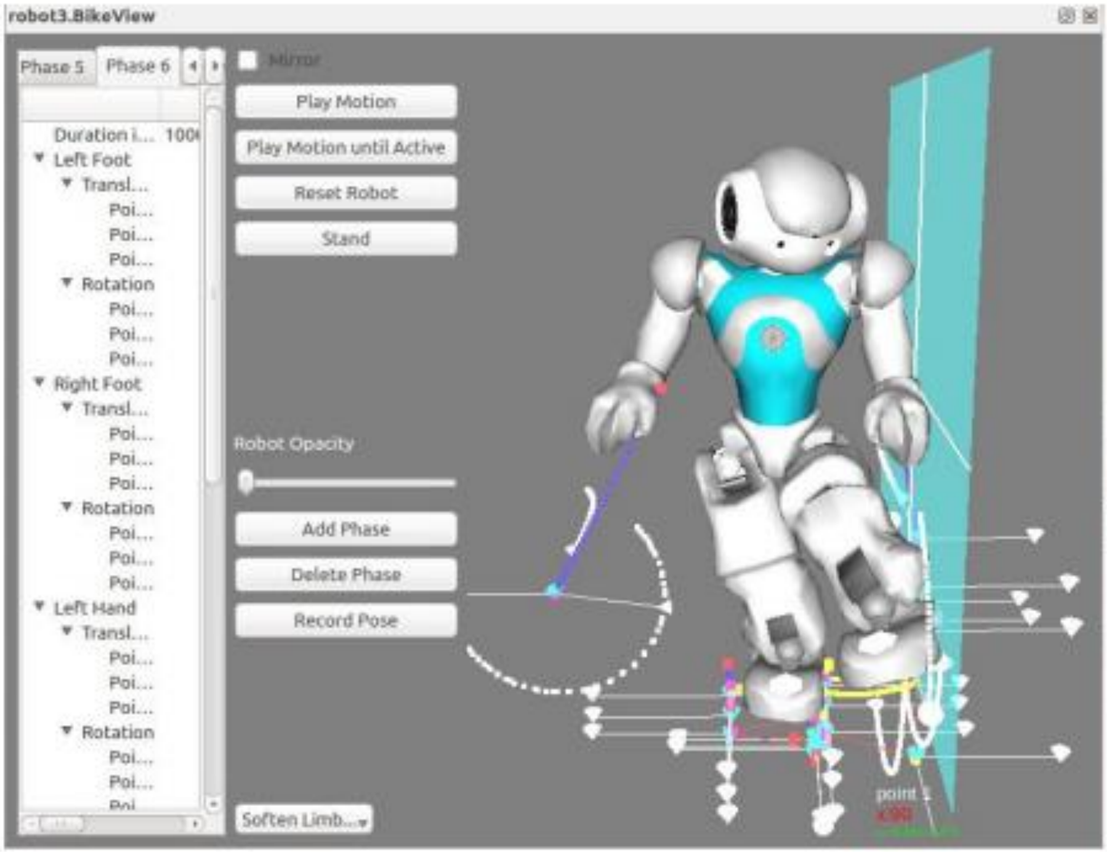
CABSL: add representations to the Behaviors Engine

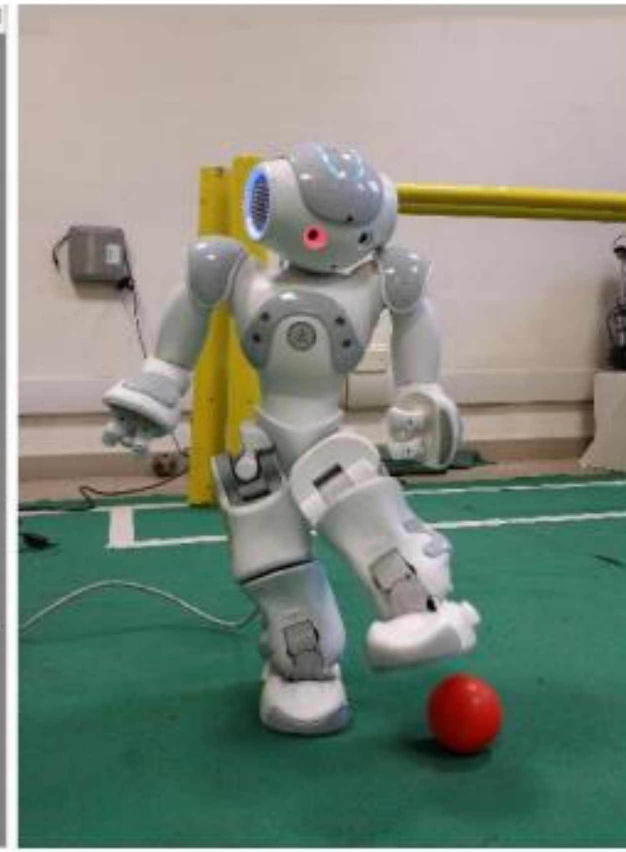
BehaviorControl2013.h <Select Symbol>

```
#include "Representations/Sensing/FallDownState.h"
#include "Representations/Sensing/FootContactModel.h"
#include "Representations/Sensing/GroundContactState.h"
#include "Representations/Sensing/TorsoMatrix.h"

#include "Representations/SPQR-Representations/ConfigurationParameters.h"
#include "Representations/SPQR-Representations/RobotPoseSpqrFiltered.h"
#include "Representations/SPQR-Representations/GlobalBallEstimation.h"
#include "Representations/SPQR-Representations/Coordination.h"
#include "Representations/SPQR-Representations/DiveHandle.h"
#include "Representations/SPQR-Representations/BallPrediction.h"

#include <Core/Processors/Processor.h>

#include <limits>
#include <algorithm>
#include <map>
#include <fstream>

MODULE(BehaviorControl2013)
  REQUIRES(GlobalBallEstimation)
  REQUIRES(RobotPoseSpqrFiltered)
  REQUIRES(Coordination)
  REQUIRES(DiveHandle)
  REQUIRES(BallPrediction)

  REQUIRES(ArmContactModel)
  REQUIRES(ArmMotionEngineOutput)
  REQUIRES(BallModel)
  REQUIRES(BallTakingOutput)
  REQUIRES(BikeEngineOutput)
  REQUIRES(CameraInfo)
  REQUIRES(CameraMatrix)
  REQUIRES(CombinedWorldModel)
```

Motion interface: Bike scene

Bikes: spqrnao2016/Config/Kicks/





Ball recognition and evaluation

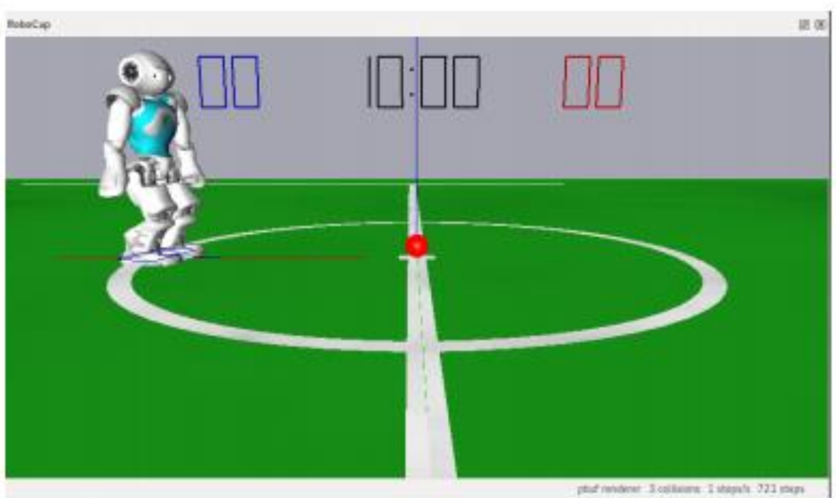
BallPercept.h

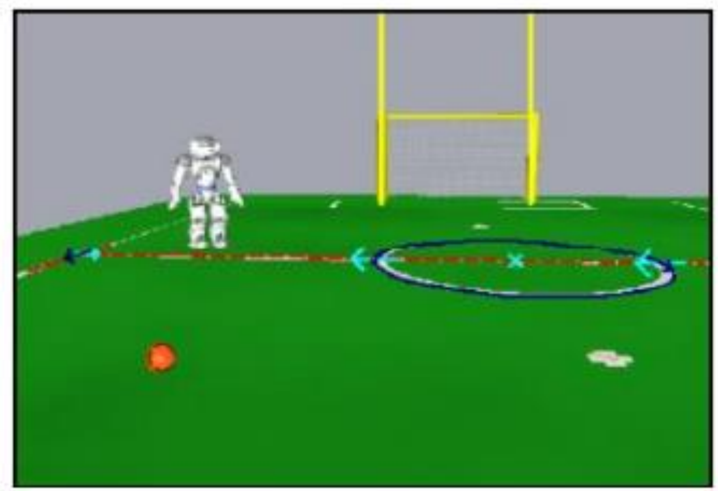
- USES BallModel
- PROVIDES BallPercept

BallModel.h

- REQUIRES BallPercept
- USES BallModel
- PROVIDES BallModel

1. Evaluate ball spots;
2. Check noise;
3. Calculate ball in image;
4. Calculate ball on field;
5. Check jersey;





SPQR Code: Hands In

Github repo: **https://github.com/SPQRTeam/spqrnao2016**

1.A Make an account on github.com, send an email to **bipeds-spqr@googlegroups.com** with your git username (“[Lab NAO RoboCup] Name LastName” as email subject) and install the software;

1.B Create a new Representation and a new Module: the update function of the module has to display:

- the robot pose <x, y, theta>;
- the ball position <x, y> (both relative and global);
- joints value;

2.A Filter the ball perception and make the robot disregard balls that are more then 2 meters away from the robot;