SAPIENZA
UNIVERSITÀ DI ROMA

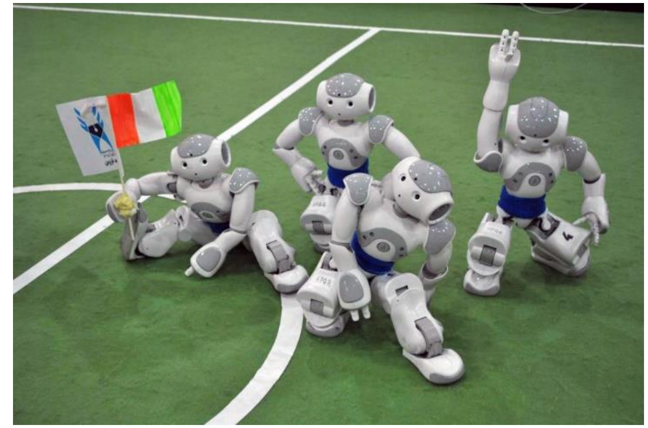
Artificial Intelligence

Prof. Daniele Nardi

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

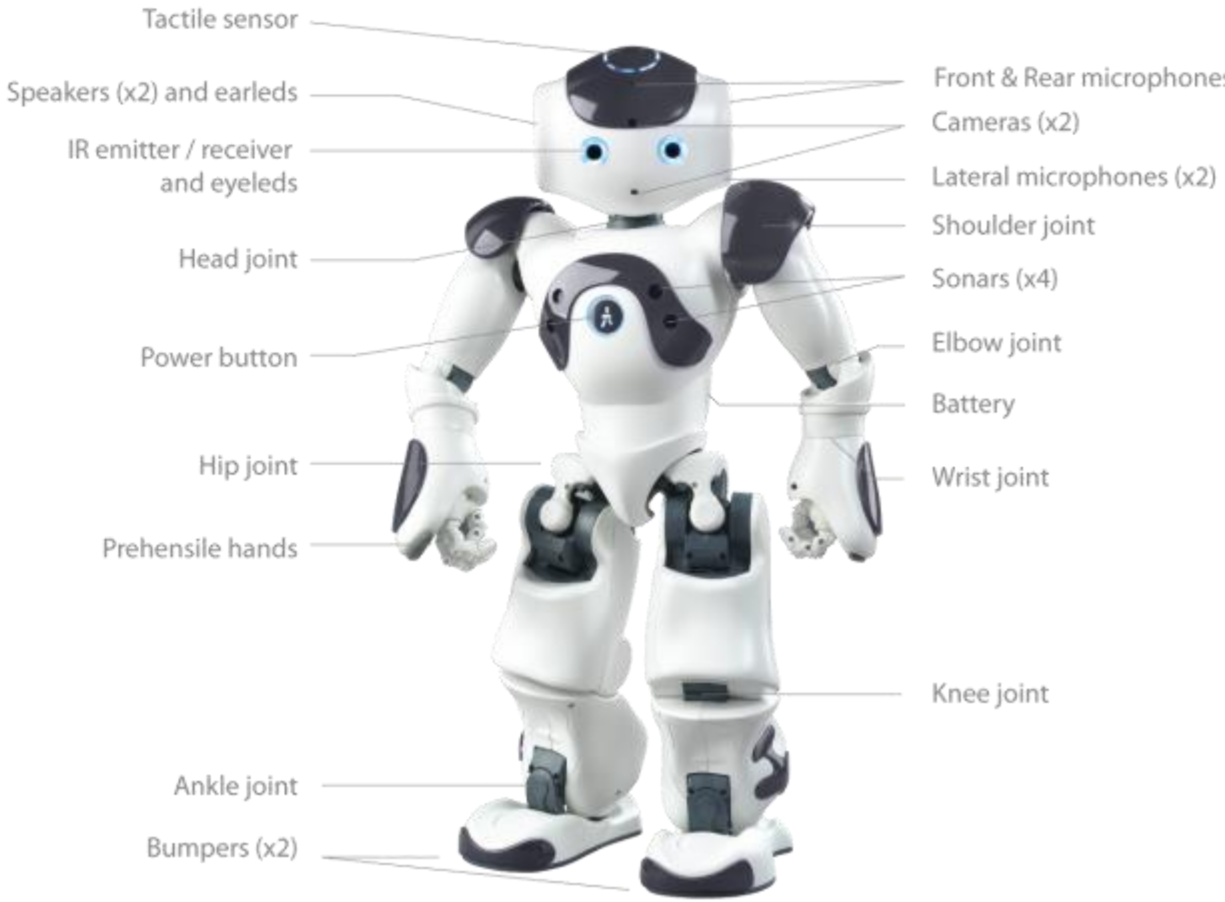
- Middle-size 1998-2002;
- Four-legged 2000-2007;
- Real-Rescue robots since 2003;
- Virtual-Rescue robots since 2006;
- Standard Platform League since 2008;

http://spqr.diag.uniroma1.it

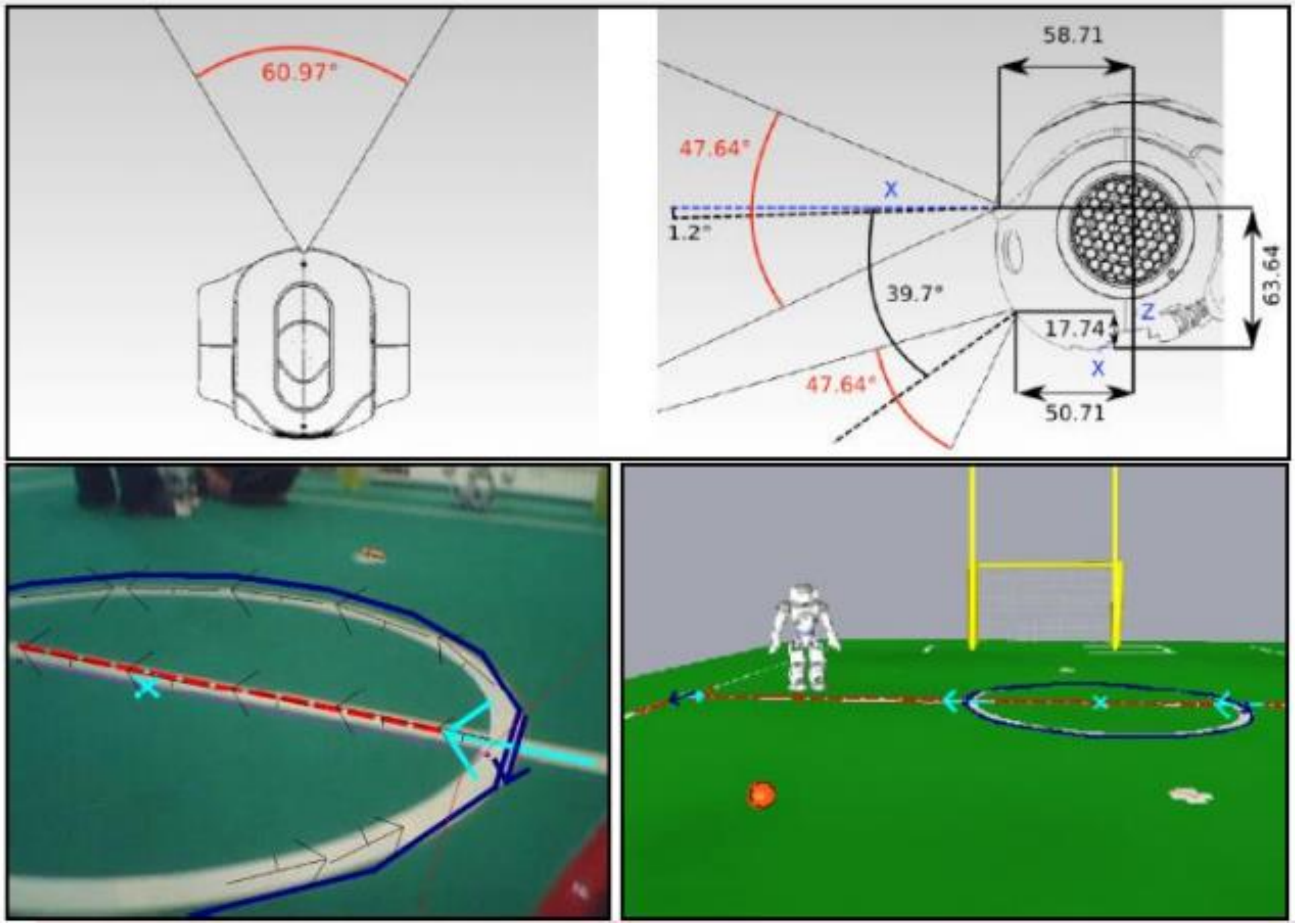
The Aldebaran Nao robot

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



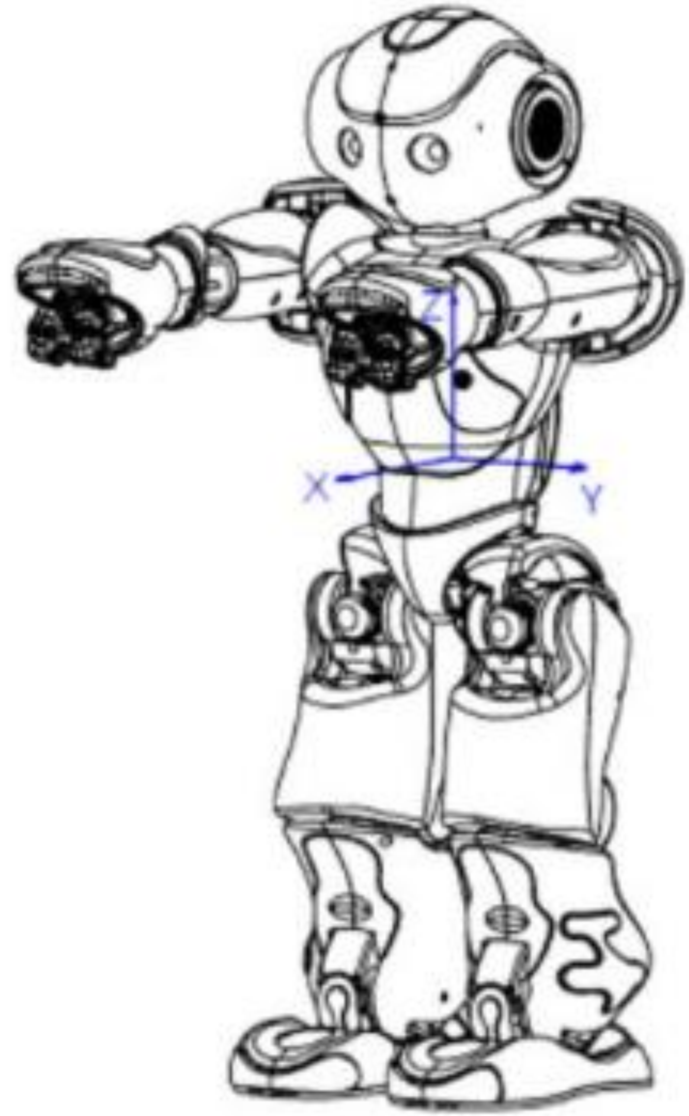
Camera Nao



Inertial Unit

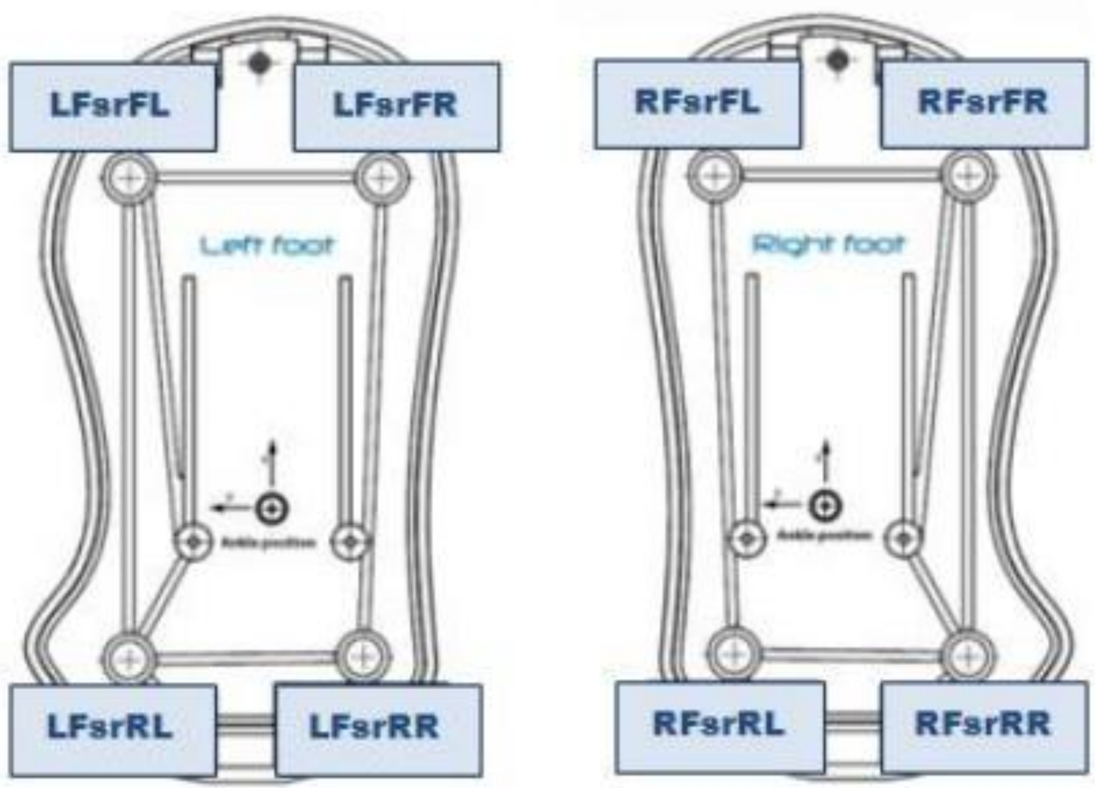
- 3 axes gyrometers
- 2 axis accelerometers

The Inertial unit is located in the torso



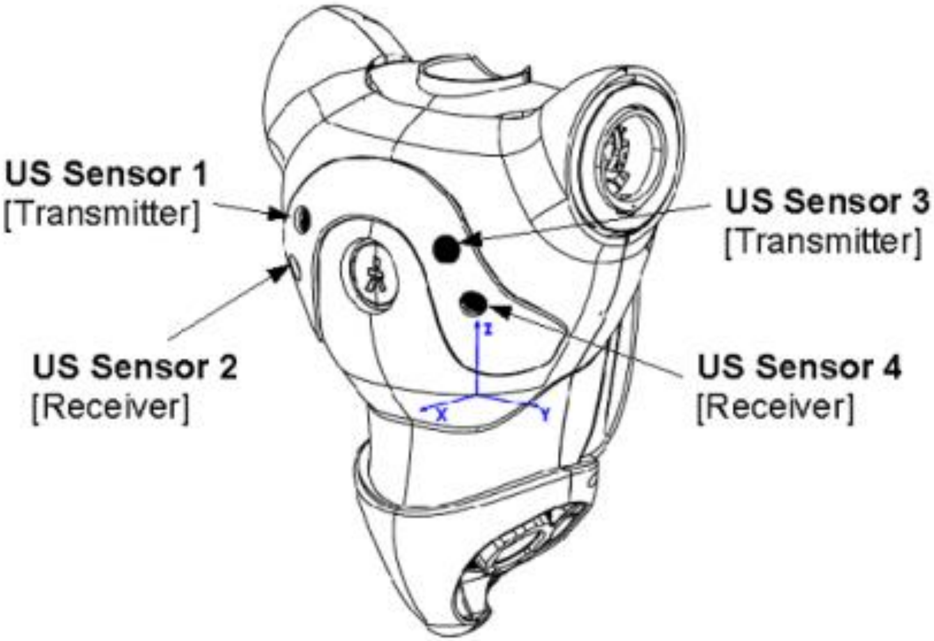
FSR – Force Sensitive Resistor

To measure a resistance change according to the pressure applied

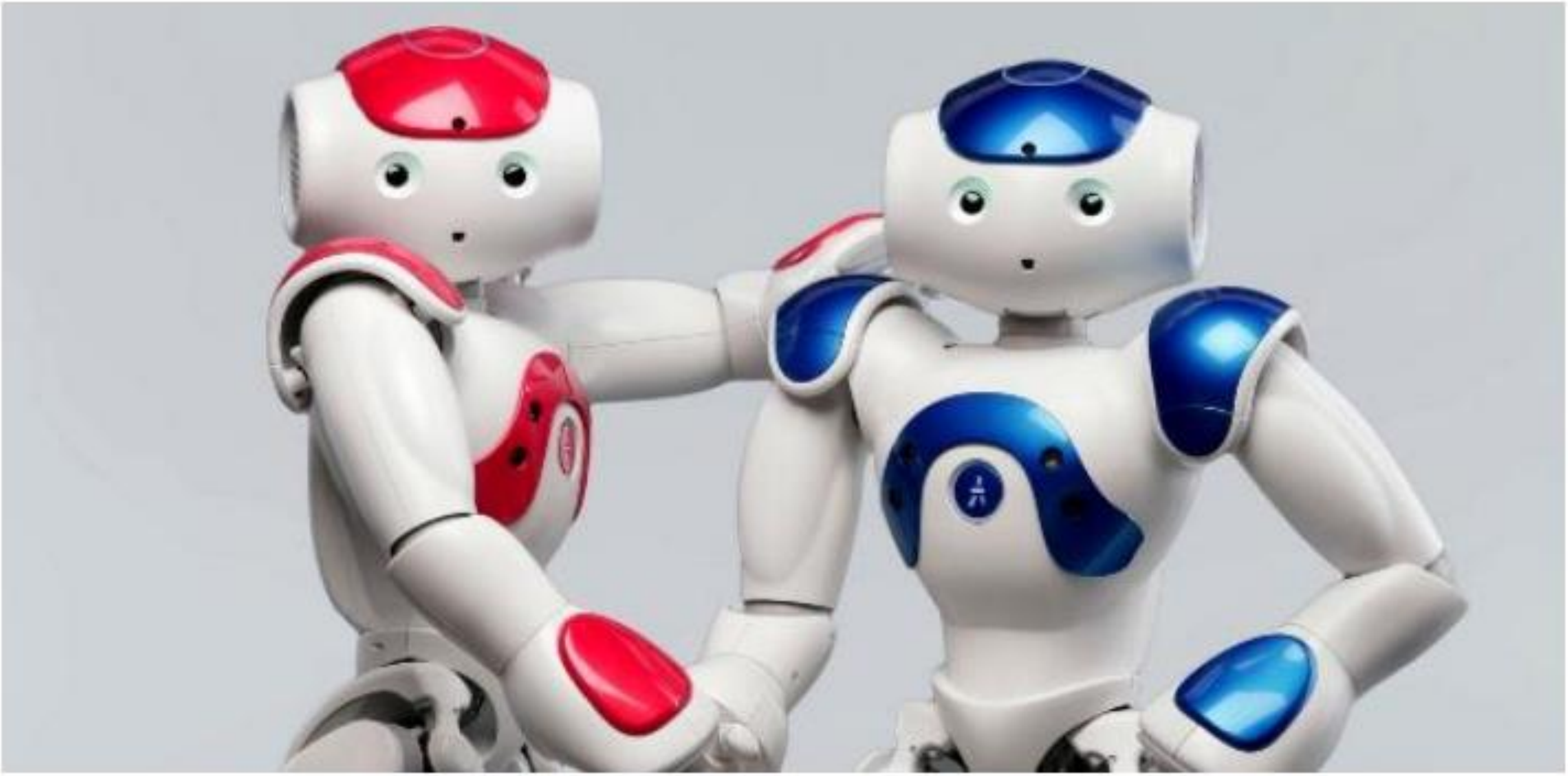


Sonars

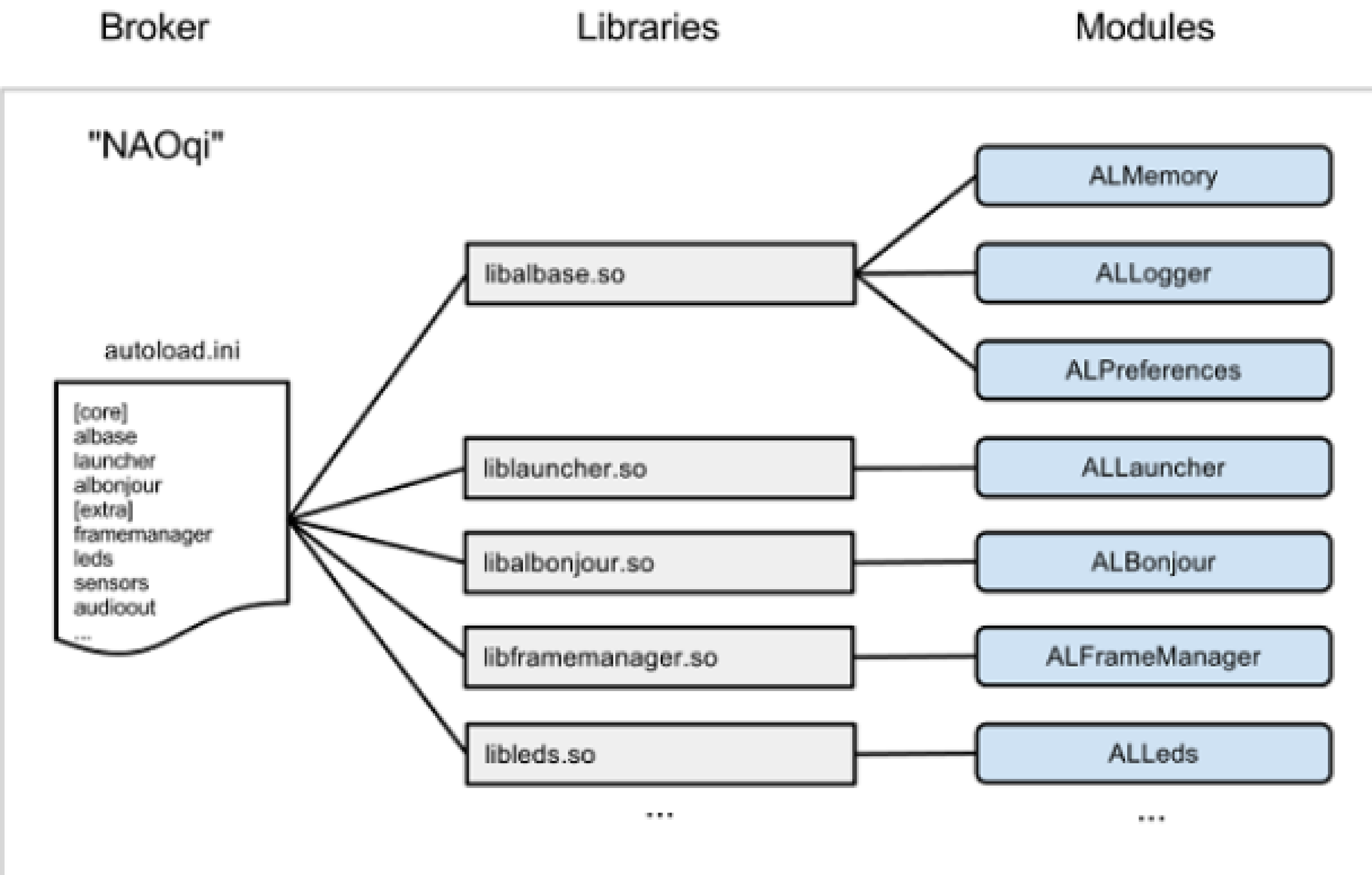
- Resolution: 1cm
- Frequency: 40kHz
- Detection range: 0.25m -2.55m
- Effective cone: 60°



Nao Robot Software Support

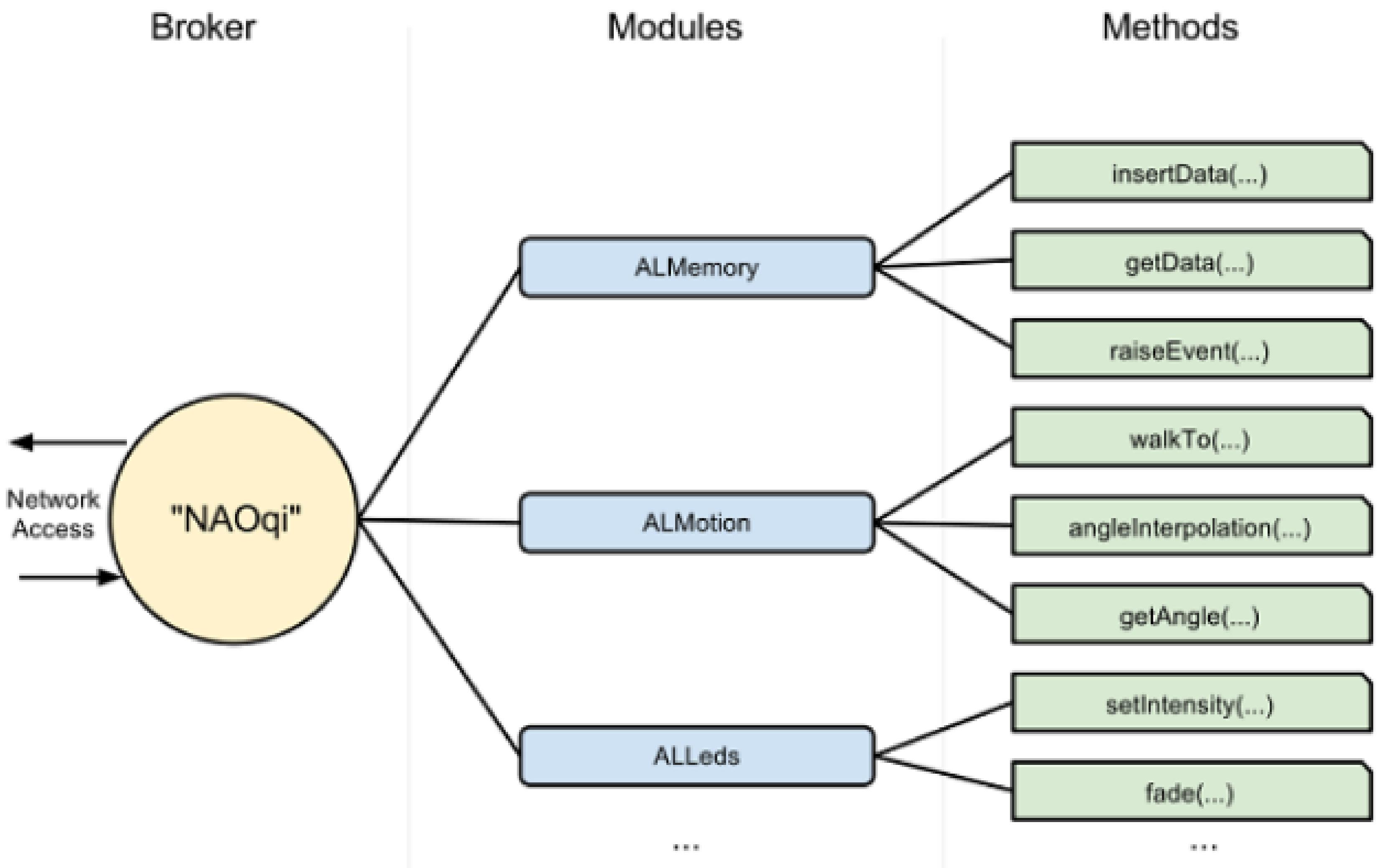


Naoqi API



Broker
Libraries
Modules
"NAOqi"
autoload.ini
[core]
albase
launcher
albonjour
[extra]
framemanager
leds
sensors
audioout
...
libalbase.so
liblauncher.so
libalbonjour.so
libframemanager.so
libleds.so
...
ALMemory
ALLogger
ALPreferences
ALLauncher
ALBonjour
ALFrameManager
ALLeds
...

Naoqi API



Broker
Modules
Methods
Network Access
"NAOqi"
ALMemory
ALMotion
ALLeds
...
insertData(...)
getData(...)
raiseEvent(...)
walkTo(...)
angleInterpolation(...)
getAngle(...)
setIntensity(...)
fade(...)
...

Naoqi API

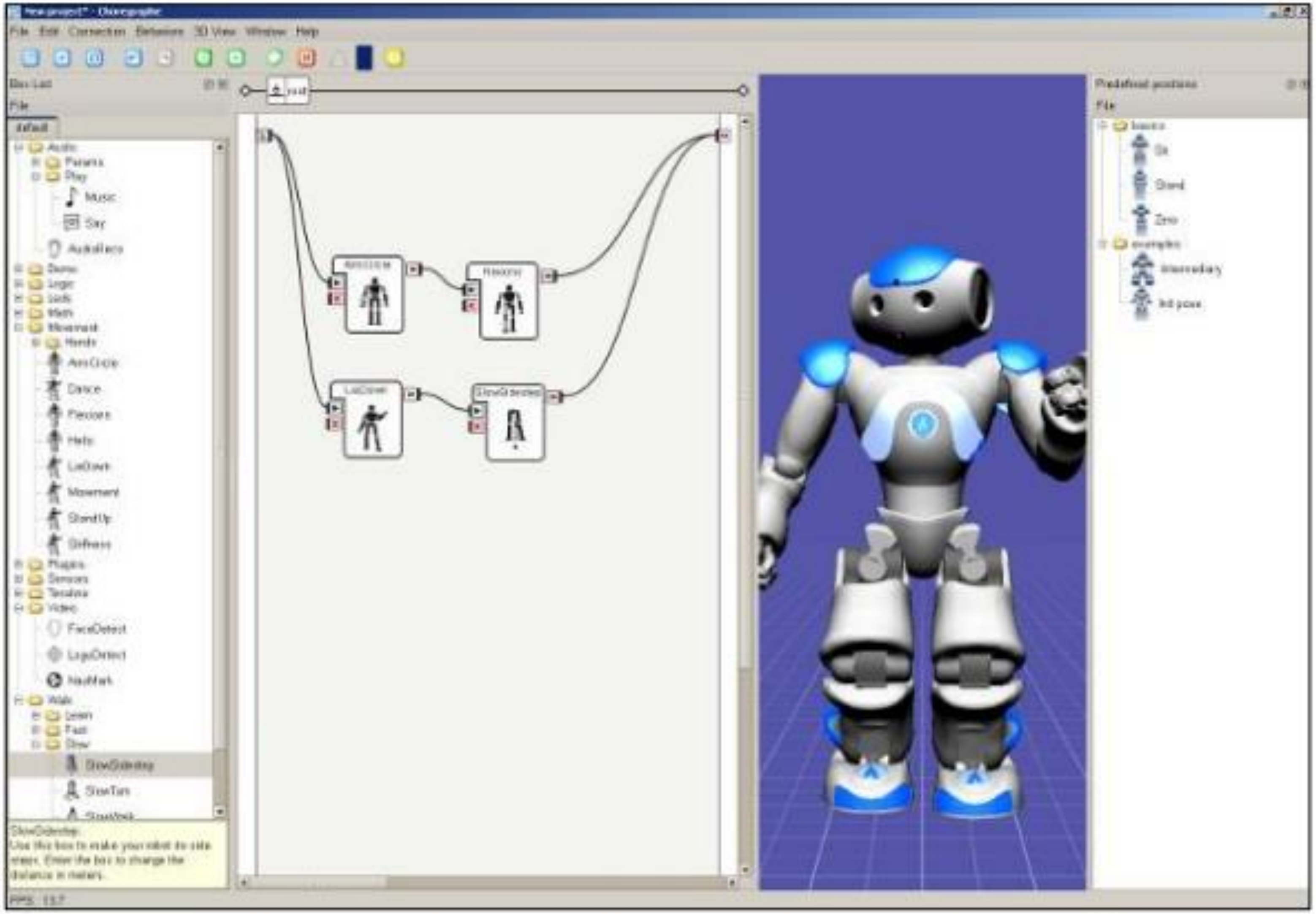
The **broker** allows the access to modules and methods within.

A **proxy** is an instantiation of a module
if you create a proxy to the ALMotion module, you will have 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

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

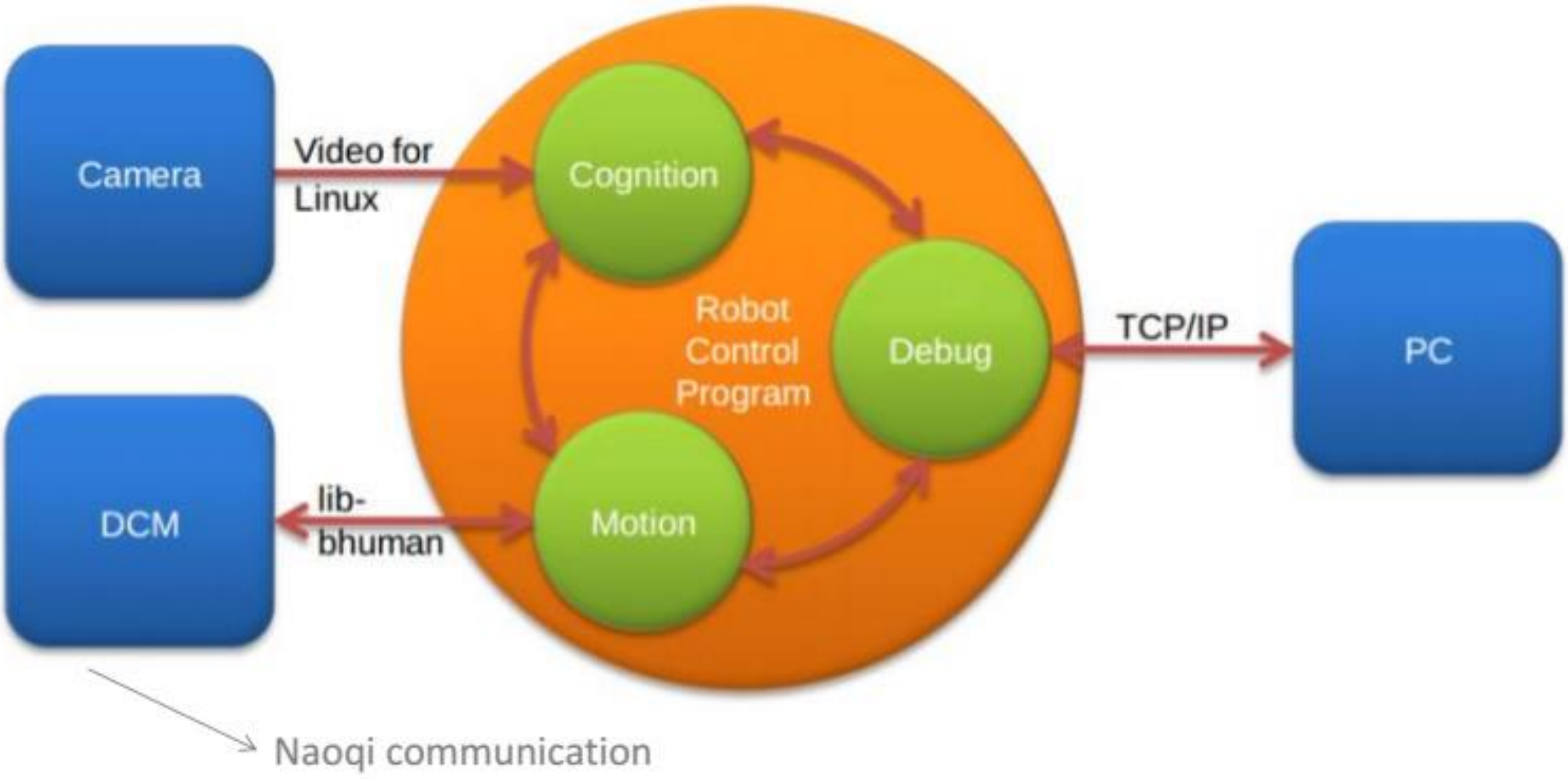
- University of Bremen;
- German Research Center for Artificial Intelligence (DFKI).

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

Documentation:

http://www.b-human.de/downloads/publications/2017/coderelease2017.pdf
http://www.b-human.de/downloads/publications/2016/coderelease2016.pdf

B-Human Framework



Processes

- **Cognition**:

  Inputs: Camera images, Sensor data;

  Outputs: High-level motion commands.

- **Motion**:

  Process high-level motion commands and generates the target vector ***q*** for the joints of the Nao.

- **Debug**:

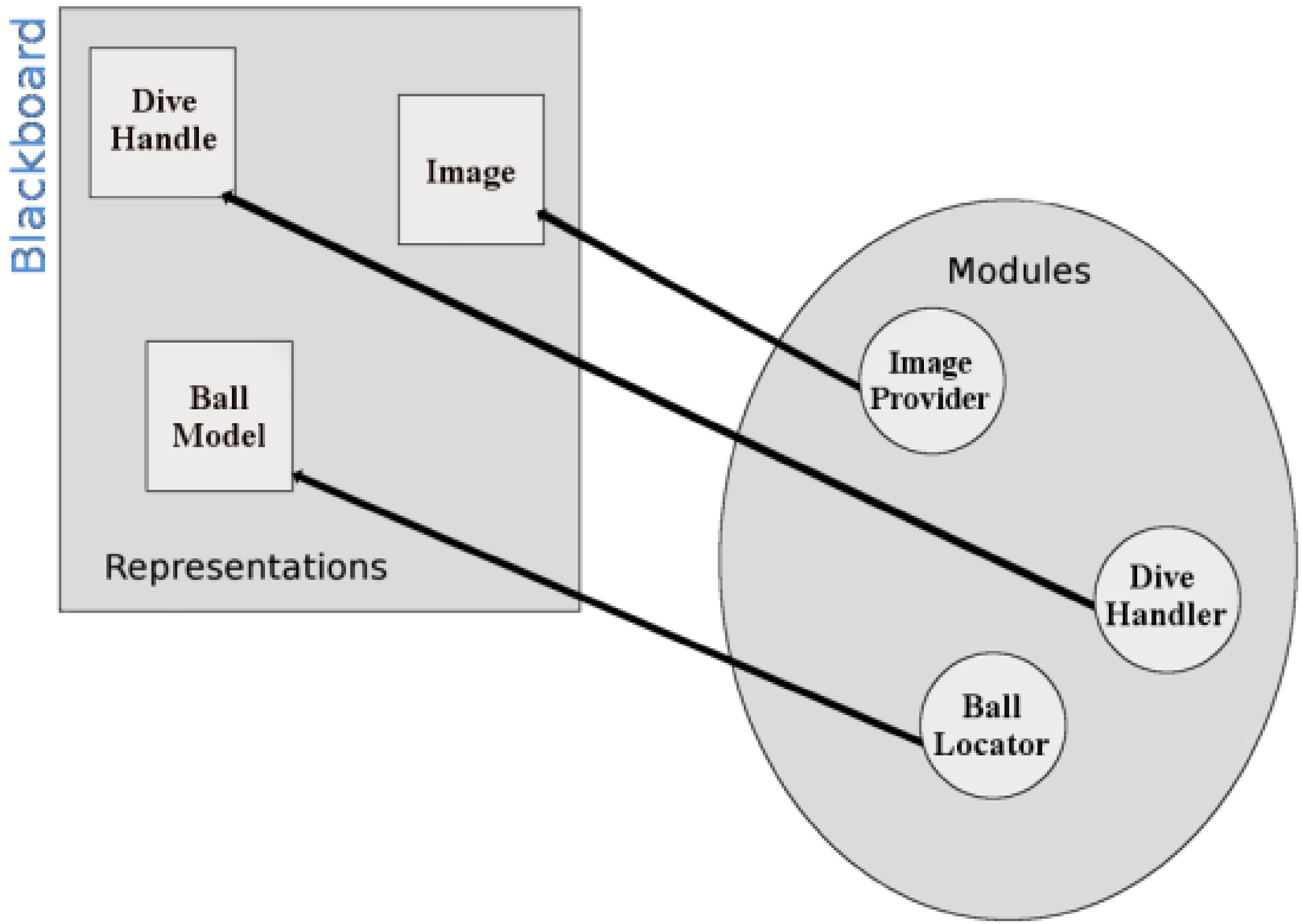
  Communicates with the host PC providing debug information **(e.g. raw image, segmented image, robot pose, etc.)**

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 (called **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

```
#pragma once
#include "Tools/Streams/Enum.h"
#include "Tools/Math/Eigen.h"
#include "Tools/Streams/AutoStreamable.h"
class MyRepresentation : public Streamable
{
  private :
    void serialize(In *in, Out *out)
    {
      STREAM_REGISTER_BEGIN;
      STREAM(robotPose);
      STREAM(ballPose);
      STREAM_REGISTER_FINISH;
    }
  public:
    Vector2f ballPose;
    Vector2f robotPose;

    MyRepresentation(){;}
};
```

Update modules.cfg

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

Module template

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

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

They have to implement an update function for each provided representation

Module template

```cpp
#include "Representations/spqr_representations/MyRepresentation.h"
#include "Representations/Perception/BallPercepts/BallPercept.h"
#include "Representations/Modeling/RobotPose.h"
#include "Tools/Module/Module.h"
#include < iostream >

MODULE (MyRepresentationProvider,
{,
  REQUIRES(BallPercept),
  REQUIRES(RobotPose),
  PROVIDES(MyRepresentation),
});

class MyRepresentationProvider : public MyRepresentationProviderBase {
  public :
  MyRepresentationProvider();
  void update(MyRepresentation &myRepresentation);
};
```

Module template

```cpp
#include "MyRepresentationProvider.h"
MAKE_MODULE(MyRepresentationProvider , spqr_modules)

MyRepresentationProvider::MyRepresentationProvider (){}

void MyRepresentationProvider::update(MyRepresentation &myRepresentation){
  myRepresentation.ballPose = theBallPercept.positionOnField;
  myRepresentation.robotPose = theRobotPose.translation;

  std::cout << "ballpose = " << myRepresentation.ballPose.transpose() << std::endl;
  std::cout << "robotpose = " << myRepresentation.robotPose.transpose() << std::endl;
}
```

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***.

Scheduler

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

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

Assuming that Foo3 is available, which is the scheduled order?

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
```

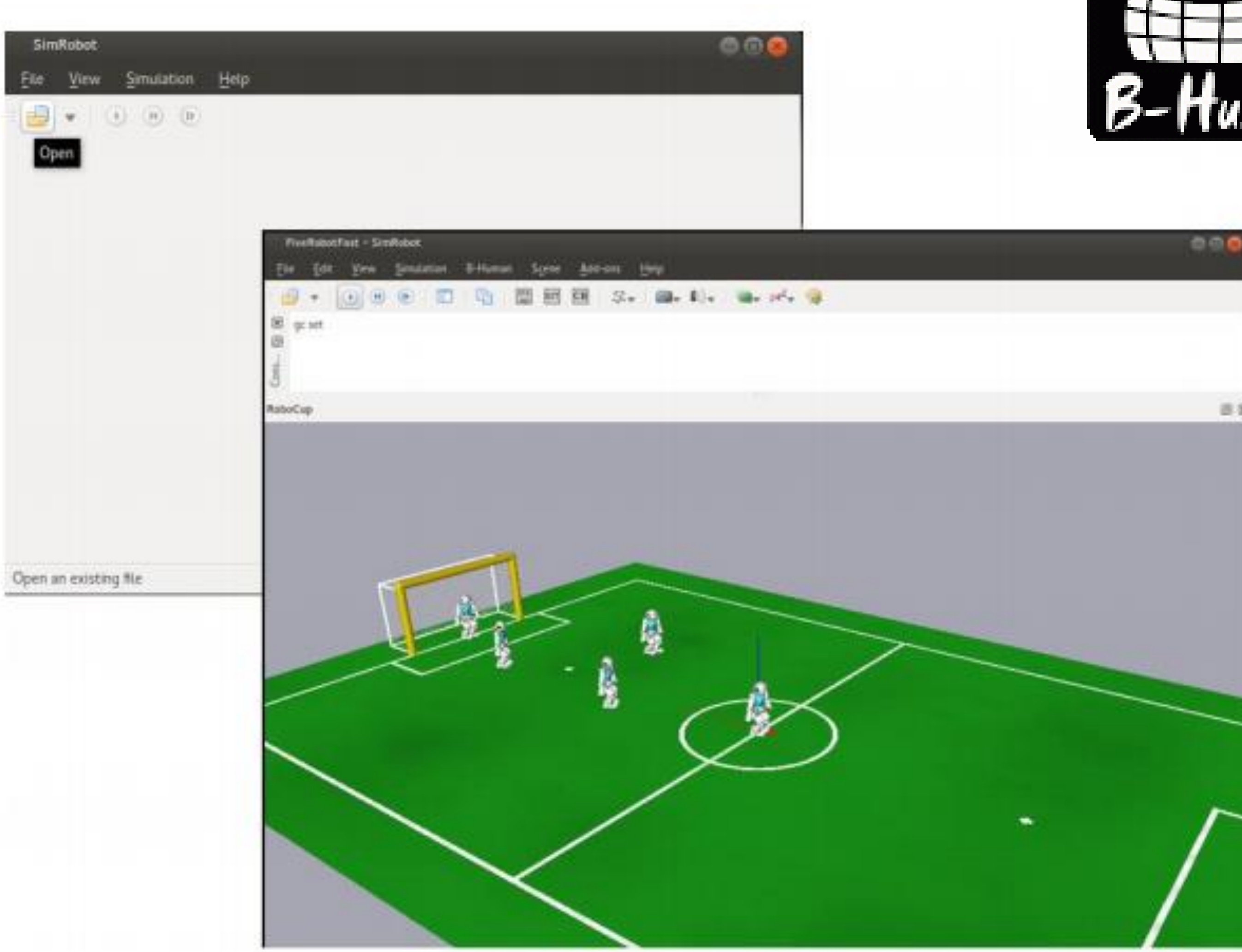
```
MODULE(B)
  REQUIRES(Foo1)
  PROVIDES(Foo2)
END_MODULE
```

D can be executed before **B**.

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

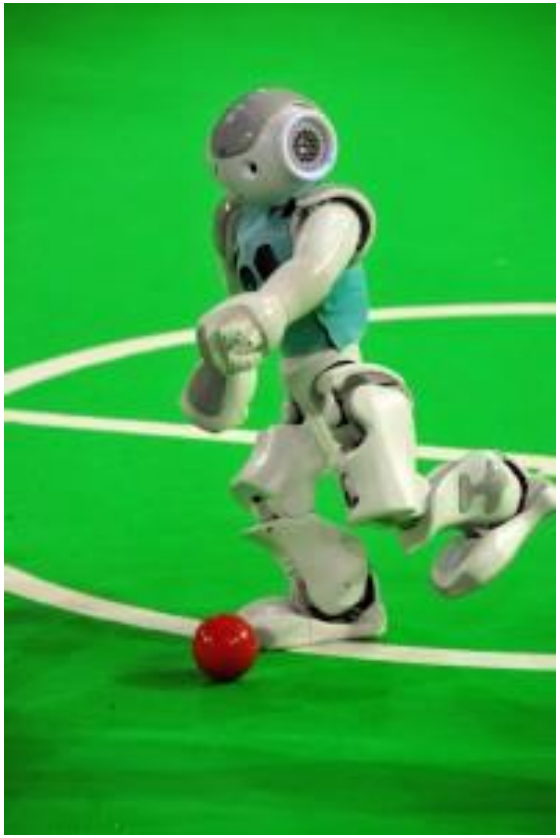
Tip: pay attention to the initialization of the *used* representations

SimRobot

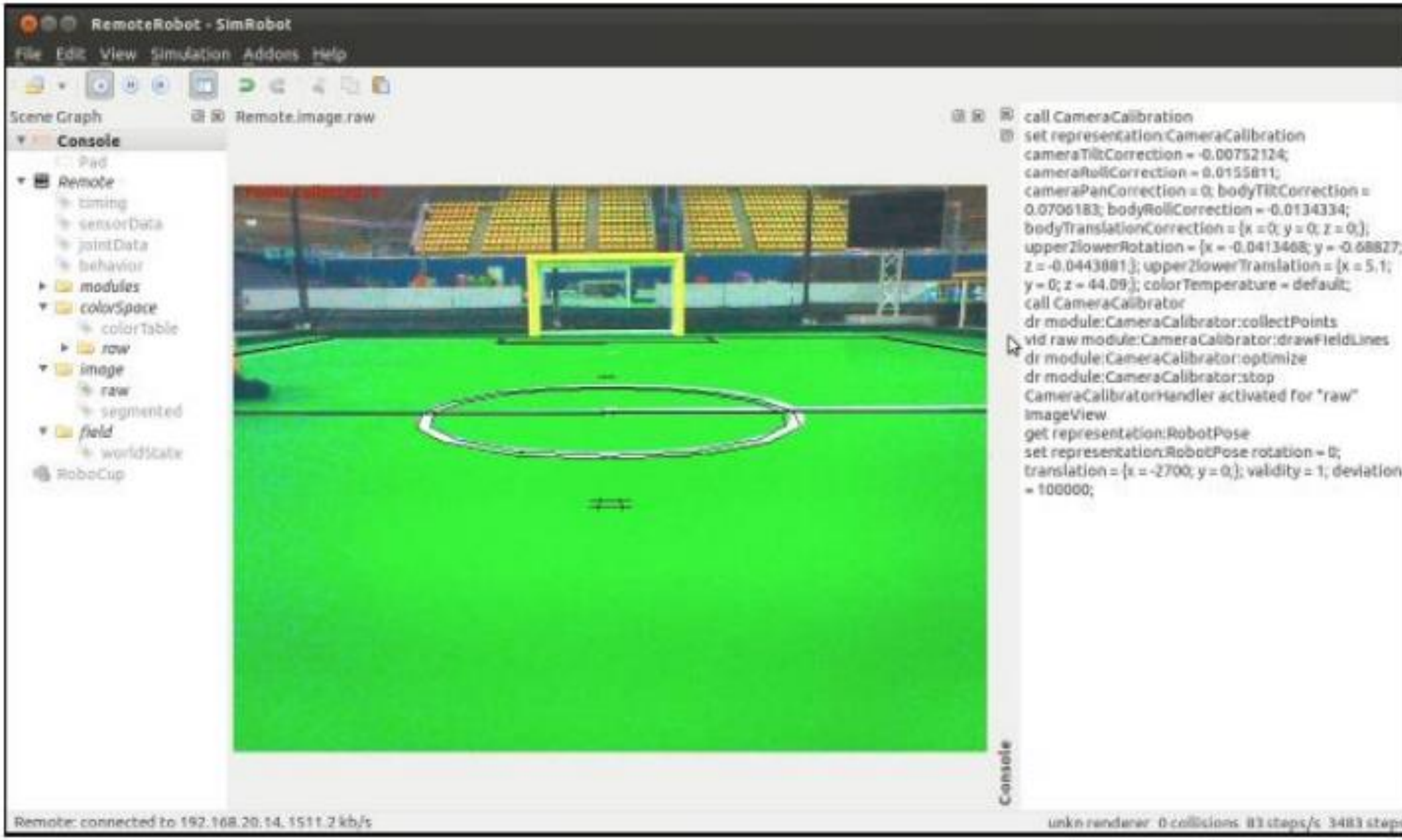

SimRobot
File View Simulation Help
Open
Open an existing file
RoboCup

SimRobot

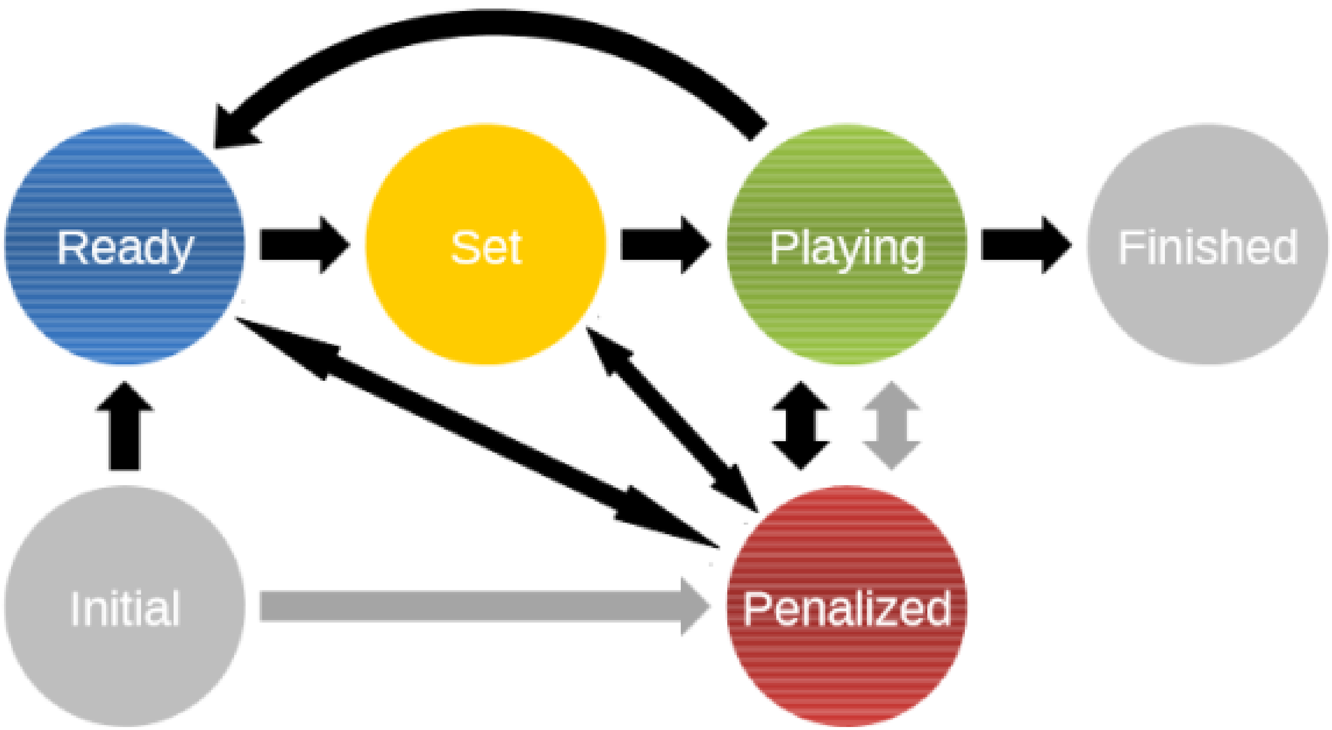
- ✓ Simulate the code;
- ✓ Connect the robot;
- ✓ Calibrate the color table;
- ✓ Calibrate the camera parameters;
- ✓ Calibrate sensors;



SimRobot


RemoteRobot - SimRobot
File Edit View Simulation Addons Help
Scene Graph
Console
Remote
modules
colorSpace
raw
image
raw
field
Remote.image.raw
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

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;

…

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 behavior as a **hierarchy of state machines**
- CABSL solely consists of C++ preprocessor macros and can be compiled with any standard 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

Used to describe behaviors for autonomous robots or NPCs in computer games

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.

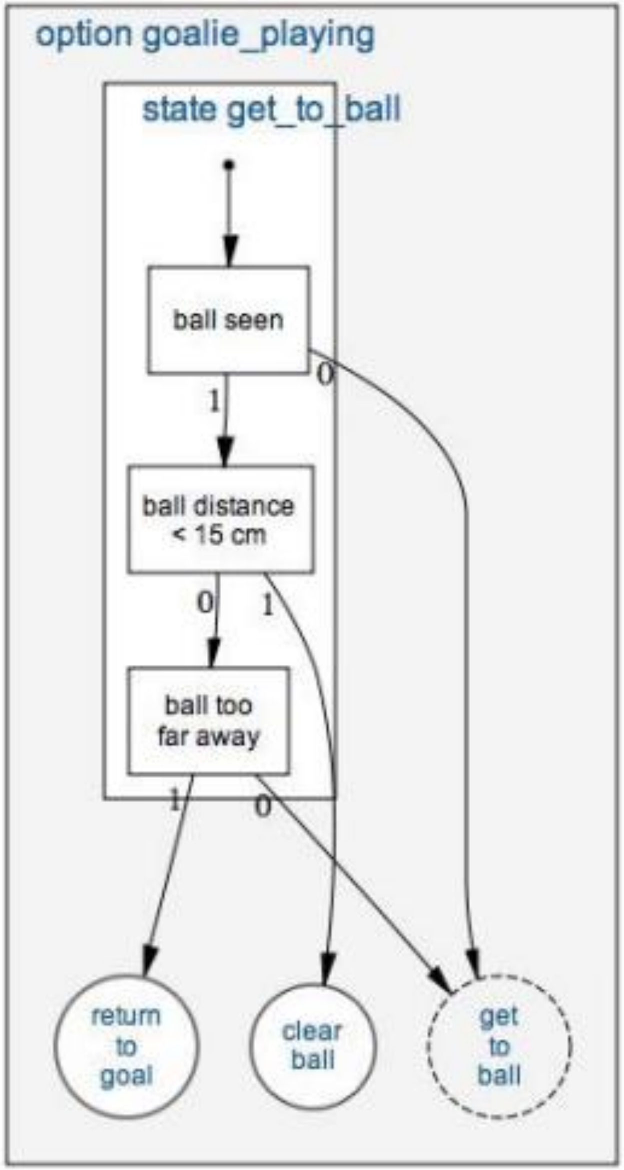
Tip: Deeply debug the inner state machine in order to avoid loops.

CABSL: Options

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

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

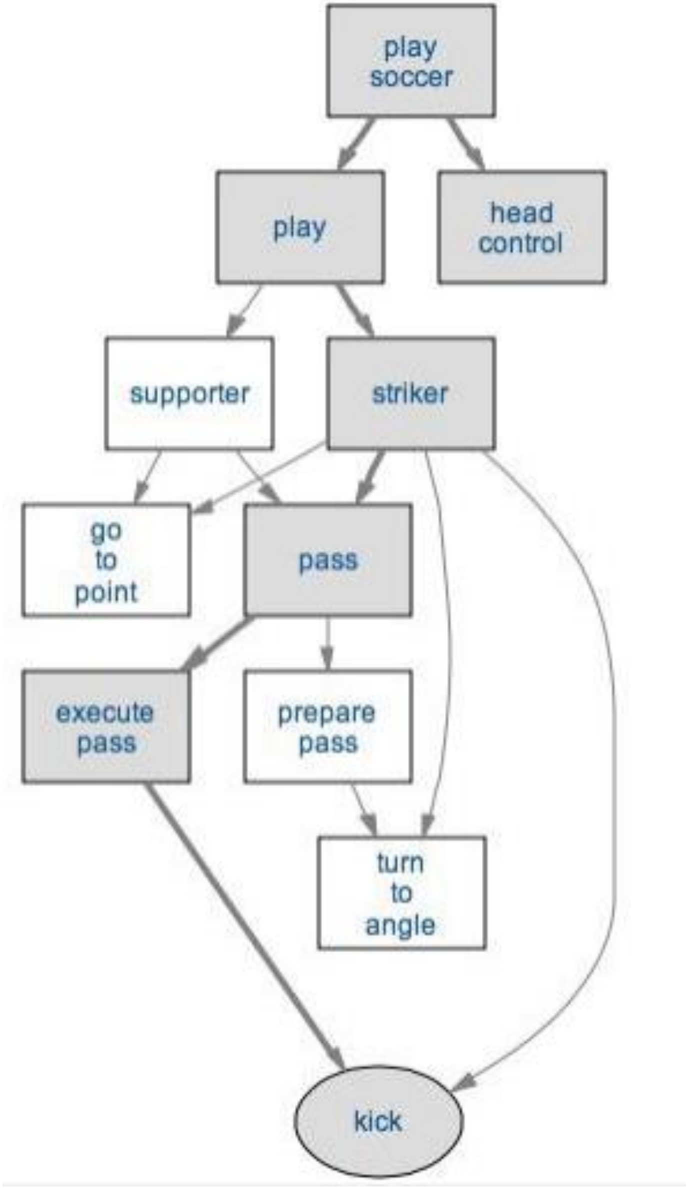
Tip: take into account how long the state has been active



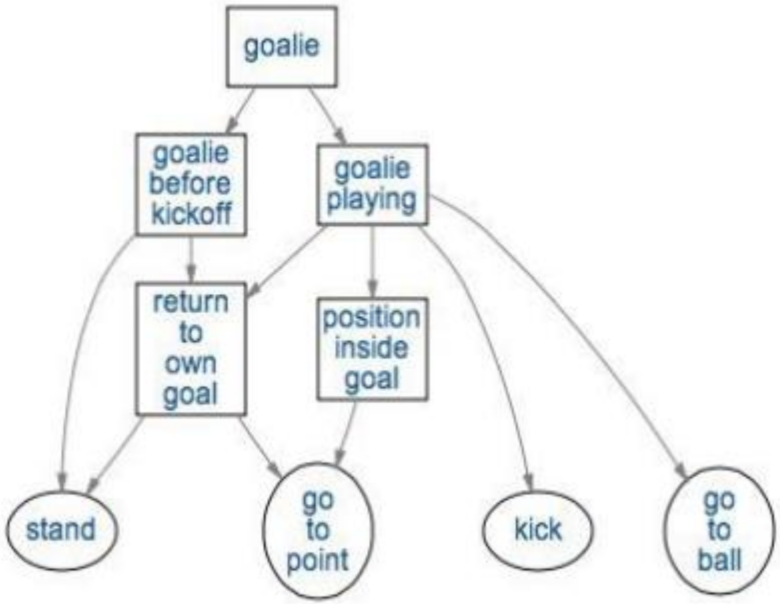
CABSL: Options

Options are activated at a specific timefrom a rooted tree

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



CABSL: Options



Pseudo-code:

Foreach iteration
{
the execution **starts** from the root and goes top-down through the option graph;
do
{
if the transition is within the current node **continues** 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)

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

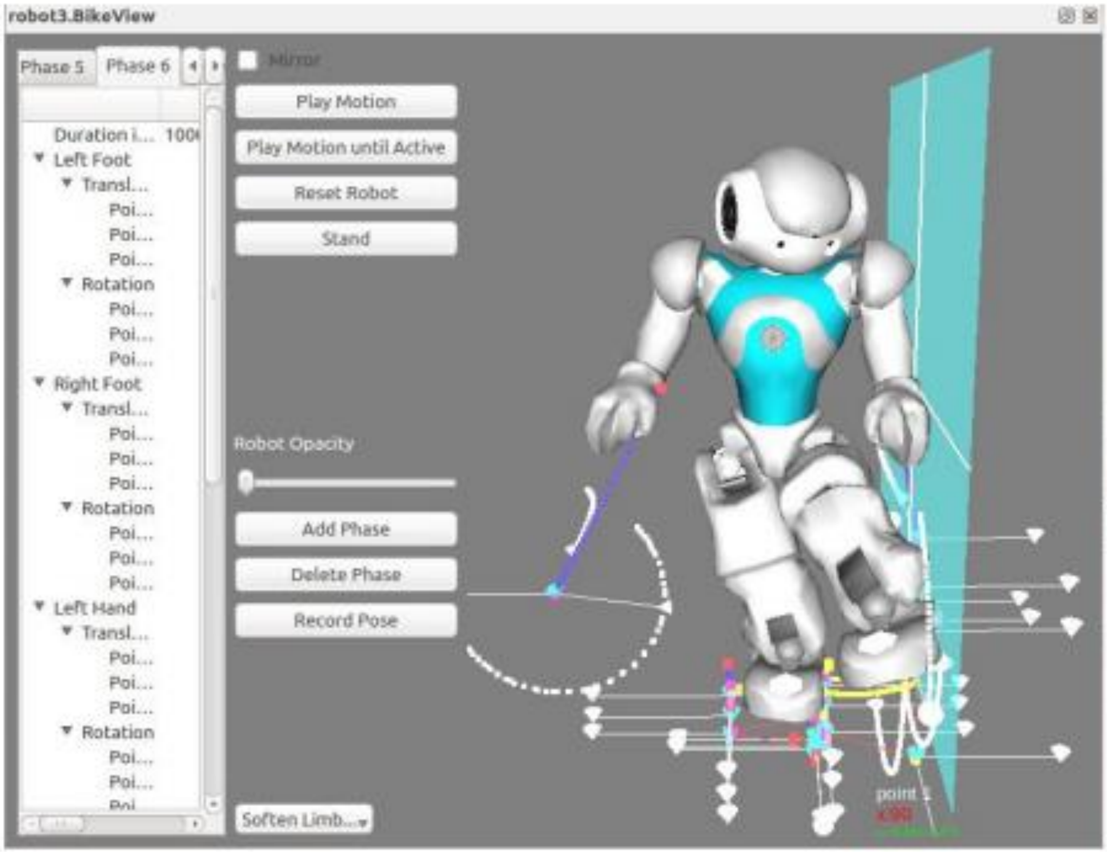
```
option(OptionWithParameters, int i, bool b, int j = 0)
{
  initial_state(firstState)
  {
    action
    {
      providedRepresentation.intValue = b ? i : j;
    }
  }
}
```

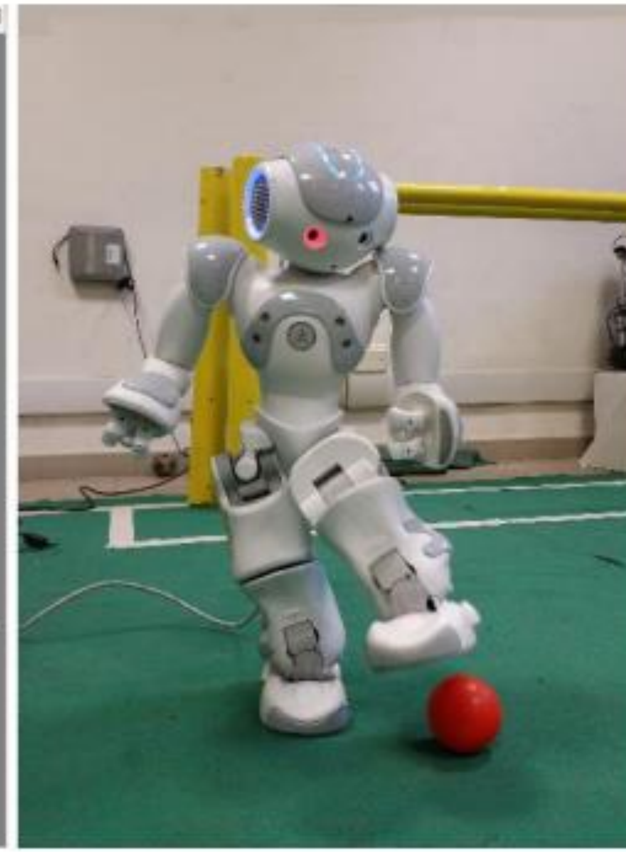
Arguments can generalize the options.

CABSL: Options

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

Motion interface: Bike scene





Ball recognition and evaluation

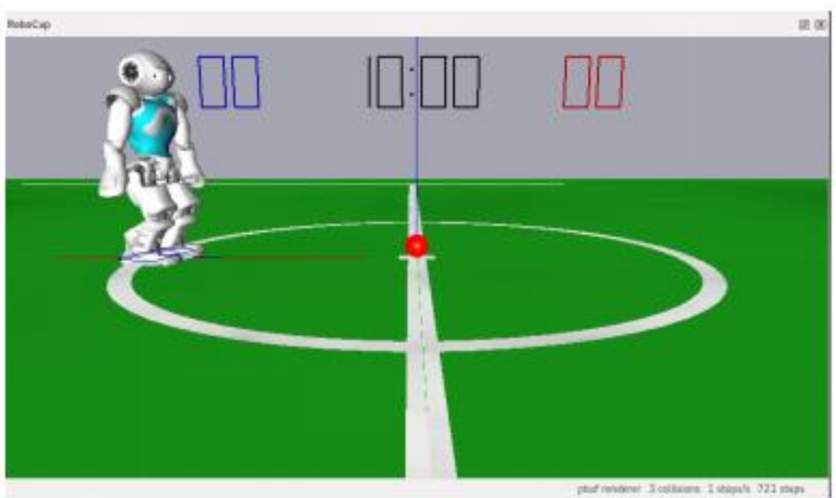
BallPercept.h

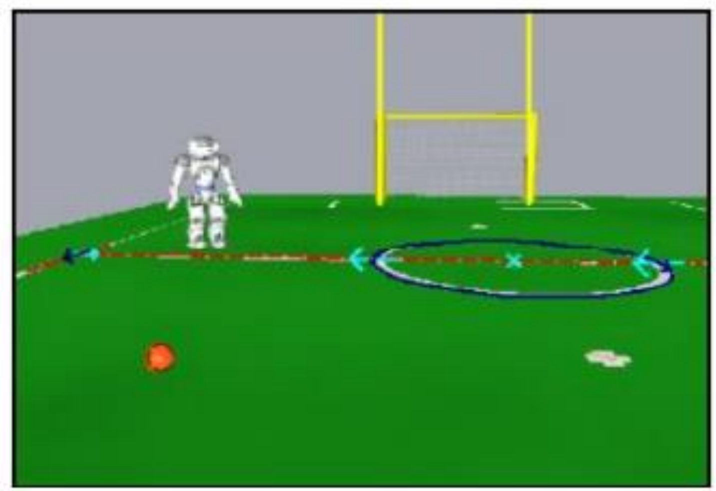
- USES BallModel
- PROVIDES BallPercept

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/masHw_18-19/**

1.A Make an account on github.com, send an email to **bipeds-spqr@googlegroups.com** with your github username ("[labnao] 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;

SPQR Code: Hands In

2.B Save images acquired from the robot camera;

2.C Detect the edges contained in the pictures using OpenCV (you can do this offline or within the bhuman module);

3.A Write a behavior that makes the robot “WalkTo” the ball;

3.B Extend the previous behavior and make the robot walk around the ball;

4.A Write a striker behavior that makes the robot kicking the ball towards its own goal;

4.B Test everything simulating two robots (striker and goalie).

4.C Write the free-kick behavior for the goalie and the striker and one(or two) defender(s) robot

SPQR Code: Hands In

```
<Compound name="robots">
  <Body ref="Nao" name="robot2">
    <Translation x="-2.2" z="320mm"/>
       <Set name="NaoColor" value="red"/>
    <Rotation z="180degree"/>
  </Body>
</Compound>
```

```
<Simulation>

  <Include href="Includes/NaoV4H21.rsi2"/>
  <Include href="Includes/Ball2016SPL.rsi2"/>
  <Include href="Includes/Field2015SPL.rsi2"/>

  <Scene name="RoboCup" controller="SimulatedNao"
stepLength="0.01" color="rgb(65%, 65%, 70%)" ERP="0.8"
CFM="0.001" contactSoftERP="0.2" contactSoftCFM="0.005">
    <!-- <QuickSolver iterations="100" skip="2"/>  -->
    <Light z="9m" ambientColor="rgb(50%, 50%, 50%)"/>

    <Compound name="teamcolors">
        <Appearance name="red"/>
        <Appearance name="blue"/>
    </Compound>

    <Compound name="robots">
      <Body ref="Nao" name="robot2">
        <Translation x="-2.2" z="320mm"/>
           <Set name="NaoColor" value="red"/>
        <Rotation z="180degree"/>
      </Body>
    </Compound>

     <Compound name="extras">
      <Body ref="NaoDummy" name="robot7">
        <Translation x="-4.3" y = "0.4" z="320mm"/>
     <Set name="NaoColor" value="blue"/>
      </Body>
    </Compound>


    <Compound name="balls">
      <Body ref="ball">
        <Translation x="-3.2" z="1m"/>
      </Body>
    </Compound>

    <Compound name="field">
      <Compound ref="field"/>
    </Compound>
  </Scene>
</Simulation>
```

SPQR Code: Hands In

.con

```
call Includes/Normal

# all views are defined in another script
call Includes/Views

dr debugDrawing3d:representation:RobotPose
```

.con Fast

```
call Includes/Fast

# field views
vf worldState
vfd worldState fieldLines
vfd worldState fieldPolygons
vfd worldState representation:RobotPose

# views relative to robot
vfd worldState origin:RobotPose
vfd worldState representation:BallModel
vfd worldState representation:MotionRequest
vfd worldState representation:ObstacleModel:rectangle
```

SPQR Code: Hands In