Ovládanie simulovaného humanoidného robota iCubSim pomocou architektúry Agent-Space

Andrej Lúčny – Matúš Kopernický

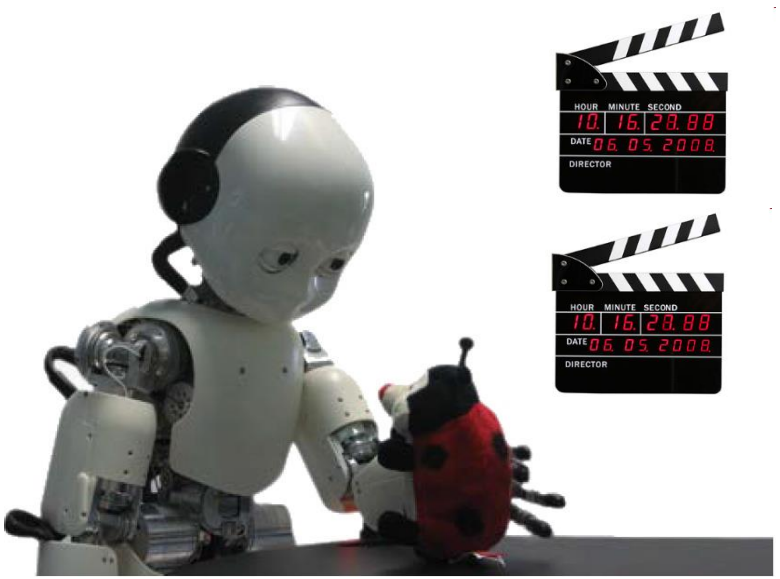
Katedra aplikovanej informatiky FMFI UK

lucny@fmph.uniba.sk

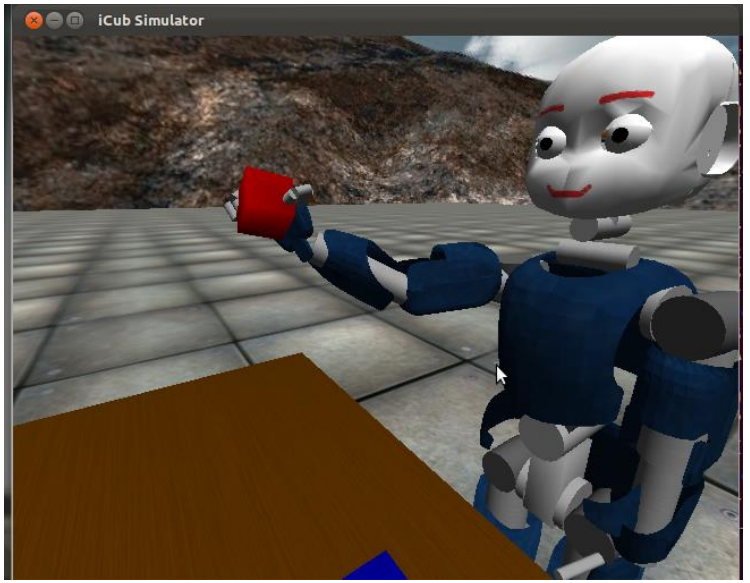
http://dai.fmph.uniba.sk/w/Andrej_Lucny

iCub a iCubSim

- EU IP RoboCub & NoE EuCognition I-III
- Cieľ: overiteľnosť výsledkov výskumu na humanoidných robotoch



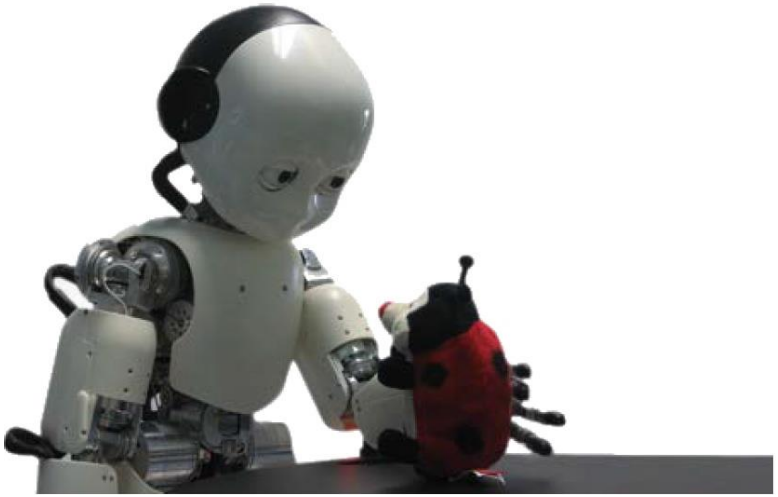
Jager - Meijering 2015



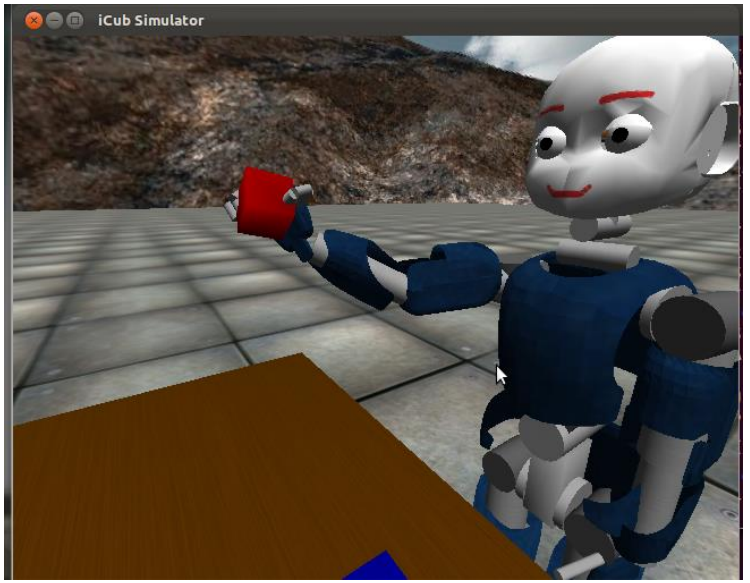
Zdechovan, 2012

iCub a iCubSim

- EU IP RoboCub & NoE EuCognition I-III
- Cieľ:



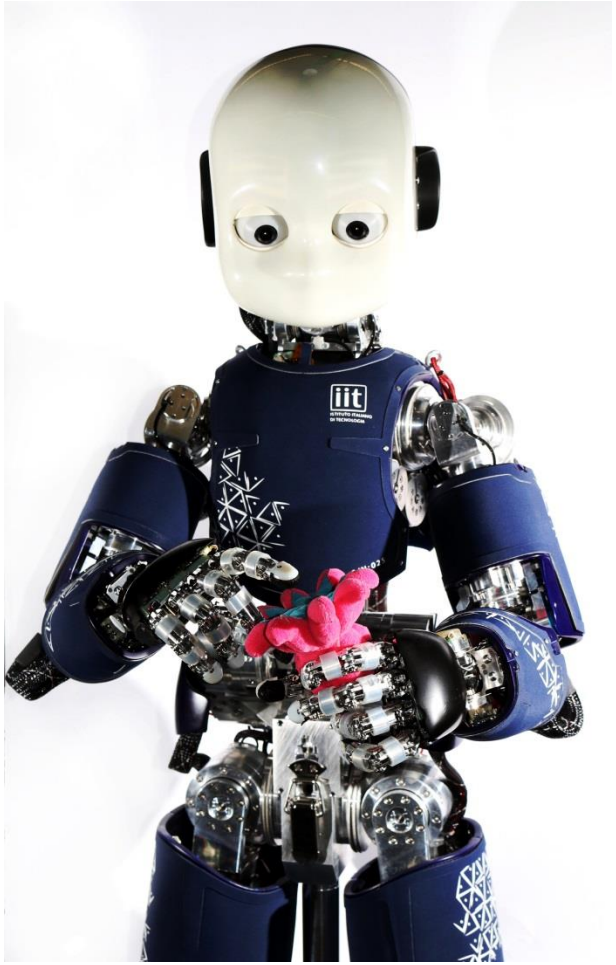
Jager - Meijering 2015



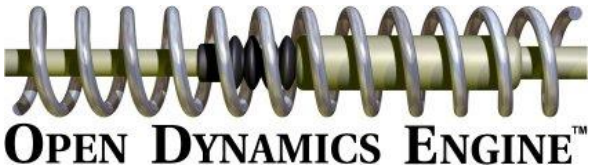
Zdechovan, 2012

iCub a iCubSim

- Humanoidná robotická platforma s otvoreným zdrojovým kódom
- 53 stupňov voľnosti
- dotykové/tlakové senzory
- kamera z ľavého a pravého oka



iCubSim



- Simulátor napísaný v C++ používajúci ODE a OpenGL, primárne na platforme Linux

  `eris.liralab.it/wiki/Simulator_README`

- Inštalácia na Ubuntu

```
# sudo sh -c 'echo "deb http://www.icub.org/ubuntu
trusty contrib/science" >
/etc/apt/sources.list.d/icub.list‘
sudo apt-get update
sudo apt-get install icub
```

- V `/usr/share/icub ... context/simconfig/iCubPartsActivation.ini`

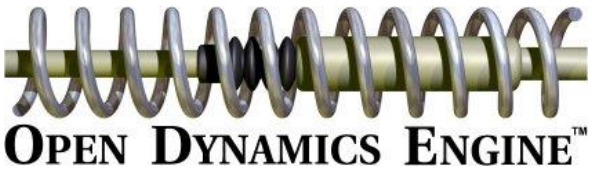
nastaviť `objects = on` a `ode_param` - timestamp - 35 (10)

iCubSim

- Vývoj svojej aplikácie
- Táto sa pripája na yarp server (TCP)
- Programovací jazyk GNU GCC
- IDE QtCreator

- Kompilácia: cmake

```
cmake ./
make
```



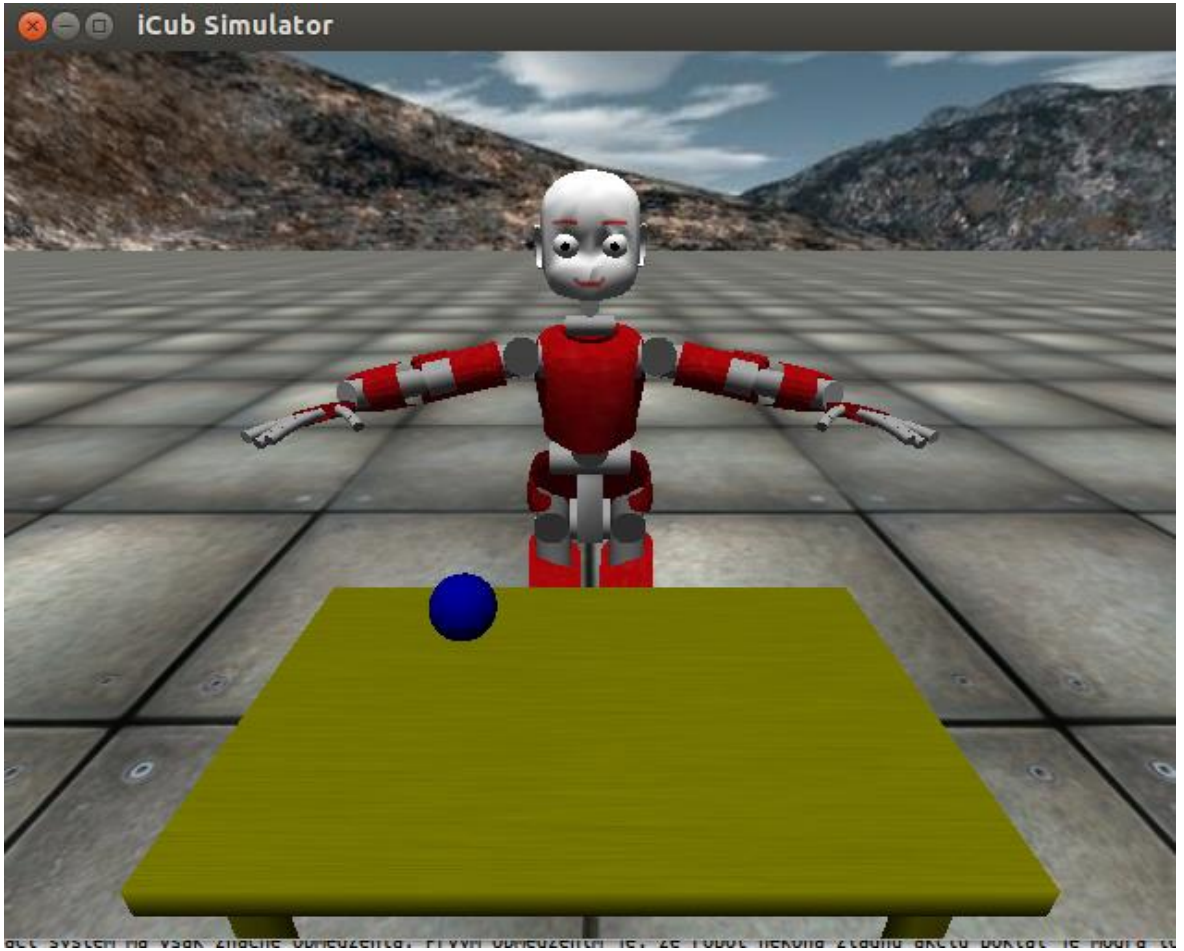
iCubSim

- Spustenie

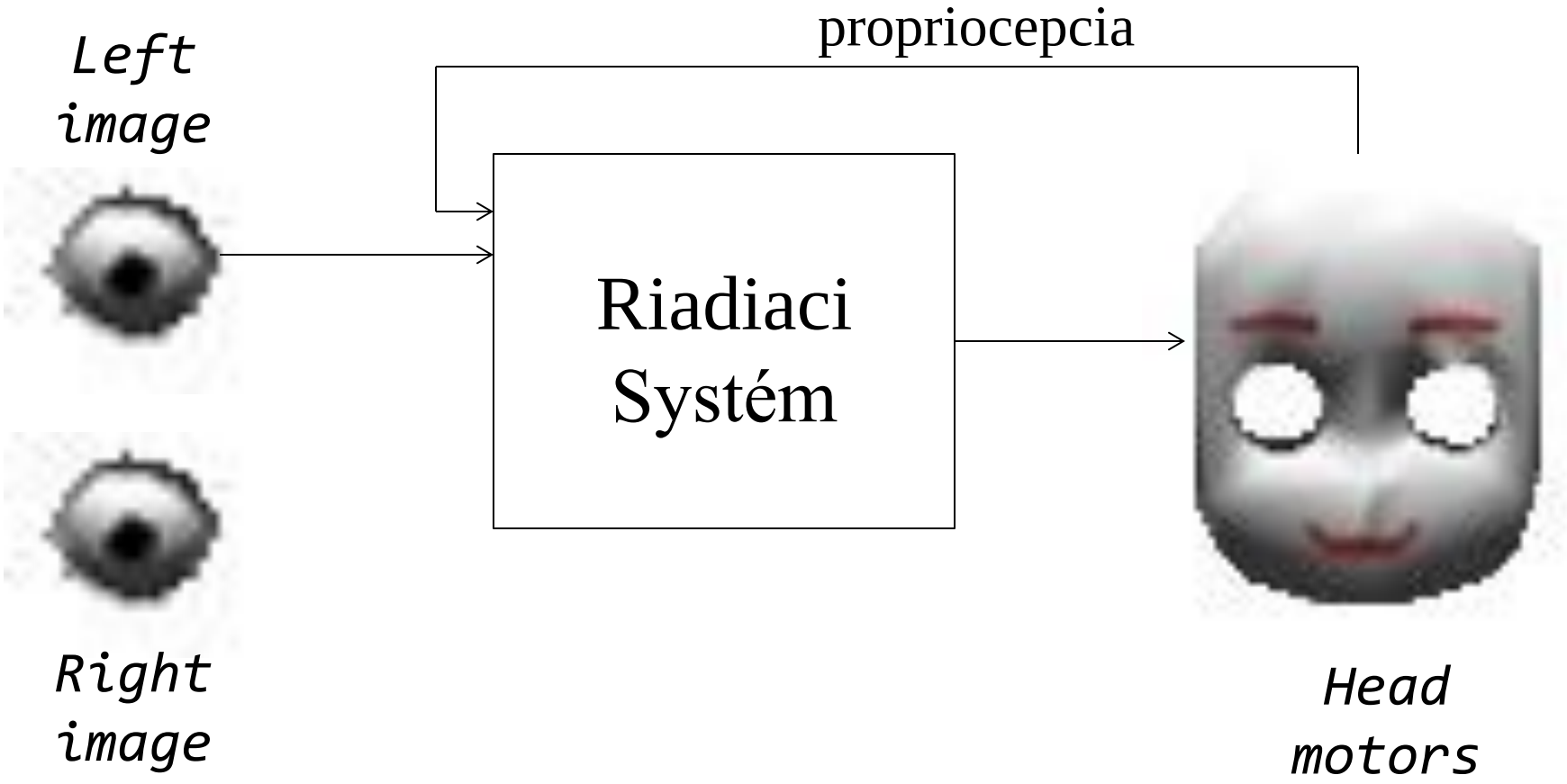
```
#gnome-terminal -e "yarpserver --write"
gnome-terminal -e "yarpserver"

gnome-terminal -e "iCub_SIM"
sleep 5
gnome-terminal -e "yarpview /camera"
sleep 1
gnome-terminal -e "yarp connect /icubSim/cam/left /camera"
sleep 4
gnome-terminal -e ".../icub_agentspace/demo/icub_agentspace"
```

Modelová úloha

- Hľadanie modrej loptičky v priestore.
- Upriamenie pozornosti.
- Udržanie loptičky v zornom poli



Modelová úloha



Práca so senzormi a motormi

```
deviceName = "/icubSim/cam/left";
portName = "/left/image/in";

imagePort = new  yarp::os::BufferedPort
                    <yarp::sig::ImageOf
                          <yarp::sig::PixelRgb> >();

imagePort->open(portName);
yarp::os::Network::connect(deviceName,portName);

yarp::sig::ImageOf<yarp::sig::PixelRgb> *image;
image = imagePort->read();
if (image != NULL) process(image);
```

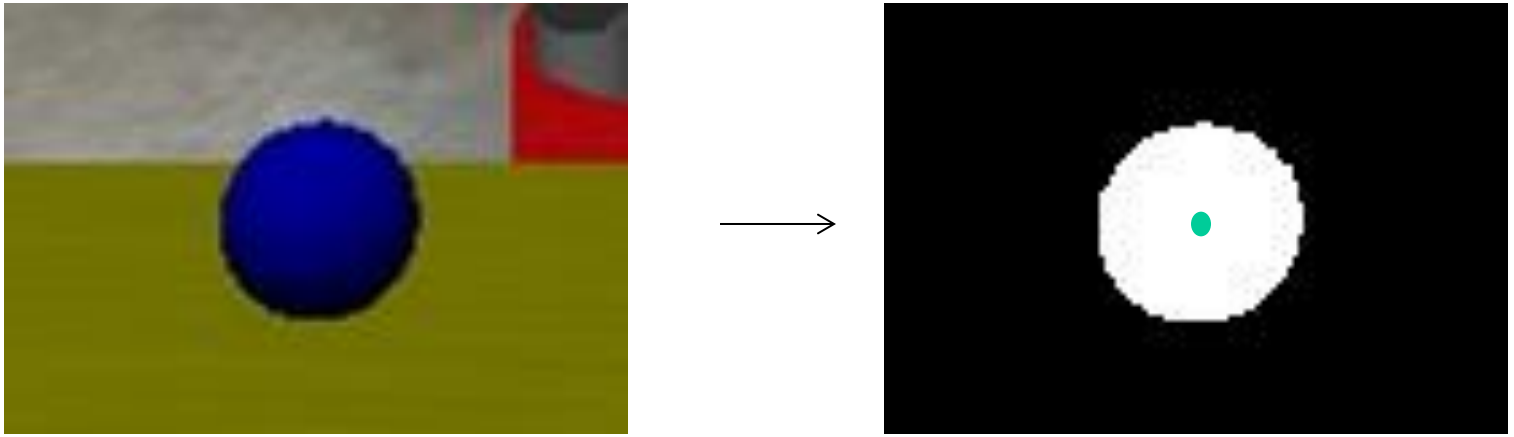
Riadiaci systém (klasický)

```
for (1.motor) {
     for (2.motor) {
          for (3.motor) {
               if (vidime(obraz()) {
                    exit;
               }
          }
     }
}
```

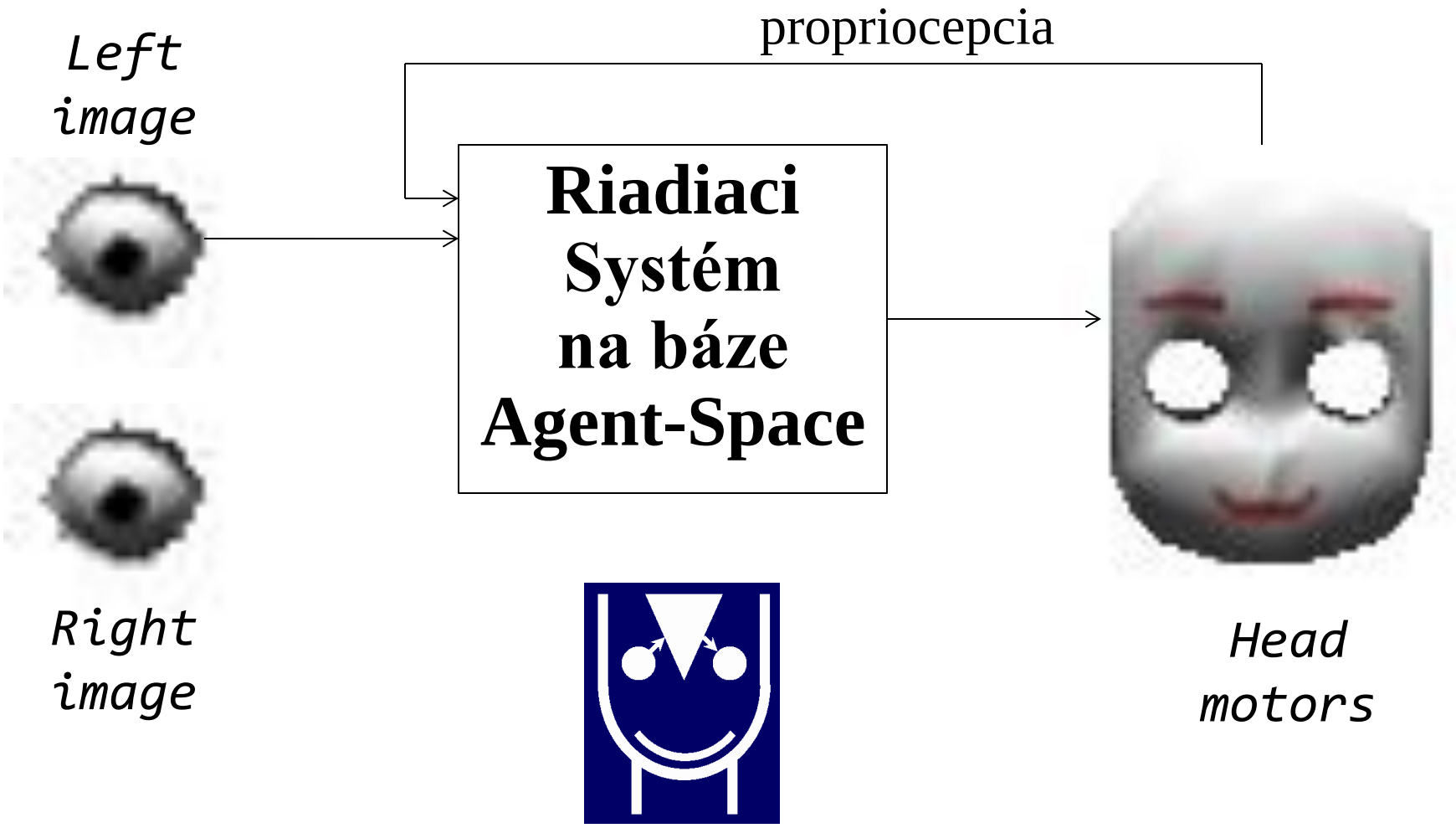
jedno vlákno

Rozpoznanie loptičky

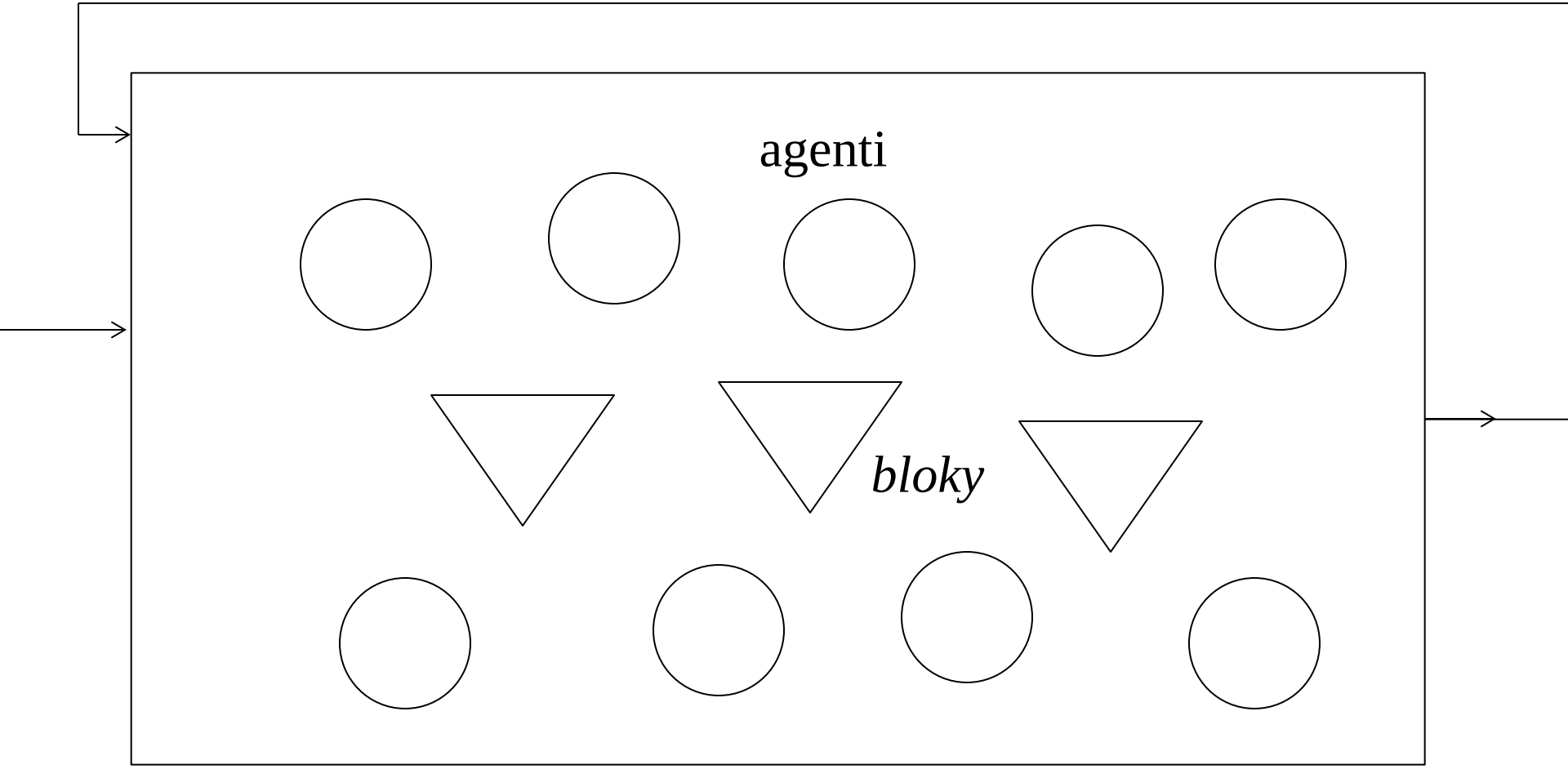
- Podľa farby: `malé R, malé G, veľké B`



Použitie architektúry Agent-Space

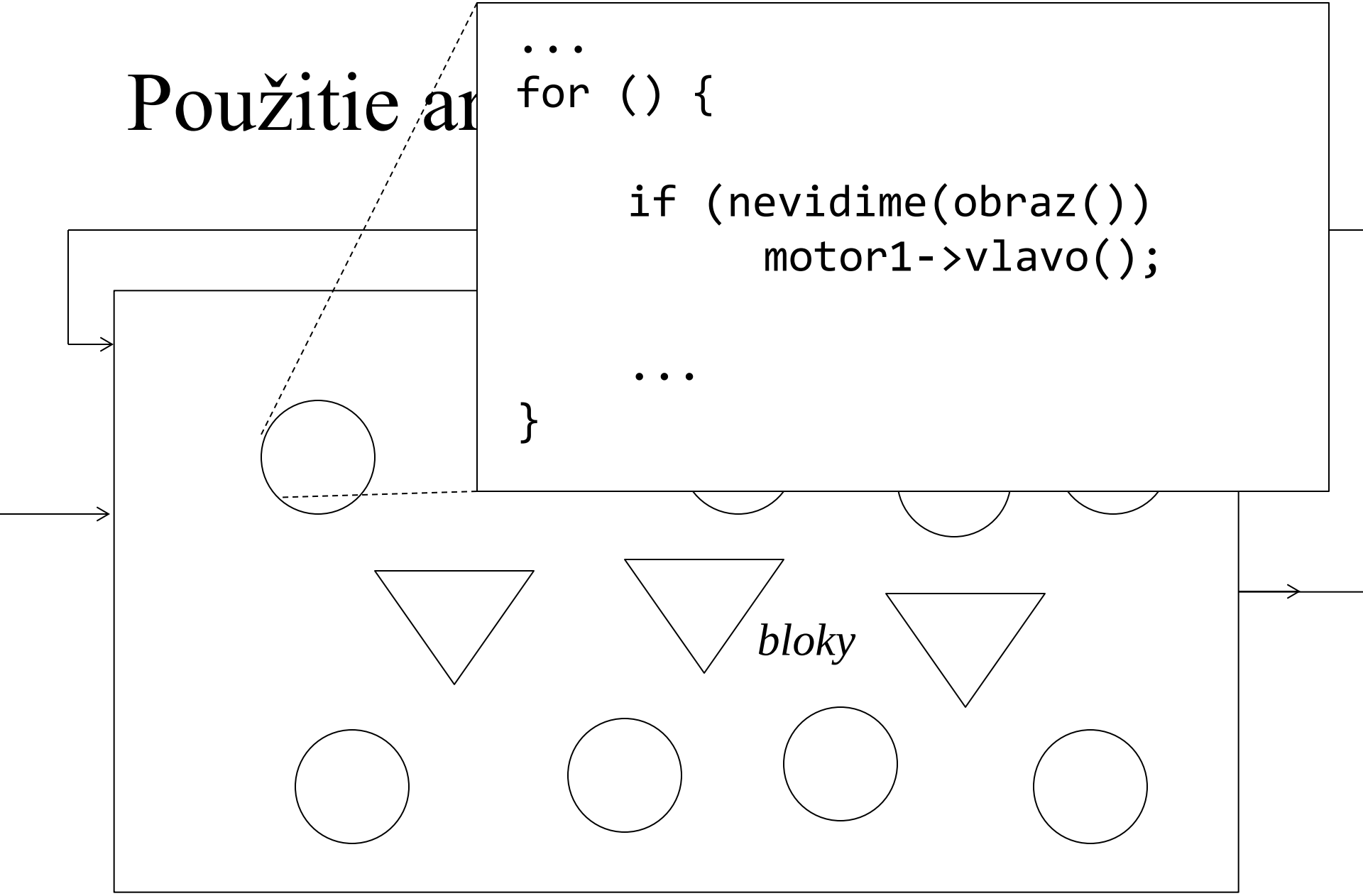


Použitie architektúry Agent-Space



Použitie ar

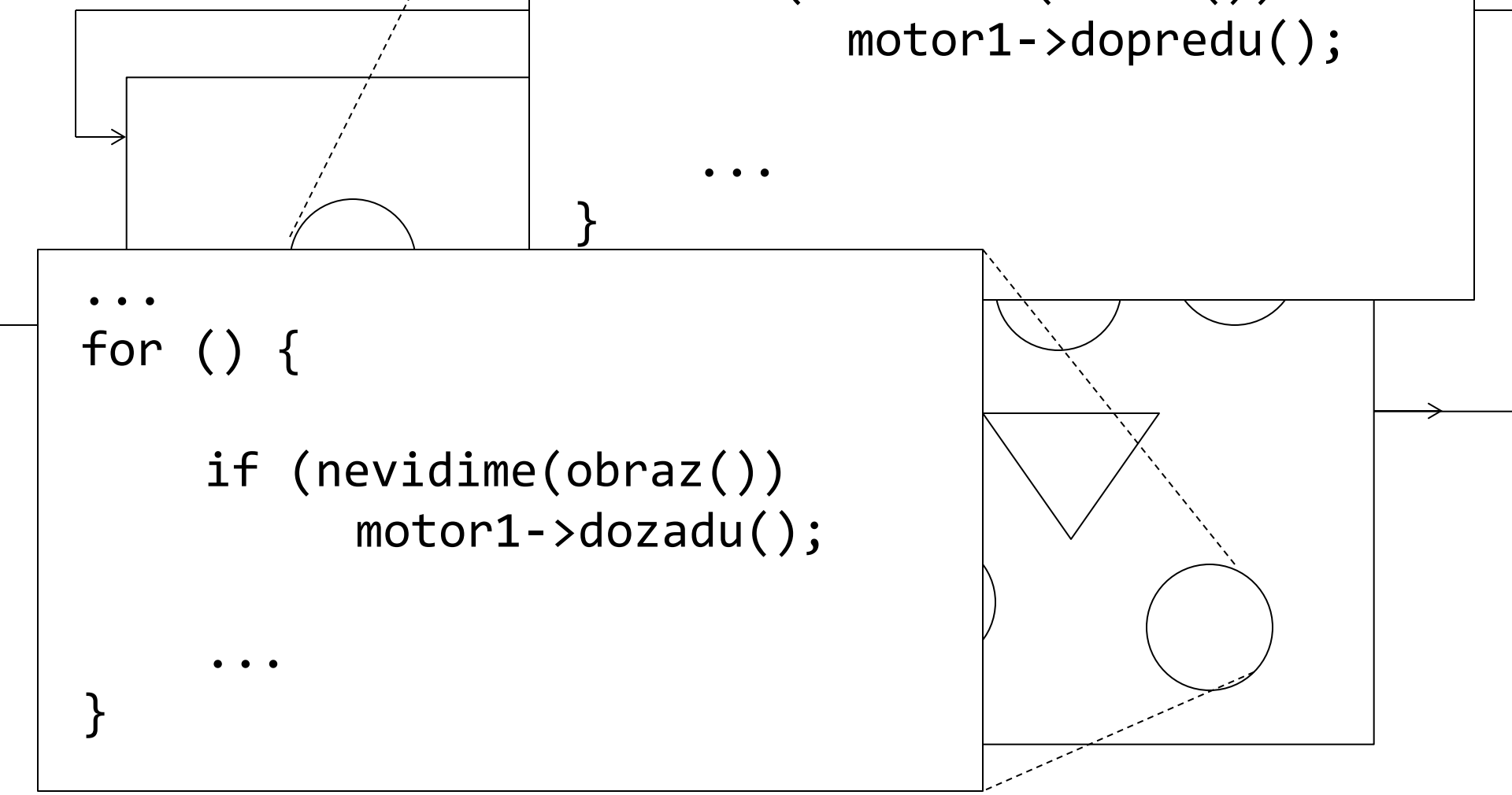
```
...
for () {

     if (nevidime(obraz())
          motor1->vlavo();

     ...
}
```



Použitie ar

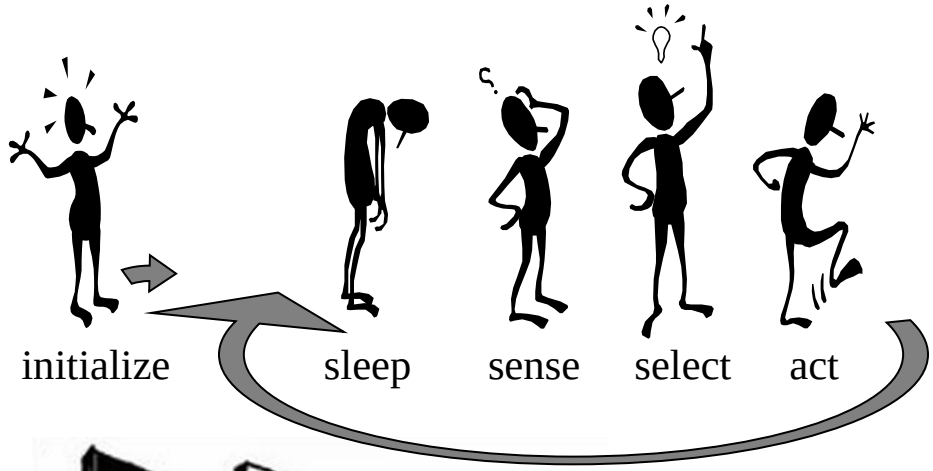
```
...
for () {

    if (nevidime(obraz())
        motor1->dopredu();

    ...
}
```

```
...
for () {

    if (nevidime(obraz())
        motor1->dozadu();

    ...
}
```

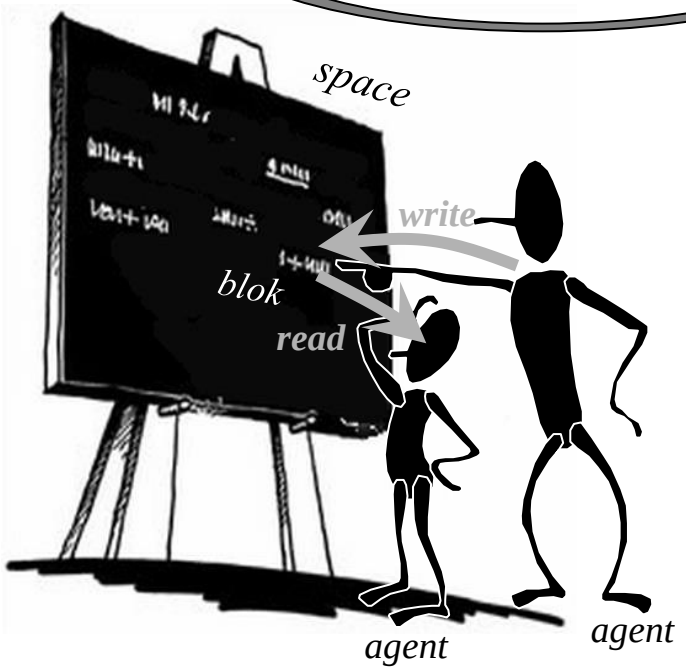


Architektúra Agent-Space

- Systém sa skladá z agentov

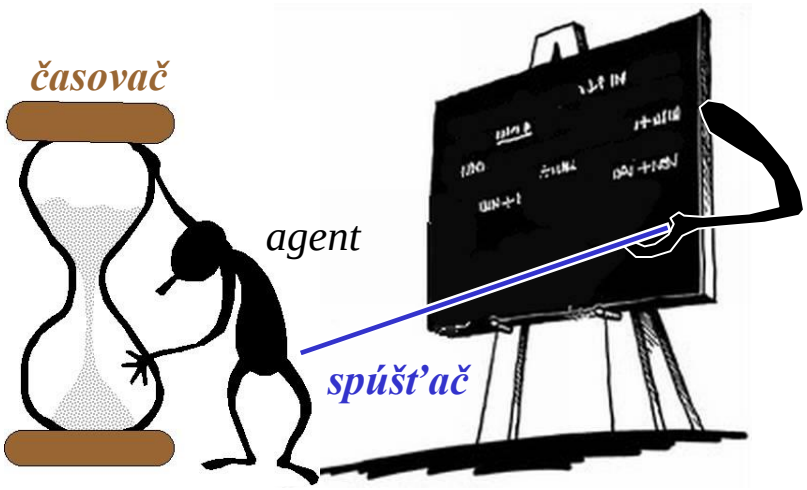


- Agenti medzi sebou komunikujú nepriamo cez Space (čiernu tabuľu)



Implementácia v C++

- Multivláknové prostredie pthreads
- Každý agent má vlastné vlákno
- Može volať metódy singletonu Space
- vlákno agenta je blokované na časovač alebo spúšťač

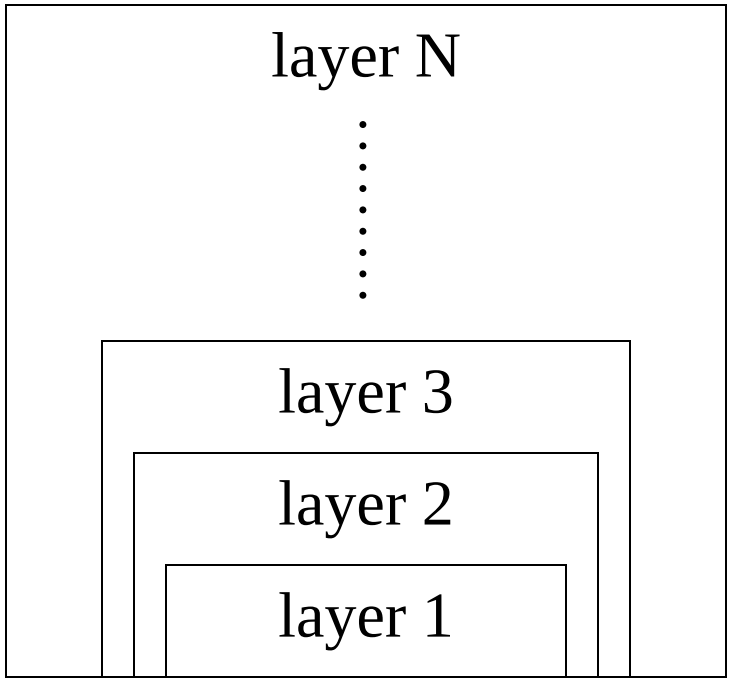


Príklad použitia

```
#include <iostream>
#include <conio.h>
#include "agentspace.h"
using namespace std;

class MyAgent1 : public Agent {
  private:
    int i;
  protected:

    void init (string args) {
      i = 0;
      timer_attach(1000,1000);
    }

    void sense_select_act (int pid) {
      i++;
      cout << "a := " << i << endl;
      space_write("a",i,1500);
    }

  public:
    MyAgent1 (string args) :
      Agent(args) {};
};
```

```
class MyAgent2 : public Agent {
  protected:

    void init (string args) {
    trigger_attach("*",TRIGGER_MATCHING);
    }

    void sense_select_act (int pid) {
      int a = space_read("a",0);
      cout << "a = " << it->value << endl;
    }

  public:
    MyAgent2 (string args) :
      Agent(args) {};
};

int main () {
        MyAgent1 a1("");
        MyAgent2 a2("");
        getch();
}
```

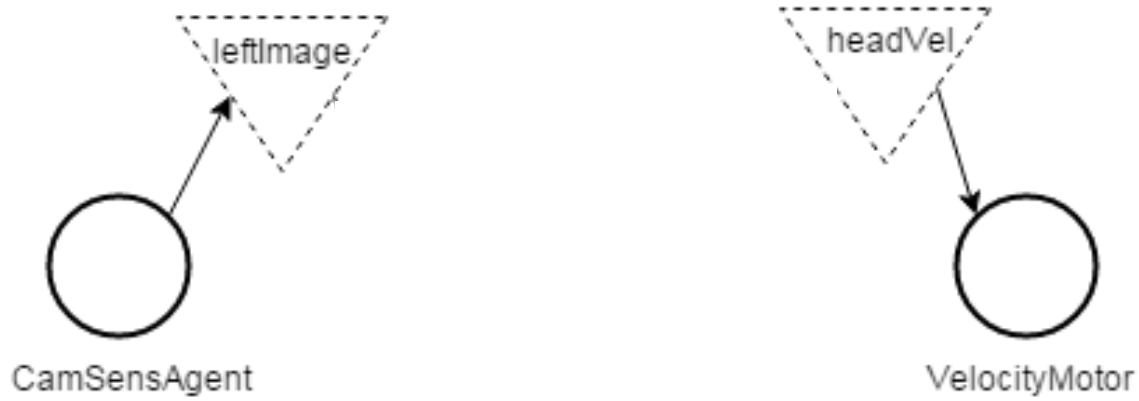
Vývojová stratégia: subsumpcia



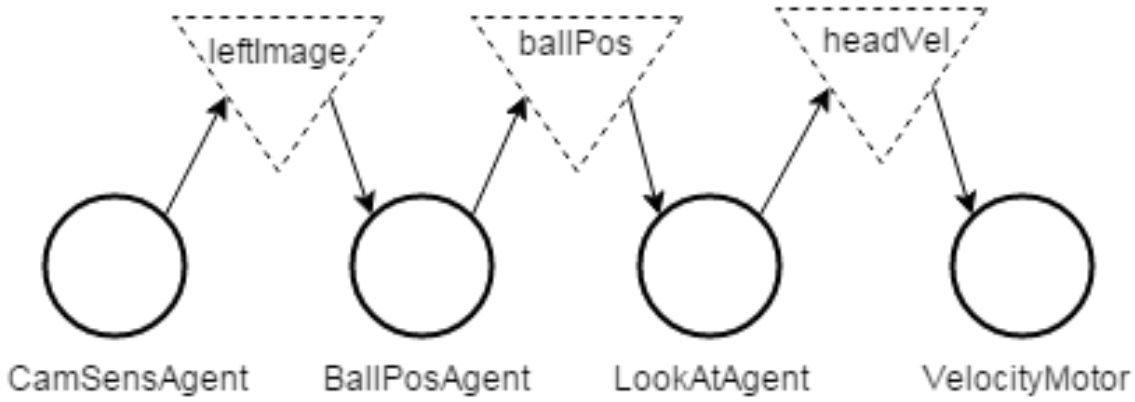
- „evolučne staršie“ vrstvy sa už nemenia
- „evolučne novšie“ vrstvy ovplyvňujú „evolučne staršie“ vrstvy

0. etapa vývoja – senzory a aktuátory

Prepojenie senzorov a aktuátorov s blokmi v Space



1. etapa vývoja - dátová výmena



1. etapa vývoja - CamSensAgent

```
void CamSensorAgent::init(string args) {
    SensorBlock defConf;
    defConf.agentName = "HeadAgent";
    defConf.deviceName = "/icubSim/cam/left";
    defConf.portName = "/left/image/in";
    defConf.outBlock = "leftImage";
    defConf.startTime = 1000;
    defConf.period = 250;
    conf = space_read(args, defConf);

    imagePort = new  yarp::os::BufferedPort<yarp::sig::ImageOf<yarp::sig::PixelRgb> >();
    imagePort->open(conf.portName);
    yarp::os::Network::connect(conf.deviceName,conf.portName);

    timer_attach(conf.startTime,conf.period);
}

void CamSensorAgent::sense_select_act(int pid) {
    yarp::sig::ImageOf<yarp::sig::PixelRgb> *image = imagePort->read();  // read an image
    if (image!=NULL) {
        space_write(conf.outBlock, image);
    } else {
        printf("No image\n");
    }
}
```

1. etapa vývoja-BallPosAgent

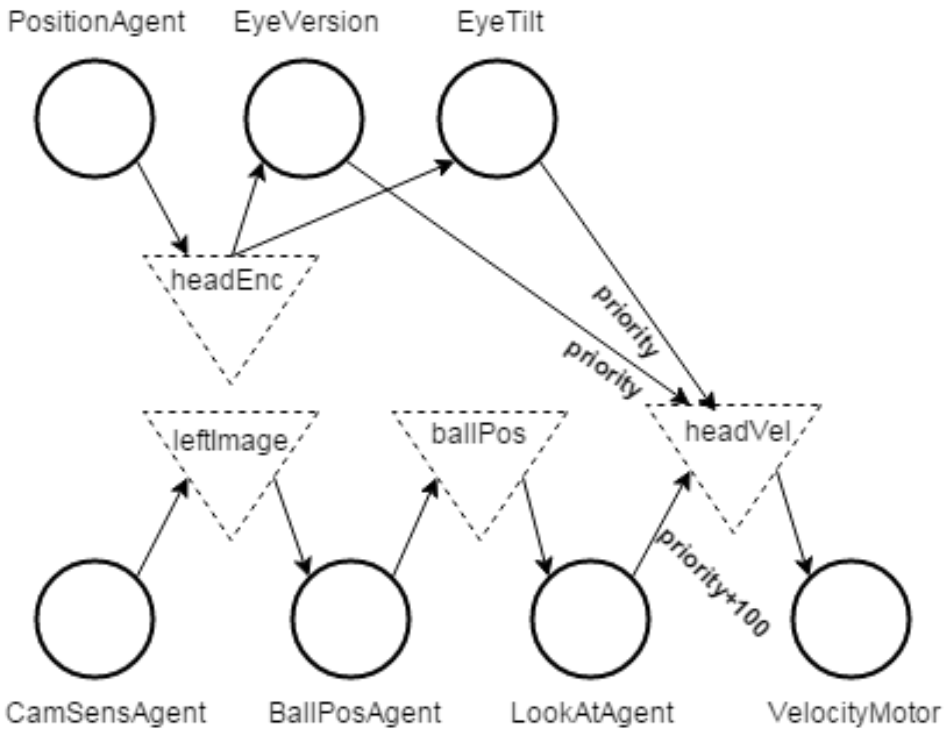
```
void BallPosAgent::init(string args) {
    ...
    conf = space_read(args, defConf);
    trigger_attach(conf.inBlock,TRIGGER_MATCHING);
}
void BallPosAgent::sense_select_act(int pid) {
   ...
    yarp::sig::ImageOf<yarp::sig::PixelRgb> *image = space_read(conf.inBlock, defimage);
    if (image != NULL) {
        width = image->width();
        height = image->height();
        for (int x = 0; x < width; x++) {
            for (int y = 0; y < height; y++) {
                yarp::sig::PixelRgb& pixel = image->pixel(x,y);
                if (pixel.b > pixel.r*1.2+10 && pixel.b > pixel.g*1.2+10) {
                    xMean += x; yMean += y; count++;
                }
            }
        }
        if (count > 0) {
            xMean /= count; yMean /= count;
        }
        target[0] = xMean; target[1] = yMean;  target[2] = 1;
    }
    space_write(conf.outBlock, target);
}
```

1. etapa vývoja-VelocityMotorAgent

```
void VelocityMotorAgent::init(string args) {
    ...
    conf = space_read(args, defConf);

    yarp::os::Property options;
    options.put("device", conf.yarpDevice);
    options.put("local", conf.yarpLocal);
    options.put("remote", conf.yarpRemote);
    ...
    trigger_attach(conf.velocityBlock,TRIGGER_MATCHING);
}

void VelocityMotorAgent::sense_select_act(int pid) {
    commands = space_read(conf.velocityBlock, def_commands);
    vel->velocityMove(commands.data());
}
```

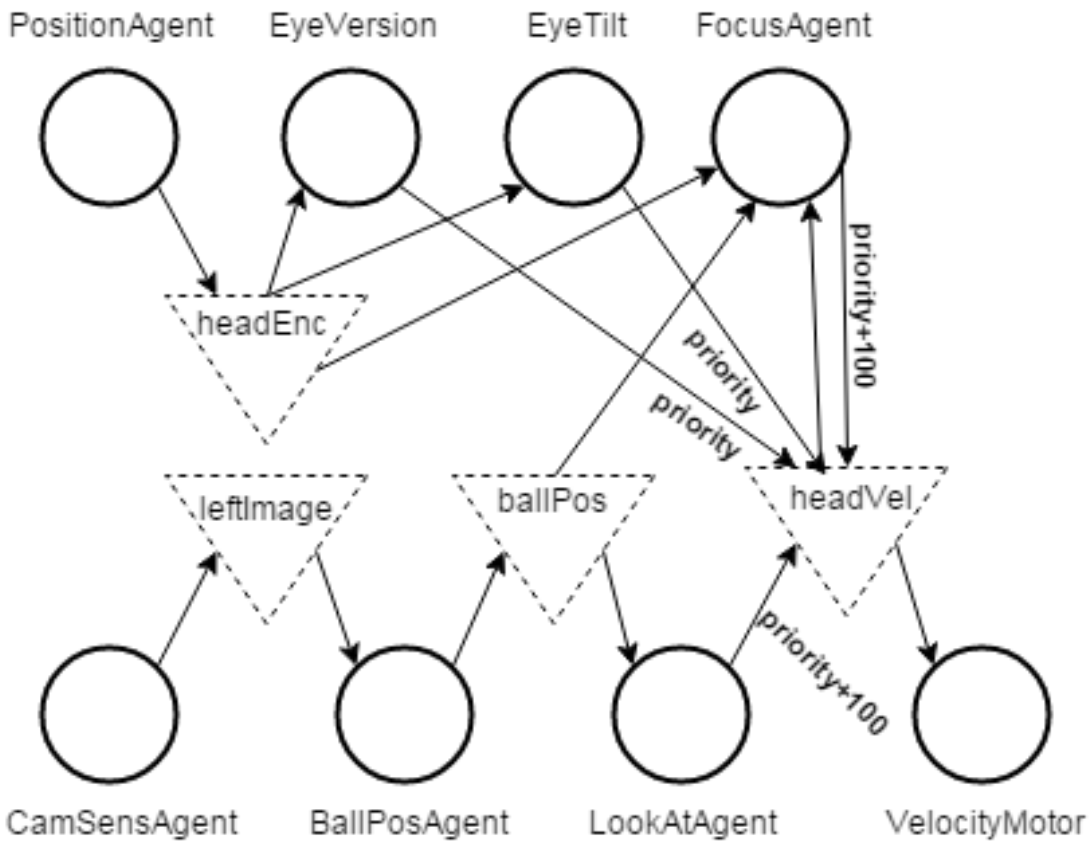
2. etapa vývoja

- Hľadanie loptičky
  - Rozširovanie zorného pola
    - PositionAgent, EyeVersionAgent, EyeTiltAgent



3. etapa vývoja

- Upriamenie pohľadu na loptičku - FocusAgent



Záver

- Rozbehali sme simulátor iCubSim
- Do simulátora iCubSim sme implementovali architektúru Agent-Space
- Prepojili sme senzory a aktuátory iCub-u s architektúrou
- Vyvinuli sme pomocou nej riadenie hlavy iCub-u tak, aby sa pozerala za modrou loptičkou na stole (čo najprirodzenejším spôsobom)
- Ďalšie info: www.agentspace.org/icub

Ďakujeme za pozornosť

Ovládanie simulovaného humanoidného robota iCubSim pomocou architektúry Agent-Space

Andrej Lúčny – Matúš Kopernický

Katedra aplikovanej informatiky FMFI UK

lucny@fmph.uniba.sk

http://dai.fmph.uniba.sk/w/Andrej_Lucny