A Multi-Robot System for Anticipatory Experiments[⋆]

Birger Johansson and Christian Balkenius

Lunds University Cognitive Science, Kungshuset Lundagård
222 22 Lund, Sweden
{birger.johansson, christian.balkenius}@lucs.lu.se

Abstract. During the years 2005 and 2006 a robot system has been developed at Lund University Cognitive Science to run navigation experiments in a dynamic environment. The main aim of the system is to study anticipation during navigation, model building, color tracking, navigation algorithms, social robotics and children's games. This technical report describes the hardware and software used in the system. The system consists of two small two-wheeled robots, which are tracked using color markers using an overhead camera. The robots are controlled by a computer that handles tracking, navigation, motor control and the experiment control for each robot. Two setups are presented with example of the implementation of the integrated parts which can be used to run the system either in simulation or with the real robots. Currently, only two robots are supported, but in the final phase the system will handle six robots.

1 Introduction

During the last two years we have been developing a multi-robot system with the primarily aim of studying anticipation during navigation. We believe that we have found a suitable domain for studying many of the behaviors that we are interested in including attention, social interaction, context, navigation algorithms etc. although the main focus is anticipation in navigation.

By using robots, algorithms and models of cognitive processing can be evaluated in a real world environment. Today, robots are used to interact with children [4], as rescue robots [3], for entertainment [16] and in many other fields. Robots were previously expensive to build and demanded large amounts of expertise to get all its components to work together. With today's low cost and more flexible electronics, a broader community sees the benefits and both companies and universities have focused more on robot research. Although many impressive robots like ASIMO [13], DB [11] and COG [2] require large funding, is is also possible to build robots using minimal resources.

Which robot and setup one should chose depends much of which type of experiment or model that one are aiming for. One of the most common robots

⋆ Johansson, B. & Balkenius, C. *A Multi-Robot System for Anticipatory Experiments*, LUCS Minor, 11.

are round robots that have differential steering. This type of robots are mainly used in studying navigation or group behavior such as flocking [12]. Often the robots have some onboard sensory system like a camera, infrared sensors or sonar, but to simplify, it is possible to use an overhead camera which simulate onboard sensors.

In the small sized robot soccer league, which is focused on multi-agent cooperation within a dynamic environment, ten out of eleven teams [1] used a single camera overlooking the whole field. An off-field PC is used to process the images captured by the overhead camera. In most cased, the off-field PC perform most of the processing required for coordination and control of the robots.

The communication among the robots on the field and the off-field PC is wireless and uses ordinary FM transmitter/receiver. To cope with the highly dynamic environment with robot movement of 1 m/second and ball movement up to 2 m/s the system must be very effective and handle frame rates of 30 frames/sec[1].

There is two ways to handle the fast movements in robot system. Either one can optimize the system even more to get an higher frame rate, more powerful servos, faster communication etc. or one can predict the movements and instead of having a system that reacts to the current event, the system reacts to a future event that is has anticipated.

A second type of anticipation concerns anticipation of the environment, for example the movement of other robots. Sharifi et al. [14] describe a system for the simulation league of RoboCup where the future state is used to anticipate which robot will posses the ball next, while Veloso et al. [15] anticipate the state of the whole team. This means that a seemingly passive agent is actively anticipating opportunities for collaboration.

With the system described in this report, we hope to contribute in the anticipation research field. The system have been used to compare different type of navigation and in the future more multi-robot experiments will be conducted within a dynamic environment.

1.1 General description

The system consists of small two wheeled robots that are able to navigate through a dynamic environment. The robots are modified versions of the Boe-Bot robot manufactured by Parallax.

The complexity of the environment can be adjusted using movable bricks. The robots and obstacles are tracked by color markers attached on the top of both robots and obstacle. White indicates an obstacle and for the robots two colors are used to obtain the position and orientation of the robot. The color tracking is performed by Ikaros [1]. All the navigation calculation are also made by Ikaros which is also responsible for transmitting Bluetooth commands to the robots. The system currently handles two robots but this will be expanded to 6 robots in the next year.

[1] www.ikaros-project.org

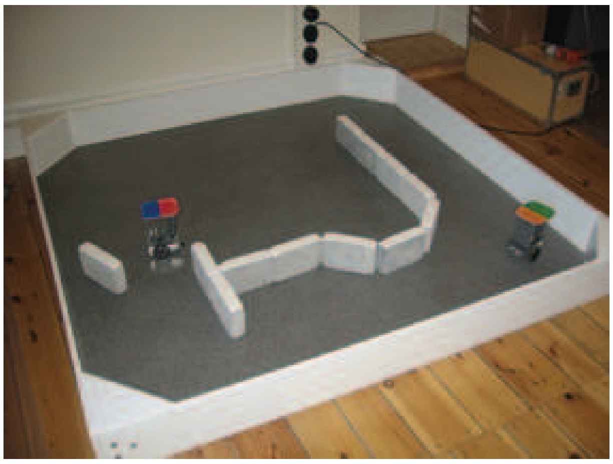

Fig. 1: *The robot area with robots and obstacles. White indicates an obstacle. The colors on top of the robots are used for localization and identification.*

2 Hardware

This section describes the main hardware used within the system at a level useful to understand the principles of the system.

2.1 Environment

The size of the area where the robots are allowed to navigate is 2×2 m (Fig. 1). This area is surrounded by 20 cm high wall that keeps the robots in place if the system loses control and also gives an excellent protection from confused cognitive scientist's walking to close the robots.

The wall is white and the area within the wall is grainy gray. The obstacles have a similar color as the floor but are marked with white at the top. The white color on the obstacles and the surrounding wall are interpreted as obstacles or as something where the robots are not allowed to be. The bricks used are $5 \times 13 \times 21$ cm which is the same height as the modified Boe-Bot robots. Each corner of the robot area has been rounded to reduce the risk of getting trapped in corners.

2.2 Robots

The robots used are two modified Boe-bots[2]. These robots are mainly used for education and are easy to learn and use. The height of the Boe-Bots has been adjusted to fit the height of the obstacles. The reason for this were constraints in our lab where it was not possible to position the overhead camera straight above the robot area, instead we had to mount the camera in an oblique angle

[2] Parallax Inc., Rocklin, California, www.parallax.com

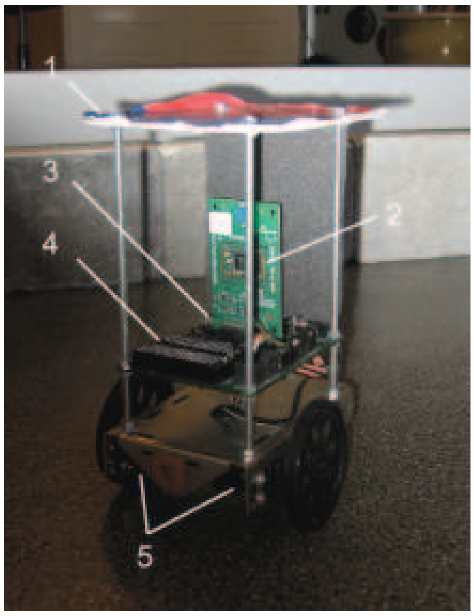

FIG. 2: *1. Color marking, 2. Bluetooth card, 3. BASIC Stamp, 4. Circuit board 5. Modified RC servos*

and transform the images to what it had looked if it had been mounted at the desired position.

The dimension of the robots, with the extended height, is $11.5 \times 13 \times 21$ cm. They use differential steering which is controlled by a BASIC Stamp [10] and can move at a velocity from approximately -0.17m/s to + 0.17m/s.

A small prototype board is located on the robots where circuits can be assembled and tested (Fig. 2). The robot operates on between 6-9 V and this is provided by four AA batteries mounted underneath. The robots are expanded with an embedded Bluetooth card for communication. This card allows for Bluetooth communication with keyword security between robots or between a computer with emulated serial communication over Bluetooth.

During our experiment, we initially had problems with a broken communication link. When the both motors were running at the same time the power for the communication was not sufficient and this resulted in lost Bluetooth connection. This was easily solved with a capacitor that supported the slow batteries and kept the Boe-Bots online. A 1000 μF capacitor was connected between pin Vss and pin Vin on the prototype board.

There exist a number of additional sensors for the Boe-Bots (camera, encoders, ir sensors, etc.) but in our set-up, no onboard sensors are used. Instead we use the overhead camera and a computer to track the position and orientation of the robots as explained above.

Parallax Programming Language (PBASIC) The Boe-Bots uses its own programming language called PBASIC and in our implementation version PBASIC 2.5 is used. The BASIC Stamp Editor is used to compile and upload the code to the Boe-Bot's memory via Universal Serial Bus (USB) or ordinarily serial communication (RS-232). The language is simple and well suited for educational purposes. In our case, we have developed protocols that handles the Bluetooth communication and runs each servo at a desired velocity. Two version of the protocol exist. Each is based on serial communication and one byte respective four bytes for controlling the robot. The first protocol had no external robot behavior debugging help (Diodes or sound onboard the robot) and were used on in our first experiment ([9]). The second protocol has been expended to four bytes to enable more information to be sent to the robot.

Bluetooth Protocol Version 1 In this first version of the protocol one speed backward and three forward speeds could be set. The protocol is simple and only uses one control byte for communication. When the robot receives a byte it replies with an identical byte. The computer compares the received byte with the one sent and if there is an exact match it assumes that the robot has received the information sent. If there is a mismatch, it retransmits the control byte. If the following bytes still does not match after 10 retransmission the Bluetooth link is assumed to be down.

When the robot receives a byte with major errors it stops. Otherwise it interprets the control message consisting of the eight bits

```
ABCD EFGH.
```

in the following way:

Bits ABCD are reserved for future use. There is preliminary work to control diodes using these bits but this is not included in the official version of the protocol. Bit EFGH are used to control the servo speed of the robot:

```
0000 = left motor backwards and right motor backwards.
0001 = left motor backwards and right motor still.
0010 = left motor backwards and right motor forward slowly.
0011 = left motor backwards and right motor forward fast.
0100 = left motor still and right motor backwards.
0101 = left motor still and right motor still.
0110 = left motor still and right motor forward slowly.
0111 = left motor still and right motor forward fast.
1000 = left motor forward slowly and right motor backwards.
1001 = left motor forward slowly and right motor forward slowly.
1010 = left motor forward slowly and right motor forward fast.
1011 = left motor forward fast and right motor backwards.
1110 = left motor forward fast and right motor forward slowly.
1111 = left motor forward fast and right motor forward fast.
```

The code in Ikaros used for communication has the following structure:

```
if (input_speed[0] == s0)
  *MyMessage |= BACKWARD;
else if (input_speed[0] == s1)
  *MyMessage |= STILL;
else if (input_speed[0] == s2)
```

```
    *MyMessage |= FORWARD1;
  else if (input_speed[0] == s3)
    *MyMessage |= FORWARD2;
  else if (input_speed[0] == s4)
    *MyMessage |= FORWARD3;
  *MyMessage <<= 4;
  // Building the second part of the message
  if (input_speed[1] == s0)
    *MyMessage |= BACKWARD;
  else if (input_speed[1] == s1)
    *MyMessage |= STILL;
  else if (input_speed[1] == s2)
    *MyMessage |= FORWARD1;
  else if (input_speed[1] == s3)
    *MyMessage |= FORWARD2;
  else if (input_speed[1] == s4)
    *MyMessage |= FORWARD3;
```

The PBASIC code in the BASIC Stamp stamp interprets the commands as follows:

```
SERIN 0,$54,20,MOTORCONTROL, [STR bData \1]
  Mask = %11111111
  SM = bData&Mask
  'Turn on spot special!
  IF SM = %00000010 THEN case67 '(S0 S2)
  IF SM = %00100000 THEN case68 '(S2 S0)
  'Backwards
  IF SM = %00000000 THEN case0 '(S0 S0)
  IF SM = %00010000 THEN case1 '(S1 S0)
  IF SM = %00000001 THEN case2 '(S0 S1)
  'Still\\
  IF SM = %00010001 THEN case3 '(S1 S1)
```

Bluetooth Protocol Version 2 Instead of one byte as in the fist protocol four bytes were used in the second version. With four bytes, more information can be sent to the robots which results in more speed levels, sound and LED control. To produce these extra features, one piezoelectric speaker and 5 LEDs were attached to the onboard prototype board. To use all the features of the protocol LEDs and the piezoelectric speaker must be connected to the prototype board and the BASIC Stamp pins in the following way:

```
  PIN  DEVICE
  P0  (Bluetooth) RX
  P1  (Bluetooth) TX
  P2  (Bluetooth) RX Flow (RTS)
  P3  (Bluetooth) TX Flow (CTS)
  P4  RESERVED
  P5  (Bluetooth) Connection Status
  P6  (Bluetooth) Mode Control
  P7  Control LED 1
  P8  Control LED 2
  P9  LED Right
  P10 SPEAKER
  P11 RESERVED
  P12 RIGHT SERVO
  P13 LEFT SERVO
  P14 LED Front
  P15 LED Left
```

Tests during development indicated that the time for sending one byte was almost the same as for sending 4 byte and that is one of the reason for the extension of the protocol. The other reason was the insufficient number of velocity levels in the first protocol.

The time needed to transmit four bytes and to receive acknowledgment from the Bot-Bots takes approximately 60 ms. When the robots are traveling at a maximum speed and the system is forced to do four retransmission, the robot can travel as much as 20 mm to 50 mm without any control from the computer.

The transmission time until it receives the acknowledgment is measured by Ikaros. the robot send the acknowledgement back immediately after it has received the command. To measure the time to command execution on the robot is harder and instead the above measurement is used to get an indication of the timing. This measure is calculated with the maximum speed which is exaggerated since this speed in seldom used during navigation. The measurements are made without the basic editor debugging window. When debugging, delays are at least twice as long. The delays are also increased when using sound.

The four byte long control message consists of the following bits:

```
LLLL LLLL    RRRR RRRR    SSSD DDDD    MMXX XXXX  bits.
```

The servos are controlled with one byte each. The first byte codes are for the left server and the second are for the right. One byte gives $2^8 = 256$ different levels of velocity for each servo. This will give approximately 128 levels in each direction which is an improvement since the previous protocol used only three levels forward and one backwards.

Each robot receives an integer between 0 and 255 in the first byte of the message then adds 600 to this value to set the pulse length that controls the motor speed. The reason for this is to minimize the amount of information sent over Bluetooth. For example, to send a zero in the first byte will result in the pulse length 600 for the right servo and 200 will result in 800.

With this protocol it is also possible to play tones. The first 3 bits in the third byte determent which tone, the robot should play.

```
Sound

000 = 264 Hz - C, DO
001 = 297 Hz - D, re
010 = 330 Hz - E, me
011 = 352 Hz - F, fa
100 = 396 Hz - G, so
101 = 440 Hz - A, la
110 = 495 Hz - B, ti
111 = 528 Hz - C, DO
```

The rest of the bits in the third bytes control the LED's. 5 LED's can be used and controlled separately. The five bits for the LED's are used as follows:

```
LED control byte: 12345

1 = Control LED 2
2 = Control LED 1
3 = LED right
4 = LED front
5 = LED left
```

The first two bits in the fourth byte are message identifier. The message is used to identify if there has been a retransmission of a message due to timeout or transmission errors. Because of the robustness of the Bluetooth communication, with the capacitor, the maximum number of retransmissions is reduced to four. The last six bits are reserved for future use.

The code in Ikaros used for communication has the following structure:

```
  // Construct the message
  if(Speed!=NULL)
  {
    SendCommand[0]=(byte)(Speed[0]*SpeedIntervall);
    SendCommand[1]=(byte)(Speed[1]*SpeedIntervall);
  }
  else
  {
    //No Speed input. Setting speed to 0
    *SendCommand|=(byte)(0.5*SpeedIntervall);
    *SendCommand<<=8;
    *SendCommand|=(byte)(0.5*SpeedIntervall);
    Notify(6,"(Boebot.cc)No speed input setting speed to half speed Interval");
  }
  if(Sound!=NULL)
    //Sound3bits
    SendCommand[2]=(byte)(Sound[0]);
  else
    SendCommand[2]=0;
  if(D!=NULL)
  {
    for(inti=0;i<NrOfDiods;i++)
    {
      SendCommand[2]=SendCommand[2]<<1;
      SendCommand[2]|=(byte)D[i];
    }
  }
  else
    SendCommand[2]|=(byte)0;

// MessSeq
SendCommand[3]|=(byte)resendcounter;
SendCommand[3]=SendCommand[3]<<6;

  // End of message construction
  //***************************
```

The receiving code in PBASIC:

```
  ' ****************** RECEIVE ***********************
  Receive:

    SERIN 0,$54,100,EXECUTE, [STR message\MessageLength]
    GOTO SEND

  ' ****************** RECEIVE END ******************

  ' ****************** SEND ************************

  SEND:
    SEROUT 1,$54,[STR message\MessageLength]
    GOTO EXECUTE

  ' ****************** SEND END ********************

  ' ****************** EXECUTE *******************
  EXECUTE:
```

```
' First part of message

M.BIT7(0) = message.BIT7(0)
M.BIT6(0) = message.BIT6(0)
M.BIT5(0) = message.BIT5(0)
M.BIT4(0) = message.BIT4(0)
M.BIT3(0) = message.BIT3(0)
M.BIT2(0) = message.BIT2(0)
M.BIT1(0) = message.BIT1(0)
M.BIT0(0) = message.BIT0(0)

' Second part of message

M.BIT7(8) = message.BIT7(8)
M.BIT6(8) = message.BIT6(8)
M.BIT5(8) = message.BIT5(8)
M.BIT4(8) = message.BIT4(8)
M.BIT3(8) = message.BIT3(8)
M.BIT2(8) = message.BIT2(8)
M.BIT1(8) = message.BIT1(8)
M.BIT0(8) = message.BIT0(8)

' Third part of message

SFQ.BIT2 = message.BIT7(16)
SFQ.BIT1 = message.BIT6(16)
SFQ.BIT0 = message.BIT5(16)
D(0) = message.BIT4(16)
D(1) = message.BIT3(16)
D(2) = message.BIT2(16)
D(3) = message.BIT1(16)
D(4) = message.BIT0(16)

' Fourth part of message

MSEQ.BIT1 = message.BIT7(24)
MSEQ.BIT0 = message.BIT6(24)

' REST RESERVED...
```

2.3 Camera

The camera used is an Axis 2130 PTZ (Fig. 3). This camera can tilt, pan and zoom and has a resolution up to 704×480 pixels. Images taken by the camera can be received over HTTP using web server running under Linux. The server sends the requested images but can also be set to stream sequence of images to get a higher frame rate. Parameter for JPEG compression and resolution can be specified in the HTTP GET request. In this setup a low compression and a big resolution is used to obtain the best performance when tracking the robots. The tilt, pan and zoom parameters are only set and stored initially and are not changed during simulation or robot experiment.

3 Software

The software developed for the system is written in C/C++ and Parallax PBASIC. PBASIC is used for the robots and the Ikaros framework is used for the rest of the system [?]. Ikaros is set to run in cycles of 250 ms, and with time steps as

Fig. 3: *An Axis 2130 PTZ is used in the robot setup. The camera has a built in web server which sends requested images in different resolution and compression*

long as this, the models must predict very well to perform smooth navigation. Not only the planning part of the navigation depends on precise models. Also the reactive part uses these models to avoid something that is too close to the robot.

3.1 Ikaros Modules

During the development phase a number of modules have been created. Some of them are very essential for the system and others are used in more specific cases. To be able to sort out the most used modules, two different setups have been designed. Each handle a scenario with two robots, either simulated or real, in which they can navigate autonomously or manually. Around 18 Ikaros modules are used to run a full experiment and this system uses the Windows version of Ikaros. Most of the modules developed could be used on any platform but there are some, like the serial communication, which is Windows specific.

Minimal setup In this setup a minimal number of modules is used and the main purpose of this setup is for testing and debugging (Fig. 4). Both robots work autonomously, but the navigation can be overridden manually for specific debugging or to study behavior of human interaction. The tracker is simulated in this system using an input raw image as obstacle and Gridworld modules for calculating the position of the robots. The Gridworld module sends the position of the robots to the world models for each robot. Those simulations then forward the position and orientation to the navigation module, which uses the goal input, if there is any, to steer the robots toward the goal. The navigation module can also gather data for statistics and forward it to Ikaros.

Robot setup This second setup is the basic one for robot experiment. It includes all modules used in a real robot experiment. The tracking component uses

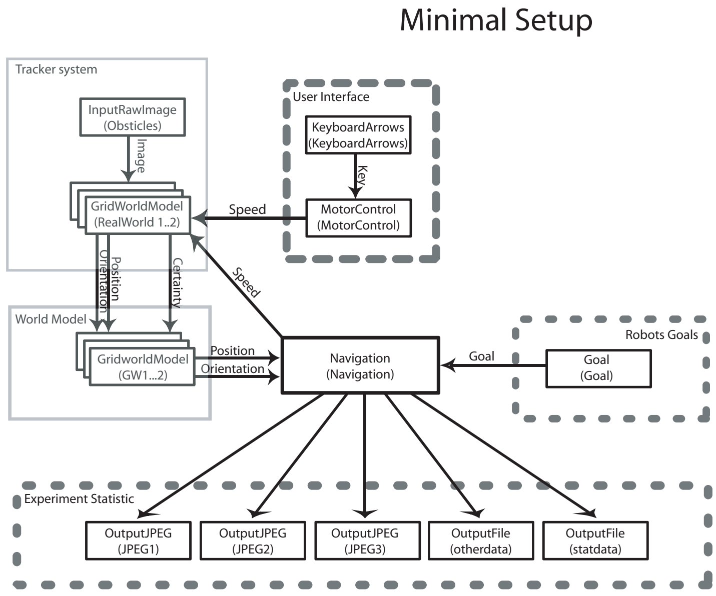

Fig. 4: *Modules in the minimal setup. All parts are simulated. The setup includes navigation, tracker system, world models and user interface. Dotted parts are optional*

the network camera module, arbitrary transform modules and the bottracker module (Fig. 5). These send the position and orientation and their certainty to the world models used by the navigation module. The navigation module then sends motor command to the robots using Bluetooth. Also in this case, it is possible to override the system behavior by adding a user interface and manually steer the robots. This is very usefully to test specific behaviors of the robot - for example that the reactive avoidance system really works in a satifactional way. It could also be used for comparing the real robot position and orientation with the simulated to measure the performance of the models used to simulate the world. Ideally the position in the simulated and the real world will be identically.

NetworkCamera This module requests an image or stream of images from the Axis 2130 camera. It downloads the images from the camera and transforms it to three raw RGB matrices, which can be interpreted by the Ikaros system. Example from the Ikaros control file:

```
 <module
    class  = "NetworkCamera"
    name  = "CAMERA"
    size_x = "704"
    size_y = "480"
    host_ip = "192.168.0.2"
    fps = "10"
    compression = "30"
/>
```

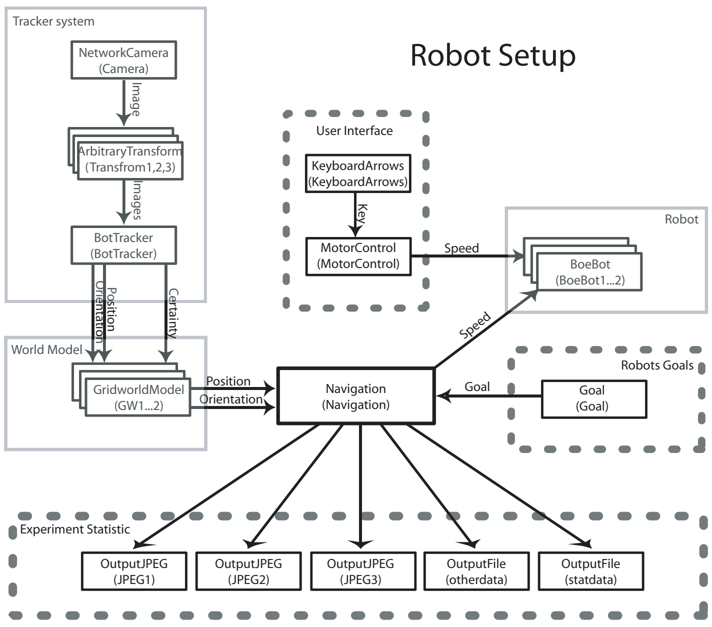

FIG. 5: *Modules in the robot setup. This setup was used for the robot experiments. The setup includes navigation, tracker system, world models, user interface and robot modules. Dotted parts are optional*

Arbitrary Transform This module performs an arbitrary transform of an image. This transformation is require to get a top view even with a camera mounted at an unknown angle above the robot area. A color image in Ikaros is divided in three matrices for each of the RGB values, this requires that three arbitrary transform modules must be used. Example from the Ikaros control file:

```
 <module
    class = "ArbitraryTransform"
    name = "Transform1"
    outputsize_x = "704"
    outputsize_y = "480"
    x0 = "73"
    y0 = "8"
    x1 = "642"
    y1 = "1"
    x2 = "110"
    y2 = "470"
    x3 = "607"
    y3 = "474"
    exp_x =  "1"
    exp_y = "0.88"
/>
```

Boe-Bots This is the Bluetooth communication module. This module transforms the wanted velocity of the motors and send them over emulated serial communication using Bluetooth. It also has functions for resending a message if

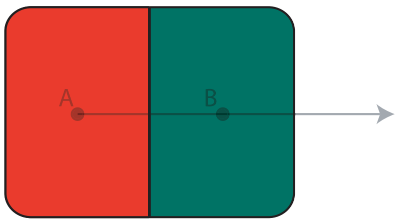

Fig. 6: *One of the robot color markers. The average position is calculated (A and B). The vector between A and B is the orientation and the average between the means are the position of the robot*

there is a timeout or if it receives an unexpected message from the robots. The only parameter that need to be defined is which port to use to the communication:

```
<module
    class = "BoeBot"
    name = "BoeBot4"
    port = " \\.\COM15"
/>
```

BotTracker This module tracks the robots and obstacle in the image. The modules receive an RGB color image using three matrices. Using these matrices the orientation and position of the robots and the position of the obstacles are found. The robots are located using two color markers (Fig. 6), which have been tested to perform well in the light conditions in our lab. Example from the Ikaros control file:

```
<module
    class = "BotTracker"
    name = "BotTracker"’

    color1 = "2.2"

    color2 = "-1.45"
    threshold1 = "0.95"
    threshold2 = "0.95"
    saturation1 = "0.05"
    saturation2 = "0.03"

    Bcolor2 = "1.55"

    Bcolor1 = "-0.25"
    Bthreshold2 = "0.95"
    Bthreshold1 = "0.95"
    Bsaturation2 = "0.05"
    Bsaturation1 = "0.03"

    search_radius = "120"
    cluster_size = "60"
    certainty_constant = "0.001"
    statistics = "no"
    grid_x = "32"
    grid_y = "32"
>
    <grid>
    </grid>
 </module>
```

ContinousGridWorld This module is a simulation of the real world. The model calculates the position of a robot in a grid, depending on wheel size, world size and speed. There exist two versions of this module. The latter one can also update the simulation with a new position. For example, if the tracking system has found the robot position and orientation with much certainty it can update the Gridworld with this position. The module is used by the navigation module to retrieve position and orientation, but it is also used to simulate the tracking system in the minimal setup.

```
<module>
    class = "ContinuousGridWorld2"
    name = "GW1"
    start_x = "0.6m"
    start_y = "0.3m"
    start_orientation_x = "1"
    start_orientation_y = "0"
    smoothness = "1"
    timebase = "0.1s"
    world_size = "2.015 m"
    wheel_radius = "0.0335m"
    wheel_distance = "0.055m"
    position_certainty_threshold = "0"
    orientation_certainty_threshold = "0"
    position_change_threshold = "0"
    orientation_change_threshold = "0"
>
    <grid>
    </grid>
</module>
```

Goal The goal module is used in the experiment to set where the start and goal are for each robot. The start and goal points can be switched during simulation depending on the input to the module.

```
<module>
    class = "Goal"
    name = "Goal"
    outputsize = "2"
    <data1>
       8 6
    </data1>
    <data2>
       24  26
    </data2>
</module>
```

Keyboard Arrows This module is used for the optional human interface part and fetches which arrow is pressed on the keyboard and sends its virtual keyboard code as output.

```
<module
    class = "KeyboardArrows"
    name = "KeyboardArrows"
/>
```

Motor Control The Motor Control module is also a part of the optional human interface. It receives a keyboard code from the Keyboard Arrows module and sends a hand coded speed forward depending on which code that was received, e.g. which arrows were pushed in the Keyboard Arrow module. This module together with the Keyboard Arrows module can be used to steer a robot manually either in simulation or reality.

```
<module
    class = "MotorControl"
    name = "MotorControl"
/>
```

Navigation The most complex part of the system is the navigation module, which is responsible for all the navigation done by the robots. The robots navigate using random, reactive, planning or anticipation approaches. The module handles different navigation approaches:

Random Control This system simply transmits random motor commands to the robot until it has reached the goal. The robot is instructed to turn toward a random orientation and then travel in this direction until an obstacle activates an obstacle avoidance system, in which case a new random direction is set.

Reactive Approach With the reactive approach, the robot always tries to go directly toward the goal. The desired path is calculated as the straight line between the current location of the robot and the goal location. This strategy will obviously have problems when there are obstacles in the way and to handle this situation a reactive avoidance system was added.

Planning System The planning system is responsible for path finding within the environment. To accomplish this, an A* based navigation algorithm is used [7]. This is a grid based navigation algorithm with full knowledge of the environment. It finds the shortest path to the goal by testing it in the grid-map. If it is unable to use the shortest path, the second shortest path is tested and so on, until a path has been found. Each robot uses the algorithm to find the best path through the robot area. The grid-map is divided into 32×32 elements with a status of either occupied or free. The planning system takes no account of where the other robots are located and only uses its own position, the desired position and the grid-map to find the path (Fig. 7).

Anticipation System The anticipation system is similar to the planning system, but also includes the movements of the other robots. If the other robots were stationary, the A* algorithm could register the other robots as obstacles. When the other robots are moving it becomes necessary to anticipate their position at each time-step in the future. To solve this, each robot has a model of the other

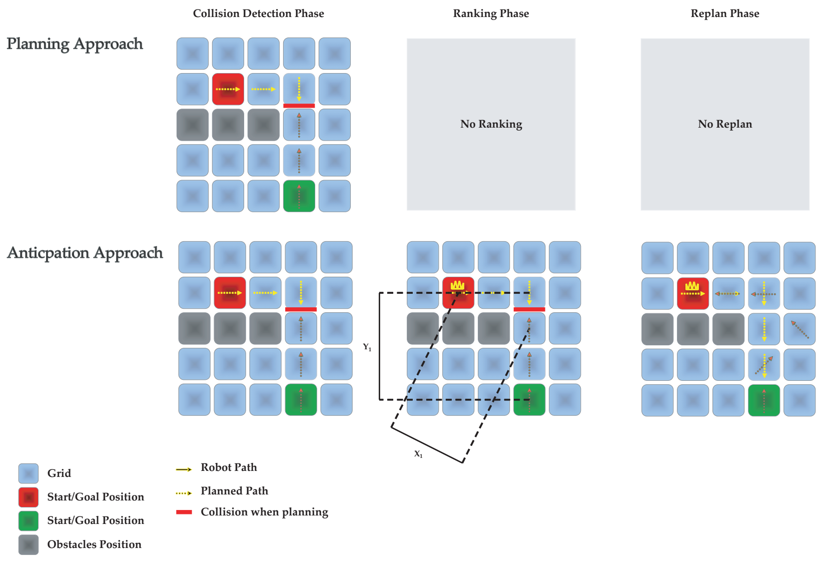

FIG. 7: *Planning and Anticipation. In the planning phase, the robot do not take any account of the other robots when planning its path to the goal, only the obstacles are used. The robot area is divided in a grid and the A* navigation algorithm is used to find a path to the goal. The anticipation phase works in the same way as the planning but also include the robots in the path finding. In the figure the system find a robot collision during its path to the goal. The one robot with the highest rank is prioried and the other robot have to chose an other way to avoid the collision.*

robot. This model is built using each robot's own planning system. For example, robot A assumes that robot B would use the path that robot A would have used if it were located at the position of robot B and heading for the goal of robot B. Before robot A tries to find its own path, it updates its model of the other robot and uses this to find the path for robot B by stepping forward in the planning and checking if there is any collisions. If there is a collision, the robot chooses an alternative path and tests if this is a valid (Fig. 7). This is repeated until a valid path is found. It should be stressed that the individual paths are not shared between the robots. Only the start and goal position is known by the other robot. With noise in the system this could lead to inaccurate models of the other robot and this could in turn lead to more activation of the reactive avoidance system. A similarly approach was presented by Guo [6].

An obvious problem arises with this approach. If both robots use the same method to find a valid path, it is possible for both robots to select the alternative path which will still result in a collision. A way to avoid this problem is to assign a rank [5] to each robot where the robot with the highest rank always takes the shortest path. For example, let the robot with the longest distance to the goal have the higher rank and let the other robot replan its path around the more

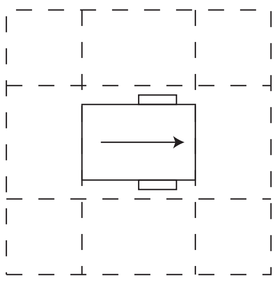

Fig. 8: *The robot with the reactive field around it. The reactive field divides the surroundings into eight regions and different avoidance behaviors are activated depending on the location of the obstacle.*

highly ranked robot. If the present robot has the lowest rank, we let A* see the other robots as a obstacle but only during that time step. This means that at just that time step there is an object at that position at some time steps later the obstacle has moved and the grid that was occupied in the first time step is free again. In the experiments, we tested three different ways to select the rank of each robot, (1) a fixed rank, (2) the robot closest to its goal would have the highest rank, and (3) the robot with the larger distance to its goal would receive the highest rank. Note that according to the last two strategies, the ranks of the robots may change when the robots move.

Reactive Avoidance A reactive avoidance system is placed on top of the other navigation systems and is activated if there is an obstacle too close to the robot. We divided the reactive area around the robot into 8 regions (Fig. 8). Three in front of the robot, one on each side of the robot and three behind the robot. The robot performs different types of avoidance behaviors depending on in which regions the obstacle was found. If an object is straight ahead, the robot turns on the spot until the obstacle has disappeared from the region and if an object is found to the left of the robot, it steers to the right to obtain a free path. Although the reactive avoidance system mainly helps the robot to reach its goal, it sometimes counteracts the control of the navigation system. For example, when the navigation system instructs the robot to turn right, the reactive avoidance system may detect an obstacle in that area and tell the robot to turn left instead.

All the information used by the navigation is received from the models of the real world and not the real world itself. This makes the system much more dynamic since it allows the tracking system to update the models at a slow speed although the motor loop can run fast to produce smooth trajectories.

The module can also collect data during an experiment. In this case, more outputs are connected that are used for Ikaros to store the data. A complete descriptions of all the parameters of the module can be found in the code documentation.

The module currently only handles two robots at this phase of the development. Example from the Ikaros control file:

```
<module
    class = "Navigation"
    name = "Navigation"
    type = "random"
    anticipation_type = "rank"
    number_of_switch = "20"
    size_x = "32"
    size_y = "32"
    number_of_robots = "2"
    CollectData = "false"
    world_complexity = "2"
    reactive_view = "300"
    reactive_size_x = "16"
    reactive_size_y = "16"
>
    <robot1
      cost = "1"
      cost_diagonal = "1.41"
      start_position_x = "10"
      start_position_y = "10"
      goal_position_x = "10"
      goal_position_y = "10"
      rank = "1"
    />
     <robot2
      cost = "1"
      cost_diagonal = "1.41"
      start_position_x = "10"
      start_position_y = "10"
      goal_position_x = "21"
      goal_position_y = "15"
      rank = "0"
    />
</module>
```

4 Conclusion

In this report we have described a multi-robot system developed to study anticipation in navigation, attention, social interaction. The system consist of two small two wheeled robots that navigate in a dynamic environment. Color markers are used to detect obstacles and robots within the area. The Beo-Bot robots have wireless communication with the Ikaros framework which runs on an ordinary computer. Ikaros performs all the processing needed for the robots, like tracking, navigation, and steering.

The system has been further described in a paper at the Swedish artificial intelligence meeting [8], the international conference of child development [**?**] and the workshop on anticipatory behavior in adaptive learning systems [9].

In [9], we compared different navigation methods with the robot system. In that paper, random, reactive, planning and anticipatory methods were compared in environments of different complexity. Random approach was always slower than the other approaches and the reactive approach was faster then the other in an environment with no obstacles. Otherwise, the planning and anticipation approaches had the best performance in the experiment. Similarl experiment are planned that will investigate the robustness of noise in a system.

The final goal for our work with the multi-robot system is to get 6 robots to play children's games with each other. Children's games are constrained to a small number of rules that determent the outcome of the game. We hope that children's games will be an excellent domain to study many of the research fields that we are currently interested in as well as being suited for demonstrations.

5 Acknowledgment

The source and documentation for the system can be downloaded, as contribution code, form the Ikaros web page (www.ikaros-project.org). The research described in this report was supported by the project MindRACES (European Project IST- 511931).

References

1. Minoru Asada, Hiroaki Kitano, Itsuki Noda, and Manuela Veloso. Robocup: Today and tomorrow -what we have have learned. *Artificial Intelligence*, 110:193–214, 1999.
2. Rodney A. Brooks, Cynthia Breazeal, Matthew Marjanovic, Brian Scassellati, and Matthew M. Williamson. The cog project: Building a humanoid robot. *Lecture Notes in Computer Science*, 1562:52–87, 1999.
3. J. Casper. Human-robot interactions during the robot-assisted urban search mid rescue response at the world trade center. 2002.
4. I Dautenhahn, K; Werry. Towards interactive robots in autism therapy: Background, motivation and challenges. *Pragmatics & Cognition*, 12:1–35, 2004.
5. Michael Erdmann and Tomas Lozano-Perez. On multiple moving objects. *Algorithmica*, 2:477–521, April 1987.
6. Y. Guo and L. Parker. A distributed and optimal motion planning approach for multiple mobile robots. In *Proceedings IEEE International Conference on Robotics and Automation*, 2002.
7. N. J. Raphael B. Hart, P. E. Nilsson. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics SSC4*, 2:100–107, 1968.
8. B. Johansson and C. Balkenius. It's a child's game: Investigating cognitive development with playing robots. In *International Conference on Development and Learning*, pages 164– 164, Osaka, Japan, 2005.
9. B Johansson and C Balkenius. The benefits of anticipation: an experimental study. In Sigaud O. Pezzulo G. Butz, M. and G. Baldassarre, editors, *Proceedings of third workshop on anticipatory behavior in adaptive learning systems (ABiALS)*, 2006.
10. Andy Lindsay. *Robotics with the Boe-Bot: Student Guide: Version 2.2.* Parallax Press, 2004.
11. Hiroyuki Miyamoto, Stefan Schaal, Francesca Gandolfo, Hiroaki Gomi, Yasuharu Koike, Rieko Osu, Eri Nakano, Yasuhiro Wada, and Mitsuo Kawato. A kendama learning robot based on bi-directional theory. *Neural Netw.*, 9(8):1281–1302, 1996.
12. Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.
13. R.; Aoyarna C.; Matsunaga S.; Higaki N.; Fujimura K. Sakagami, Y.; Watanabe. The intelligent asimo: system overview and integration. *Intelligent Robots and System*, 3:2478–2483, 2002.

14. H. Aavani A. Sharifi, M. Mousavian. Predicting the future state of the robocup simulation environment: heuristic and neural networks approaches. *Systems, Man and Cybernetics*, 1:32–27, 2003.
15. Manuela Veloso, Peter Stone, and Michael Bowling. Anticipation as a key for collaboration in a team of agents: A case study in robotic soccer. In Paul S. Schenker and Gerard T. McKee, editors, *Proceedings of SPIE Sensor Fusion and Decentralized Control in Robotic Systems II*, volume 3839, pages 134–143, Bellingham, September 1999.
16. Cao; Jingliang Zhou; Yi Huang; Lewis Frank L. Yanwen, Huang; Qixin. Multiagent cooperation based entertainment robot. *Robotica*, 24(5):643–648, 2006.