The IRIDIA TAM: A Device for Task Abstraction for the E-Puck Robot

A. Brutschy, G. Pini, N. Baiboun, A. Decunire, and M. Birattari
The TAM: A device for task abstraction for the e-puck robot

Arne Brutschy
Acknowledgements

The author would like to thank everyone that contributed towards the design, test and publishing of the TAM. In alphabetical order:

- Anthony Antoun
- Nadir Baiboun
- Mauro Birattari
- Manuele Brambilla
- Manuel Castillo-Cagigal
- Antal Decugnière
- Marco Dorigo
- Simon Lefort
- Marco Frison
- Gianpiero Francesca
- Lorenzo Garattoni
- Álvaro Gutiérrez
- Nam-Luc Tran
- Giovanni Pini

Furthermore, the author would like to everybody that helped with the assembly of the final TAM:
# Contents

## Acknowledgements

## 1 Introduction

1.1 Note on updated version ........................................ 2
1.2 Using the TAM ................................................. 2

## 2 Motivation

2.1 Swarm Intelligence ............................................. 4
2.2 The e-puck Robot ............................................... 5
2.3 Possible Experiments .......................................... 6

## 3 Concept

3.1 Basic concept .................................................. 7
3.2 Usage examples ............................................... 7

## 4 Simulator Implementation .................................. 10

## 5 Real-World Implementation ................................. 12

5.1 Hardware ...................................................... 12
5.2 Firmware ..................................................... 13

## 6 Conclusions and Future Work ............................. 14

6.1 Published works .............................................. 15

## A Appendix

A.1 Simulation ..................................................... 16
A.1.1 ARGoS 1 Entity ............................................ 16
A.1.2 ARGoS 2 Entity ............................................ 25
A.2 Real World .................................................... 34
A.2.1 Hardware ................................................... 34
A.2.2 Firmware .................................................. 47

## List of Figures

## References

## Bibliography
1 Introduction

This technical report describes the TAM, a device for task abstraction for the e-puck robots (TAM = task abstraction module. The e-puck is a simple and cheap robot, designed for education and research purposes (Mondada et al., 2009). Because of its simplicity, the e-puck cannot be used for complex experiments (i.e., the robot has not gripper or any other manipulation device).

We have designed a device that works around this deficiency by emulating tasks which can be served by the robot. This is accomplished by placing programmable modules in the environment. A robot can work on the specific task simply by driving into such a module. The robot recognizes the module by its color, which can be changed by the experimenter. The module can detect a robot’s presence by a light barrier. The device, referred to as TAM, is programmable, and can therefore react to events as requested by the experimenter.

In this document, we present the overall reasoning that stands behind the design of the device, as well as the actual design in software and hardware.

1.1 Note on updated version

Please note that this revision of the technical report covers a new version of the TAM (version 5, revision D). The new version does provide not a whole array of tasks but only a single task, thus giving the researcher more flexibility in terms of placement in the experiment. Additionally, the new version of the TAM is a stand-alone device independent from any support cables or computers. This independence accomplished by powering the module with a battery (the same as used by the e-puck). Additionally, the TAM features a wireless radio in order to report statistics to the experiment’s workstation and to receive commands. This also allows the researcher more freedom in placing the device in the environment.

1.2 Using the TAM

All necessary information to replicate the TAM can be found in this technical report and the supplementary on-line material of the accompanying journal paper (Brutschy et al., 2013).
All the material in the technical report and the supplementary pages is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License. In practice this license lets you remix, tweak, and build upon the TAM non-commercially, as long as you credit the TAM and its authors, and license your new creation under the identical terms. You are free to use the TAM or any reproduction thereof in your experiments.

If you use the TAM in your experiments, please cite this technical report as follows:

```latex
@techreport{BruPinBai-etal2010:IridiaTAM,  
  Author = {Arne Brutschy},  
  Title = {The \{IRIDIA TAM\}: A device for task abstraction for the e-puck robot},  
  Institution = {IRIDIA, Université Libre de Bruxelles},  
  Year = {2010},  
  Number = {TR/IRIDIA/2010-015},  
  Address = {Brussels, Belgium},}
```

Additionally, please report your publications to Arne Brutschy (arne.brutschy@ulb.ac.be) so that we can update the list of publications using the TAM (see page 72).
2 Motivation

In this chapter, we introduce the reasoning that stands behind the creation of the TAM. First, we give a short introduction to swarm intelligence and the related swarm robotics. Afterwards, we introduce the e-puck educational robot, for which the TAM has been designed. Finally, we will present classes of experiments and possible ideas that can be enabled by using the TAM.

2.1 Swarm Intelligence

_Swarm intelligence_ (commonly abbreviated with SI) is a branch of artificial intelligence based on the collective behavior of natural systems like social insects (Bonabeau et al., 2000; Garnier et al., 2007). It is commonly defined as:

\[
\text{\ldots} \text{any attempt to design algorithms or distributed problem-solving devices inspired by the collective behavior of social insects and other animal societies.}
\]

(Bonabeau et al., 1999)

The expression “swarm intelligence” was originally introduced by Beni and Wang (1989) for cellular robotics systems, but is nowadays widely used in the field of artificial intelligence. Swarm intelligence systems consist typically of a population of relatively simple individuals. Similar to social insects, the individuals follow simple behavioral rules and rely purely on local sensing and communication. Therefore, the individuals interact with each other and the environment on a local basis only. Because of the dynamic nature of swarm systems, these interactions are to a certain degree of a stochastic nature. Although there is no centralized control, such systems show the emergence of global behaviors that transcend the behavioral repertoire of the single individual: the swarm _self-organizes_. Natural examples of swarm intelligence include ant colonies (Detrain and Deneubourg, 2006), bird flocking (Reynolds, 1987), animal herding (Gautrais et al., 2007), colony of bacteria (Ben-Jacob et al., 2000), and fish schooling (Grünbaum et al., 2004).

Swarm intelligence systems have a few invariant properties:

- the system is composed of many, relatively homogeneous individuals;
- interactions among the individuals are based on simple behavioral rules that exploit only local information;
Figure 2.1: The e-puck robot. Left: A picture of the real robot. Right: The e-puck as represented in simulation. In the experiments presented in this article we employ the wheel actuators (with a maximum speed of 8 cm/s), the proximity sensors for obstacle avoidance, and the camera for the detection of tasks in the environment. Note that the e-puck does not have any manipulation capabilities.

- control is fully distributed among a number of individuals;
- communications among the individuals happen in a localized way;
- system-level behavior results from the interactions of individuals with each other and with their environment and appears to transcend the behavioral repertoire of the single individual;
- the overall response of the system is quite robust and adaptive with respect to changes in the environment.

Artificial systems that were developed following the swarm intelligence approach are usually flexible, robust, adaptive and scalable (Camazine et al., 2003; Cao et al., 1997). Nowadays, an increasing amount of human created algorithms fall into the domain of swarm intelligence, for example algorithms for optimization (Dorigo and Stützle, 2004), data analysis (Abraham et al., 2006) or network routing (Di Caro and Dorigo, 1998).

2.2 The e-puck Robot

The e-puck\(^1\) is a small wheeled robot, which has been designed by Dr. Francesco Mondada and his team at EPFL in 2006, with the purpose to be both a research and an educational tool for universities and schools Mondada et al. (2009).

\(^1\)http://www.e-puck.org/
Both the hardware design and the software libraries have been entirely developed as open-source projects; free access to related documents has greatly increased the robot’s success. The e-puck features 8 infra-red proximity sensors that can double as light sensors, a forward-facing colour camera (resolution of 640x480 pixels), 8 red LEDs and the wheels actuator. In the experiments presented in this article we employ the wheel actuators (with a maximum speed of 8 cm/s), the proximity sensors for obstacle avoidance, and the camera for the detection of tasks in the environment. Note that the e-puck does not have any manipulation capabilities.

Several extensions exist, including ground sensors to recognize gray scale colors. The ARGoS simulation framework (see Section 4) simulates the whole set of sensors and actuators of the e-Puck.

### 2.3 Possible Experiments

The TAM is meant to be used for experiments that exceed the capabilities of an e-puck alone. As the e-puck possesses no gripper or any other actuator able to manipulate objects, experiments with the e-puck alone are limited to tasks that are purely based on robot navigation (i.e., exploring the environment, or pushing a box).

The TAM enables the researcher to abstract from any task that requires the presence of a robot. It therefore enables the e-puck to accomplish a wide range of abstracted tasks. This, the TAM broadens the range of problems that can be researched using the e-puck considerably.

Possible experimental scenarios are:

- Foraging scenarios, in which robots have to harvest and transport (virtual) material;
- Dirt removal scenarios;
- Search and rescue scenarios;
- Construction scenarios;
- ...

In general, the TAM allows to study task allocation and partitioning problems in a general way, as almost all tasks can be abstracted by using it. This allows to study a whole classes of different problems just by reprogramming and repositioning the TAM in the environmental setup.
3 Concept

In this chapter, we explain the concept of the TAM. As the e-pucks are not capable of grasping objects or any other advanced manipulation of their environment, we developed a device to abstract from tasks of complex nature. Figure 3.1 shows a conceptual representation of the this device, referred to as TAM (task abstraction module).

3.1 Basic concept

Each module is equipped with three RGB LEDs that can be detected by the e-puck robots using their color camera. A light barrier can detect the presence of a robot within the module. Upon the detection of a robot, the module reacts by changing the color of its LEDs following a user defined logic. In most experiments, the color of the LEDs represent the different types of task.

Conceptually, each module represents a stationary task. Whenever a robot navigates into a module, the robot is considered to be working on the associated task. Tasks are identified by the color of the module. Colors are also used to signal to the robot if a task as been completed or if it is still in progress. The behavior of each module is programmable, thus it is up to the researcher to define the behavior of the abstracted task.

3.2 Usage examples

2-task allocation problem

An example usage would be a 2-task allocation problem. In this problem, two different types of tasks are scattered in the environment, and have to be tackled by the robots. The two tasks could be represented by green and blue LEDs, respectively. Thus, robots that want to work on the first task type can simply search for green LEDs in the environment. As soon as a robot navigates towards a task of his choice, the module would detect its presence with the light barrier. The module could then switch to red LEDs in order to signal a task being currently worked on. As soon as the LEDs are turned off (i.e., the robot does not perceive the red LED anymore),
the task is considered to be completed and the robot can exit the module. The
time it takes to complete this task is configurable by the researcher and could be,
for example, uniformly sampled in a given interval. Upon completion, the module
would report the result to the researcher for experiment statistics using a wireless
connection.

This experiment outlines how the modules can abstract various tasks available in
the environment. An experiment of this type has been published in Brutschy et al.
(2011).

**2-task partitioning problem**

Another example could be to use a number modules to exchange objects between
robots working in two different parts of the environment. Modules would be arranged
in pairs, each with its back to another module.

Consider a task partitioned into two sub-tasks. The robots working in the first sub-
task can exchange objects with robots working in the second sub-task by depositing
objects in a cache site. The robots working in the second sub-task could then
pick them up from this cache site. This experimental setup could be abstracted
using the TAM as follows. A green module means that an object is available there.
Analogously, a blue module means that an object can be deposited there. By using
this abstraction, when a robot enters a module in which the LEDs are lit up in green,
we consider that the robot picks up an object from that module. When a robot enters a module in which the LEDs are lit up in blue, we consider that the robot drops an object in that module. In both cases, when the module perceives the presence of the robot, it reacts by turning the LEDs red, until the robot has left. Once the robot has left, the module, previously green, turns off and the corresponding module on the other side turns blue signaling that the spot is now available again for dropping an object. Conversely, if the robot leaves after dropping an object, the module, previously blue, turns off and the corresponding module on the other side turns green signaling that an object is available for being picked up.

This simple logic allows us to simulate object transfer through a cache site. An experiment of this type has been published in Pini et al. (2011).
4 Simulator Implementation

**Note:** The simulation-related information on the TAM given in this section is intended for ARGoS version 1 and 2. An updated version for ARGoS version 3 is in preparation.

The TAM has been implemented as actuated device in the ARGoS simulation framework (Pinciroli et al., 2012). ARGoS is a discrete-time, physics-based simulation environment that allows one to simulate experiments at different levels of detail. For example, it can simulate robotic experiments using simulate kinematics in a bi-dimensional space. Other experiments, for example with flying robots, might call for more complex simulation based on a dynamics physics engine. A common control interface provides transparent access to real and simulated hardware, allowing the same controller to run also on the real robots without modifications.

The TAM can be programmed in simulation similar to the robots using a controller written in C++. Each module has a proximity sensor, which represents the light barrier. It can be used to detect the presence of a robot inside the module. Additionally, the module has a LED actuator, which allows the running controller to change the color of the module, and thus communicate states to the robots in the environment or the module. The dimensions of the TAM are modeled after the real world prototype, and can be placed in arbitrary position and numbers in the environment.

In contrary to the robots, controllers written for the simulated TAM can not be transferred 1:1 on the real world device. This is no major drawback as the controllers for the TAM are usually very simple.
Figure 4.1: Screenshot from the ARGoS 1 simulator which shows a single TAM. It is positioned in a way that four of the modules can be accessed by the robots. Here, each module represents a blue or a green task (left module or the two right modules, respectively). When a task is currently being processed by a robot, it turns red (second module from the left). The module switches its LEDs off in order to signal to the robot currently working on it that the task has been completed.

Figure 4.2: Screenshot from a complete experimental scenario in the ARGoS 1 simulator. Six modules are positioned in the environment, representing tasks that randomly appear at the borders of the arena.
5 Real-World Implementation

In this section we describe the real-world hardware implementation of the TAM. First, we will focus briefly on the hardware implementation, and the we will outline the low-level firmware that allows the TAM to be versatile and re-programmable depending on the experiment.

5.1 Hardware

The real-world, hardware implementation of the TAM is based on the Arduino project (Banzi, 2008):

> Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It’s intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments.

> Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software on running on a computer.

http://www.arduino.cc/

The Arduino project provides a tested and proven base that can be easily extended. With only a few additional parts, a TAM can be realized. The schematics in Appendix A.2 are build on the openly available Arduino Duemilanova, which have been extended to support the LEDs and light barrier. The circuits have been laid out with Cadsoft Eagle, a CAD software for electronics engineering that is available for free.¹

The design of the Arduino allows the researcher to connect the TAM directly via USB to a host computer. Controllers can be written on the microcontroller of the

¹http://www.cadsoft.de/
Figure 5.1: Left: Photo of the back side of the TAM. The TAM uses the same battery as the e-puck robot, which eases charging and handling of the batteries during the course of an experiment. The TAM features an XBee mesh networking module, which allows the researcher to control a large number of TAMs. Right: The TAM is compatible with various extensions of the e-puck. The e-puck shown in the image uses the range and bearing sensor (Gutiérrez et al., 2008), the embedded computer running Linux, and the omni-directional camera. Please note that these photos show an prototype of the TAM with a 3d-printed body.

TAM directly using this connection, without the requirement of a specialized hardware programmer. The same USB connection can also be used by the device to report back to the researcher, for example to keep statistics of an ongoing experiment.

The schematics of the electronic circuit required to build a TAM can be found in Appendix A.2.1. The circuit boards, which form the electronics as well as the body of the TAM by interconnecting them similar to a jigsaw-puzzle, can be found in Appendix A.2.1.

5.2 Firmware

The Arduino project provides a simple, yet powerful language. As it is close to classical C, it is easy to understand and can thus quickly be taught to students. The Arduino project provides an extensive library for modifying the state of the microcontroller and its attached devices. This software has been extended in order to drive all LEDs and light barrier of the TAM. The source code of an example controller written in the Arduino language can be found in Appendix A.2.2. The source can be compiled using the Arduino development environment, and subsequently loaded onto the microcontroller using USB.
6 Conclusions and Future Work

In this technical report we described the TAM, a device for task abstraction for the e-puck robots. The e-puck is a simple and cheap robot, designed for education and research purposes. Because of its simplicity, the e-puck cannot be used for complex experiments (i.e., the robot has not gripper or any other manipulation device).

We have presented a device that works around this deficiency by abstracting from the actual task by placing a device in the environment which represent tasks. The device, referred to as task allocation module (TAM), is programmable, and can therefore react to events as requested by the researcher. We explained the motivation that stood behind the creation of the TAM, and the concepts that are underlying its design. We presented the implementation of the device in the ARGoS simulation framework as well as in a real-world hardware.

All the components of this implementation are open-source, which allows other research groups to adapt the TAM for their research and even re-implement it for any other mobile robot.

We produced a total of 50 TAMs for the used in our laboratory. To date, experiments with up to 20 TAMs and 20 e-puck robots have been conducted successfully.
6.1 Published works

The TAM has been successfully used in several scientific experiments over the last years, be it in journal papers, conference papers, or master’s thesis. If you are using the TAM for your publications and they are not listed, please report your publications to Arne Brutschy (arne.brutschy@ulb.ac.be).

See page 72 for a full list of publications using the TAM.

International journal papers:

- Pini et al. (2011)
- Brutschy et al. (2012)
- Pini et al. (2013)

Peer-reviewed conference papers:

- Brutschy et al. (2011)
- Pini et al. (2012)
- Frison et al. (2010)

Technical reports:

- Francesca et al. (2012)

Masters thesis:

- Frison (2010)
- Tran (2010)
- Baiboun (2010)
- Compère (2011)
A Appendix

A.1 Simulation

Note: The simulation-related information on the TAM given in this section is intended for ARGoS version 1 and 2. An updated version for ARGoS version 3 is in preparation.

A.1.1 ARGoS 1 Entity

/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; version 2.
 * *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
 * 02111-1307 USA
 */

/**
 * @file <simulator/swarmanoid_space/entities/gate_entity.h>
 *
 * @brief This file contains the definition of the gate entity.
 *
 * This file contains the class definition of the gate entity.
 */

16
```cpp
#ifndef _CGATEENTITY_H_
#define _CGATEENTITY_H_

#include <vector>

/* To avoid dependency problems when including */
namespace ahss {
    class CGateEntity;
};

#include "robot_entity.h"
#include "colored_entity.h"
#include "led_enabled_entity.h"
#include "swarmanoid_datatypes/colour.h"

namespace ahss {

class CGateEntity: public CRobotEntity, public CColoredEntity,
virtual public CLedEnabledEntity {

    // Associations
    unsigned int m_unNumDoors;
    vector<unsigned int> m_vecDoorStatus;
    double m_fRGBA[4];

    // Operations
    protected:
    void MakeGate( void );
    CVector3 ComputeLedOffset( double f_start_offset, int n_multiplier );

    public:
    CGateEntity( );
    ~CGateEntity( );

};
```
virtual int Init( const TConfigurationTree
	t_configuration_tree );

const unsigned int GetNumDoors( void ) const;

void SetNumDoors( const unsigned int un_door );

const unsigned int GetDoorStatus( const unsigned int
	un_door_index );

void SetDoorStatus( const unsigned int un_door_index , const
	unsigned int un_task );

const double* GetRGBA( void ) const;

/**
 * @brief Returns the number identifying uniquely the type
 * of the entity.
 *
 * @return The type number.
 */
virtual const int GetType( ) const;

virtual void Accept(EntityVisitor *visitor);

static const double GATE_HEIGHT;
static const double GATE_THICKNESS;

static const double GATE_SUPPORT_WIDTH;
static const double GATE_SUPPORT_HEIGHT;
static const double GATE_SUPPORT_THICKNESS;

static const double GATE_INTERDOOR_WIDTH;
static const double GATE_INTERDOOR_HEIGHT;
static const double GATE_INTERDOOR_THICKNESS;

static const double GATE_DOOR_WIDTH;

static const unsigned int NO_TASK;
static const unsigned int TASK_WIP;
static const unsigned int GREEN_TASK;
static const unsigned int BLUE_TASK;

static const CColour NO_TASK_COLOR;
static const CColour TASK_WIP_COLOR;
static const CColour GREEN_TASK_COLOR;
static const CColour BLUE_TASK_COLOR;
};
#
#include "gate_entity.h"
#include "entities/entity_visitor.h"
#include "string_utilities.h"
using namespace ahss;

const string CONFIGURATION_GATE_DOORS = "doors";
const string CONFIGURATION_GATE_RGBA = "rgba";
const double GATE_MASS = 1;
const double CGateEntity::GATE_HEIGHT = 0.15;
const double CGateEntity::GATE_THICKNESS = 0.05;
const double CGateEntity::GATE_SUPPORT_WIDTH = 0.20;
const double CGateEntity::GATE_SUPPORT_HEIGHT = 0.15;
const double CGateEntity::GATE_SUPPORT_THICKNESS = 0.025;
const double CGateEntity::GATE_INTERDOOR_WIDTH = 0.20;
const double CGateEntity::GATE_INTERDOOR_HEIGHT = 0.15;
const double CGateEntity::GATE_INTERDOOR_THICKNESS = 0.06;
const double CGateEntity::GATE_DOOR_WIDTH = 0.095;
const unsigned int CGateEntity::NO_TASK = 0;
const unsigned int CGateEntity::TASK_WIP = 1;
const unsigned int CGateEntity::GREEN_TASK = 2;
const unsigned int CGateEntity::BLUE_TASK = 3;
const CColour CGateEntity::NO_TASK_COLOR = CColour::White;
const CColour CGateEntity::TASK_WIP_COLOR = CColour::Red;
const CColour CGateEntity::GREEN_TASK_COLOR = CColour::Green;
const CColour CGateEntity::BLUE_TASK_COLOR = CColour::Blue;
CGateEntity::CGateEntity( ) : CRobotEntity( )
{
    SetMass ( GATE_MASS );
}
/******************
/********************/
CGateEntity::~CGateEntity( )
{}
/******************
/********************/
int CGateEntity::Init ( const TConfigurationTree t_configuration_tree )
{
    CRobotEntity::Init ( t_configuration_tree);

    TConfigurationTree::iterator it_node;

    /* get the doors’ number */
string s_doors;
int nTmpDoor;
s_doors = CExperimentConfiguration::GetNodeValue(t_configuration_tree, CONFIGURATION_GATE_DOORS);
if (s_doors == "" || StringToInt(s_doors, nTmpDoor) == -1)
{
    CSwarmanoidLogger::LogErr() << "[FATAL] Missing required " << CONFIGURATION_GATE_DOORS " > subsection in " << m_sId << " entity section of configuration file." << endl;
    return -1;
} else if ((nTmpDoor % 2) != 0)
{
    CSwarmanoidLogger::LogErr() << "[FATAL] Required " << CONFIGURATION_GATE_DOORS " > parameter must be even." << endl;
    return -1;
}
m_unNumDoors = static_cast<int>(nTmpDoor);

MakeGate();

// get the rgb and alpha values
string s_rgba = CExperimentConfiguration::GetNodeValue(t_configuration_tree, CONFIGURATION_GATE_RGBA);
if (s_rgba == "" || CExperimentConfiguration::GetNReals(s_rgba, m_fRGBA, 4) == -1)
{
    CSwarmanoidLogger::LogErr() << "[WARNING] No gate color specified, using default color." << endl;

    // set default color
    m_fRGBA[0] = 0.8;
    m_fRGBA[1] = 0.55;
    m_fRGBA[2] = 0.25;
    m_fRGBA[3] = 1;
}
SetEntityColour(CColour(m_fRGBA[0], m_fRGBA[1], m_fRGBA[2], m_fRGBA[3]));

/* Everything OK */
return 0;
}

const double* CGateEntity::GetRGBA( void ) const
{    return m_fRGBA; }

/***************************************************************************/
/***************************************************************************/

const unsigned int CGateEntity::GetNumDoors ( void ) const
{
    return ( const unsigned int) m_unNumDoors;
}

/***************************************************************************/
/***************************************************************************/

void CGateEntity::SetNumDoors( const unsigned int un_door )
{
    if ( un_door < 2 || un_door % 2 != 0 )
    {
        CSwarmanoidLogger::LogErr( ) << "[FATAL] CGateEntity.SetNumDoor(
        ) - Bad door number." << endl;
        return;
    }
    m_unNumDoors = un_door;

    MakeGate( );
}

/***************************************************************************/
/***************************************************************************/

const unsigned int CGateEntity::GetDoorStatus( const unsigned int un_door_index )
{
    if ( m_unNumDoors <= un_door_index )
    {
        CSwarmanoidLogger::LogErr( ) << "[FATAL] CGateEntity.GetDoorStatus( ) - Overbound door number." << endl;
        return m_vecDoorStatus[m_vecDoorStatus.size( ) - 1];
    }

    return m_vecDoorStatus[un_door_index / 5];
}

/******************************************************************************/
void CGateEntity::SetDoorStatus( const unsigned int un_door_index,
const unsigned int un_task )
{
if ( m_unNumDoors <= un_door_index )
{
    CSwarmanoidLogger::LogErr() << "[FATAL]
CGateEntity.SetDoorStatus( ) - Overbound door number."
<< endl;
    return;
}

    // update internal representation
m_vecDoorStatus[un_door_index] = un_task;

    // map task to led color
CColour cColor;
switch ( un_task )
{
    case NO_TASK:
        cColor = NO_TASK_COLOR;
        break;
    case TASK_WIP:
        cColor = TASK_WIP_COLOR;
        break;
    case GREEN_TASK:
        cColor = GREEN_TASK_COLOR;
        break;
    case BLUE_TASK:
        cColor = BLUE_TASK_COLOR;
        break;
    default:
        CSwarmanoidLogger::LogErr() << "[FATAL]
CGateEntity.SetDoorStatus( ) - Unrecognized task."
<< endl;
}

    unsigned int index = un_door_index * 5;
    SetLedColor( index++, cColor );
    SetLedColor( index++, cColor );
    SetLedColor( index++, cColor );
    SetLedColor( index++, cColor );
    SetLedColor( index, cColor );
}
const int CGateEntity::GetType() const
{
    return ENTITY_TYPE_GATE;
}

/****************************************/
/****************************************/

void CGateEntity::Accept(EntityVisitor *visitor)
{
    visitor->Visit(this);
}

/****************************************/
/****************************************/

void CGateEntity::MakeGate()
{
    m_vecDoorStatus.clear();
    SetNumberOfLeds( 0 );

    const double kfGateWidth = ((CGateEntity::GATE_DOOR_WIDTH +
    CGateEntity::GATE_INTERDOOR_THICKNESS) * (m_unNumDoors / 2.0)) -
    CGateEntity::GATE_INTERDOOR_THICKNESS +
    CGateEntity::GATE_SUPPORT_THICKNESS;

    const double kfLOffset = ((CGateEntity::GATE_SUPPORT_THICKNESS -
    kfGateWidth) / 2.0);
    const double kfLeftStartOffset = kfLOffset + 0.01;
    const double kfMiddleStartOffset = kfLOffset +
    CGateEntity::GATE_DOOR_WIDTH / 2.0;
    const double kfRightStartOffset = kfLOffset +
    CGateEntity::GATE_DOOR_WIDTH - 0.01;

    const double kfTmp1 = kfMiddleStartOffset - 0.02;
    const double kfTmp2 = kfMiddleStartOffset + 0.02;

    for ( unsigned int i = 0; i < m_unNumDoors; i++ )
    {
        m_vecDoorStatus.push_back( NO_TASK );

        AddLed ( NO_TASK_COLOR, ComputeLedOffset( kfLeftStartOffset, i )
    );
    }

    AddLed ( NO_TASK_COLOR, ComputeLedOffset( kfTmp1, i )
    );
AddLed ( NO_TASK_COLOR, ComputeLedOffset( kfMiddleStartOffset, i ));
AddLed ( NO_TASK_COLOR, ComputeLedOffset( kfTmp2, i ));
AddLed ( NO_TASK_COLOR, ComputeLedOffset( kfRightStartOffset, i ));
}

CVector3 CGateEntity::ComputeLedOffset( double f_start_offset, int n_i_multiplier )
{
  CVector3 cLedOffset;
  cLedOffset.x = f_start_offset + (n_i_multiplier / 2) *
  (CGateEntity::GATE_DOOR_WIDTH +
   CGateEntity::GATE_INTERDOOR_THICKNESS);
  if ( (n_i_multiplier % 2) == 0 )
    { cLedOffset.y = (CGateEntity::GATE_THICKNESS / 2.0); }
  else
    { cLedOffset.y = -(CGateEntity::GATE_THICKNESS / 2.0); }
  cLedOffset.z = CGateEntity::GATE_SUPPORT_HEIGHT / 4.0;
  return cLedOffset;
}

A.1.2 ARGoS 2 Entity

/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; version 2.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 */
namespace argos {
    class CBoothEntity;
    class CEmbodiedEntity;
    class CControllableEntity;
    class CLedEquippedEntity;
    class CQuaternion;
}

#include <argos2/simulator/space/entities/composable_entity.h>
#include <argos2/simulator/space/entities/led_equipped_entity.h>

namespace argos {

    class CBoothEntity : public CComposableEntity {

    public:

        CBoothEntity();
        virtual ~CBoothEntity();

        virtual void Init(TConfigurationNode& t_tree);
        virtual void Reset();
        virtual void Destroy();

        virtual CEntity& GetComponent(const std::string& str_component);
        virtual bool HasComponent(const std::string& str_component);

        virtual void UpdateComponents();

    }
inline CEmbodiedEntity& GetEmbodiedEntity() {
    return *m_pcEmbodiedEntity;
}

inline CControllableEntity& GetControllableEntity() {
    return *m_pcControllableEntity;
}

inline CLedEquippedEntity& GetLEDEquippedEntity() {
    return *m_pcLEDEquippedEntity;
}

inline virtual std::string GetTypeDescription() const {
    return "booth_entity";
}

inline virtual void Accept(CEntityVisitor& visitor) {
    visitor.Visit(*this);
}

private:

void SetLedPosition();

private:

CEmbodiedEntity* m_pcEmbodiedEntity;
CControllableEntity* m_pcControllableEntity;
CLedEquippedEntity* m_pcLEDEquippedEntity;

};

#endif

/*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; version 2.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
namespace argos {

/****************************************/
/**/  
static const Real BOOTH_WIDTH = 0.12f;
static const Real BOOTH_HEIGHT = 0.12f;
static const Real BOOTH_DEPTH = 0.12f;
static const Real BOOTH_LATERAL_WALL_THICKNESS = 0.012f;
static const Real BOOTH_BACK_WALL_THICKNESS = 0.024f;
static const Real BOOTH_LED_X = 0.0f;
static const Real BOOTH_LED_Y = -0.019f; // it should be 20, but with 19 it doesn’t overlap with the back side
static const Real BOOTH_LED_Z = 0.05f;
/**/  
class CBoothEmbodiedEntity : public CEmbodiedEntity {
public:
    CBoothEmbodiedEntity(CBoothEntity* pc_parent) :
        CEmbodiedEntity(pc_parent) 

m_cHalfSize.SetX(BOOTH_WIDTH * 0.5f);
m_cHalfSize.SetY(BOOTH_DEPTH * 0.5f);
m_cHalfSize.SetZ(BOOTH_HEIGHT * 0.5f);
}

protected:

virtual void CalculateBoundingBox() {
    m_cCenterPos = GetPosition();
    m_cCenterPos.SetZ(m_cCenterPos.GetZ() + m_cHalfSize.GetZ());
    m_cOrientationMatrix.SetFromQuaternion(GetOrientation());
    CalculateBoundingBoxFromHalfSize(GetBoundingBox(),
        m_cHalfSize,
        m_cCenterPos,
        m_cOrientationMatrix);
}

private:

CVector3 m_cHalfSize;
CVector3 m_cCenterPos;
CRotationMatrix3 m_cOrientationMatrix;

};

CBoothEntity::CBoothEntity() :
    CComposableEntity(NULL),
    m_pcEmbodiedEntity(new CBoothEmbodiedEntity(this)),
    m_pcControllableEntity(new CControllableEntity(this)),
    m_pcLEDEquippedEntity(new CLedEquippedEntity(this)) {
    m_pcLEDEquippedEntity->AddLed(CVector3());
}

CBoothEntity::~CBoothEntity() {
    delete m_pcEmbodiedEntity;
    delete m_pcControllableEntity;
    delete m_pcLEDEquippedEntity;
}

//********************************************************************************
//********************************************************************************
CBoothEntity::CBoothEntity() :
    CComposableEntity(NULL),
    m_pcEmbodiedEntity(new CBoothEmbodiedEntity(this)),
    m_pcControllableEntity(new CControllableEntity(this)),
    m_pcLEDEquippedEntity(new CLedEquippedEntity(this))
} {
    m_pcLEDEquippedEntity->AddLed(CVector3());
}

//********************************************************************************
//********************************************************************************
CBoothEntity::~CBoothEntity() {
    delete m_pcEmbodiedEntity;
    delete m_pcControllableEntity;
    delete m_pcLEDEquippedEntity;
}

//********************************************************************************
//********************************************************************************

void CBoothEntity::Init(TConfigurationNode& t_tree) {
    try {
        /* Init parent */
        CEntity::Init(t_tree);
        /* Init components */
        m_pcEmbodiedEntity->Init(t_tree);
        m_pcControllableEntity->Init(t_tree);
        m_pcLEDEquippedEntity->Init(t_tree);
        //TODO: check, we set only the leds, we don't need to update
        the other components
        UpdateComponents();
        }
    catch(CARGoSException& ex) {
        THROW_ARGOSEXCEPTION_NESTED("Failed to initialize entity ", ex);
    }
}

void CBoothEntity::Reset() {
    /* Reset all components */
    m_pcEmbodiedEntity->Reset();
    m_pcControllableEntity->Reset();
    m_pcLEDEquippedEntity->Reset();
    /* Update components */
    UpdateComponents();
}

void CBoothEntity::Destroy() {
    m_pcEmbodiedEntity->Destroy();
    m_pcControllableEntity->Destroy();
    m_pcLEDEquippedEntity->Destroy();
}

CEntity& CBoothEntity::GetComponent(const std::string&
str_component) {
    if(str_component == "embodied_entity") {
        return *m_pcEmbodiedEntity;
    } else if(str_component == "controllable_entity") {
        return *m_pcControllableEntity;
    } else if(str_component == "led_equipped_entity") {
        return *m_pcLEDEquippedEntity;
    } else {
        THROW_ARGOSEXCEPTION("A booth does not have a component of type \\
        \\
        type " << str_component << \\
        \\
        "");
    }
}

bool CBoothEntity::HasComponent(const std::string& str_component) {
    return (str_component == "embodied_entity" ||
            str_component == "controllable_entity" ||
            str_component == "led_equipped_entity" );
}

void CBoothEntity::UpdateComponents() {
    SetLedPosition();
    //TODO: are these updates necessary since the booth doesn't move?
    m_pcEmbodiedEntity->UpdateBoundingBox();
}

#define SET_RING_LED_POSITION(IDX)
    cLEDPosition.Set(BOOTH_LED_X, BOOTH_LED_Y, BOOTH_LED_Z );
    cLEDPosition.Rotate(m_pcEmbodiedEntity->GetOrientation());
    cLEDPosition += cEntityPosition;
m_pcLEDEquippedEntity->SetLedPosition(IDX, cLEDPosition);

void CBoothEntity::SetLedPosition() {
    /* Set LED positions */
    const CVector3& cEntityPosition =
        GetEmbodiedEntity().GetPosition();
    CVector3 cLEDPosition;
    CRadians cLEDAngle;
    SET_RING_LED_POSITION(0);
}

/*******************************************************************************/
/*******************************************************************************/
REGISTER_ENTITY(CBoothEntity,
    "booth",
    "The booth entity.",
    "Manuele Brambilla [mbrambilla@iridia.ulb.ac.be]",
    "The booth is a simple box that can interact with
an e-puck."
    "REQUIRED XML CONFIGURATION"
    " <arena ...>
    " ..."
    " <booth id="bt0" position="0.4,2.3,0.25" orientation="45,90,0"
    " controller="mycntrl" />
    " ..."
    " </arena>"
    "The 'id' attribute is necessary and must be unique among the entities. If two aborts."
    "The 'position' attribute specifies the position of the bottom point of the booth in the arena. When the booth is untranslated and unrotated, the pucktom point is in the origin and it is defined as the middle point between the two walls on the XY plane and the lowest point of the booth on the Z axis, that is the point where the walls touch the floor. The attribute values are in the X,Y,Z order.
    "The 'orientation' attribute specifies the orientation of the booth. All"
rotations are performed with respect to the bottom point. The order of the angles is Z,Y,X, which means that the first number corresponds to the rotation around the Z axis, the second around Y and the last around X. This reflects the internal convention used in ARGoS, in which rotations are performed in that order. Angles are expressed in degrees. When the booth is unrotated, it is oriented along the X axis.

The 'controller' attribute is used to assign a controller to the booth. The value of the attribute must be set to the id of a previously defined controller. Controllers are defined in the <controllers> XML subtree.

Optional XML configuration

None for the time being.
Under development.
A.2 Real World

A.2.1 Hardware

Bill-of-materials

Bill-of-materials of the main board: Figure A.1
Bill-of-materials of the side board (right): Figure A.2
Bill-of-materials of the side board (left): Figure A.3

Schematics

Circuit schematics of the main board (Power supply): Figure A.4
Circuit schematics of the main board (Side board connectors): Figure A.5
Circuit schematics of the main board (LED controller): Figure A.6
Circuit schematics of the main board (Arduino base circuit): Figure A.7
Circuit schematics of the main board (Xbee module): Figure A.8
Circuit schematics of the side board (right): Figure A.9
Circuit schematics of the side board (left): Figure A.10

Circuit Boards

Prototype circuit board layout of the main board: Figure A.11
Prototype circuit board layout of the side board (right): Figure A.12
Prototype circuit board layout of the side board (left): Figure A.13
<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Desc</th>
<th>Package</th>
<th>Manufacturer</th>
<th>Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Figure A.1: Bill-of-materials of the main board.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Desc</th>
<th>Package</th>
<th>Manufacturer</th>
<th>Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>C9</td>
<td></td>
<td>CAP CER 100PF 16V 10% X7R 0603</td>
<td>TDK Corporation</td>
<td>1</td>
</tr>
<tr>
<td>C7</td>
<td></td>
<td>CAP CER 22UF 6.3V 10% X5R 1206</td>
<td>Avx Corporation</td>
<td>1</td>
</tr>
<tr>
<td>C6</td>
<td></td>
<td>CAP CER 47PF 50V 5% NP0 0603</td>
<td>TDK Corporation</td>
<td>1</td>
</tr>
<tr>
<td>C5</td>
<td></td>
<td>CAP CER 220PF 50V 5% NP0 0603</td>
<td>TDK Corporation</td>
<td>1</td>
</tr>
<tr>
<td>C1</td>
<td></td>
<td>CAP CER 10UF 10V 10% X5R 0805</td>
<td>Murata Electronics</td>
<td>1</td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td>CAP CER 0.1UF 16V 10% X7R 0603</td>
<td>Taiyo Yuden</td>
<td>1</td>
</tr>
<tr>
<td>C10</td>
<td></td>
<td>RES 68 OHM 1/10W 5% 0603 SMD</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>C11</td>
<td></td>
<td>RES 470 OHM 1/10W 5% 0603 SMD</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>C13</td>
<td></td>
<td>RES 39 OHM 1/10W 5% 0603 SMD</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>C14</td>
<td></td>
<td>RES 20K OHM 1/10W 1% 0603 SMD</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>C15</td>
<td></td>
<td>RES 47k OHM 1/10W 1% 0603 SMD</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>C16</td>
<td></td>
<td>RES 10k OHM 1/10W 5% 0603 SMD</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>C17</td>
<td></td>
<td>RES 0.0 OHM 1/10W 5% 0603 SMD</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>C18</td>
<td></td>
<td>RES 221k OHM 1/10W 1% 0603 SMD</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>R9</td>
<td></td>
<td>LED MULTILED RGB WHT BINNED 6PLC</td>
<td>Lite-On Inc</td>
<td>1</td>
</tr>
<tr>
<td>R26</td>
<td></td>
<td>LED RED ORAN CLEAR THIN 0805 SMD</td>
<td>Lite-On Inc</td>
<td>1</td>
</tr>
<tr>
<td>R27</td>
<td></td>
<td>LED AMBER CLR THIN 0805 SMD</td>
<td>Lite-On Inc</td>
<td>1</td>
</tr>
<tr>
<td>R28</td>
<td></td>
<td>LED CHIPLED 570NM GREEN 1206 SMD</td>
<td>Osram Opto Semiconductors</td>
<td>1</td>
</tr>
<tr>
<td>R18</td>
<td></td>
<td>INDUCTOR MULTILAYER 10UH 0805</td>
<td>C&amp;K Components</td>
<td>1</td>
</tr>
<tr>
<td>R21</td>
<td></td>
<td>INDUCTOR POWER 4.7UH 1.3A SMD</td>
<td>TDK Corporation</td>
<td>1</td>
</tr>
<tr>
<td>R15</td>
<td></td>
<td>SWITCH TACTILE SPST-NO 0.05A 12V</td>
<td>Keystone Electronics</td>
<td>1</td>
</tr>
<tr>
<td>R14</td>
<td></td>
<td>do not populate</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>R12</td>
<td></td>
<td>do not populate</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>R13</td>
<td></td>
<td>do not populate</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>R19</td>
<td></td>
<td>do not populate</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>R20</td>
<td></td>
<td>do not populate</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>R24</td>
<td></td>
<td>do not populate</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>R25</td>
<td></td>
<td>do not populate</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>R10</td>
<td></td>
<td>do not populate</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>R11</td>
<td></td>
<td>do not populate</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>R7</td>
<td></td>
<td>RES 6.49k OHM 1/10W 1% 0603 SMD</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>R6</td>
<td></td>
<td>RES 49.9k OHM 1/10W 1% 0603 SMD</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>R5</td>
<td></td>
<td>RES 100K OHM 1/10W 1% 0603 SMD</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>R4</td>
<td></td>
<td>RES 1M OHM 1/10W 1% 0603 SMD</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>R3</td>
<td></td>
<td>RES 42.2k OHM 1/10W 1% 0603 SMD</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>R8</td>
<td></td>
<td>RES 4200000 OHM 1/10W 1% 0603 SMD</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>R16</td>
<td></td>
<td>RES 42.2k OHM 1/10W 1% 0603 SMD</td>
<td>Open/any</td>
<td>1</td>
</tr>
<tr>
<td>R1</td>
<td></td>
<td>LED LED 1206 SMD</td>
<td>C&amp;K Components</td>
<td>1</td>
</tr>
<tr>
<td>LED5</td>
<td></td>
<td>LED LED 1206 SMD</td>
<td>C&amp;K Components</td>
<td>1</td>
</tr>
<tr>
<td>LED4</td>
<td></td>
<td>LED LED 1206 SMD</td>
<td>C&amp;K Components</td>
<td>1</td>
</tr>
<tr>
<td>LED3</td>
<td></td>
<td>LED LED 1206 SMD</td>
<td>C&amp;K Components</td>
<td>1</td>
</tr>
<tr>
<td>LED2</td>
<td></td>
<td>LED LED 1206 SMD</td>
<td>C&amp;K Components</td>
<td>1</td>
</tr>
<tr>
<td>LED1</td>
<td></td>
<td>LED LED 1206 SMD</td>
<td>C&amp;K Components</td>
<td>1</td>
</tr>
<tr>
<td>L2</td>
<td></td>
<td>LED LED 1206 SMD</td>
<td>C&amp;K Components</td>
<td>1</td>
</tr>
<tr>
<td>L1</td>
<td></td>
<td>LED LED 1206 SMD</td>
<td>C&amp;K Components</td>
<td>1</td>
</tr>
<tr>
<td>S2</td>
<td></td>
<td>LED LED 1206 SMD</td>
<td>C&amp;K Components</td>
<td>1</td>
</tr>
<tr>
<td>S1</td>
<td></td>
<td>LED LED 1206 SMD</td>
<td>C&amp;K Components</td>
<td>1</td>
</tr>
<tr>
<td>JP4</td>
<td></td>
<td>LED LED 1206 SMD</td>
<td>C&amp;K Components</td>
<td>1</td>
</tr>
<tr>
<td>JP3</td>
<td></td>
<td>LED LED 1206 SMD</td>
<td>C&amp;K Components</td>
<td>1</td>
</tr>
<tr>
<td>JP2</td>
<td></td>
<td>LED LED 1206 SMD</td>
<td>C&amp;K Components</td>
<td>1</td>
</tr>
<tr>
<td>JP1B</td>
<td></td>
<td>LED LED 1206 SMD</td>
<td>C&amp;K Components</td>
<td>1</td>
</tr>
<tr>
<td>U7</td>
<td></td>
<td>IC LED DRIVER LINEAR 28-TSSOP</td>
<td>Atmel</td>
<td>1</td>
</tr>
<tr>
<td>U6</td>
<td></td>
<td>IC REG BUCK BST SYNC ADJ 12MSOP</td>
<td>Texas Instruments</td>
<td>1</td>
</tr>
<tr>
<td>U5</td>
<td></td>
<td>IC OPAMP GP 700KHZ DUAL 8TSSOP</td>
<td>Texas Instruments</td>
<td>1</td>
</tr>
<tr>
<td>U4</td>
<td></td>
<td>IC CTLR ON/OFF W/DEB TSOT23-6</td>
<td>Maxim Integrated</td>
<td>1</td>
</tr>
<tr>
<td>U3</td>
<td></td>
<td>TRANSISTOR GP NPN AMP SOT-23</td>
<td>Maxim Integrated</td>
<td>1</td>
</tr>
<tr>
<td>U2</td>
<td></td>
<td>CERAMIC RESONATOR 16.0MHZ SMD</td>
<td>TDK Corporation</td>
<td>1</td>
</tr>
</tbody>
</table>
Figure A.2: Bill-of-materials of the side board (right).
<table>
<thead>
<tr>
<th>Name</th>
<th>Desc</th>
<th>Value</th>
<th>Package</th>
<th>Manufacturer</th>
<th>Manufacturer part number</th>
<th>Qty</th>
<th>Example supplier</th>
<th>Example supplier reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR_LED_L</td>
<td>IR EMITTER</td>
<td>SFH 4258</td>
<td>4-PLCC</td>
<td>OSRAM Opto Semiconductors</td>
<td>SFH4258-Z</td>
<td>1</td>
<td>Digi-Key</td>
<td>475-1243-1-ND</td>
</tr>
<tr>
<td>IR_TRANSISTOR_L</td>
<td>IR PHOTOTRANSISTOR</td>
<td>SFH 320 FA-Z</td>
<td>PLCC-2</td>
<td>OSRAM Opto Semiconductors</td>
<td>SFH320FA-Z</td>
<td>1</td>
<td>Mouser</td>
<td>720-SFH320FA-Z</td>
</tr>
<tr>
<td>PROX_L</td>
<td>do not populate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure A.3: Bill-of-materials of the side board (left).
Figure A.4: Circuit schematics of the main board (Power supply).
Figure A.5: Circuit schematics of the main board (Side board connectors).
Figure A.6: Circuit schematics of the main board (LED controller).
Figure A.7: Circuit schematics of the main board (Arduino base circuit).
Figure A.8: Circuit schematics of the main board (Xbee module).
Figure A.9: Circuit schematics of the side board (right).
Figure A.10: Circuit schematics of the side board (left).
Figure A.11: Prototype circuit board layout of the main board.
Figure A.12: Prototype circuit board layout of the side board (right).

Figure A.13: Prototype circuit board layout of the side board (left).
A.2.2 Firmware

The firmware below is an example for a firmware that detects the presence of an e-puck and reports this back to the central node.

```c
// IRcom library
#include <ircom.h>
#include <ircomReceive.h>
#include <ircomMessages.h>
#include <ircomTools.h>
#include <ircomSend.h>

// all libs below are included in the TAM git
#include <I2C.h>
#include <TLC59116.h>  // based on http://brmlab.cz/project/ledbar
#include <XBee.h>      // based on http://code.google.com/p/xbee-arduino/ (v0.4b)

/**************************************************************************/
* 
* Example Arduino Controller for the IRIDIA TAM 
* 
/**************************************************************************/
#define TAM_FIRMWARE_VERSION "v5rD-0.2"

/**
 * Debug macros which enable bits of code used for debugging the firmware.
 */
#define DEBUG_ENABLE_SERIAL_OUTPUT  // disabled all serial printout and strings
#define ENABLE_IRCOM
#define DEBUG_TAM_INFO
#define DEBUG_XBEE_INFO

/**
 * Controls which messages are logged.
 */
// general output classes
#define P_INFO(format, ...) p("[INFO] %04d " format, cycle, ##
```
__VA_ARGS__
#define P_INFO(format, ...) p( "[WARN] %04d " format, cycle, __VA_ARGS__ )
#define P_WARN(format, ...) p( "[WARN] %04d " format, cycle, __VA_ARGS__ )
#define P_XBEE_RECV(format, ...) p( "[XBRX] %04d " format, cycle, __VA_ARGS__ )
#define P_XBEE_AT(format, ...) p( "[XBAT] %04d " format, cycle, __VA_ARGS__ )
#define P_LIBA(format, ...) p( "[LIBA] %04d " format, cycle, __VA_ARGS__ )
#define P_PROX(format, ...) p( "[PROX] %04d " format, cycle, __VA_ARGS__ )
#define P_IRCOM(format, ...) p( "[IRCOM] %04d " format, cycle, __VA_ARGS__ )

/**
 * Definition of packet types used in Xbee communication.
 * Packet type MUST be designated in first byte.
 */

#define PACKET_TYPE_TC_CURRENT_STATE 0 // state report
#define PACKET_TYPE_TC_READ_ROBOT_RESULT 1 // result of read robot operation
#define PACKET_TYPE_TC_WRITE_ROBOT_RESULT 2 // result of write robot operation

// coordinator to TAM
#define PACKET_TYPE_CT_SET_LEDS 10
#define PACKET_TYPE_CT_READ_ROBOT 11
#define PACKET_TYPE_CT_WRITE_ROBOT 12
#define PACKET_TYPE_CT_SHUTDOWN 13

/**
 * Definition of constants (thresholds, intervals and times).
 * Adjust only if you know what you are doing, ESPECIALLY the network
 * stuff!
 */
// delay between switching the light barrier on and off
#define IR_WAIT_TIME 2
// threshold required between IR LED on and off in the reading of the
IR transistor
#define IR_THRESHOLD 120

// threshold for low battery voltage detection
#define BATT_VOLTAGE_WARN 3000
// threshold for low battery voltage detection
#define BATT_VOLTAGE_SHUTDOWN 2700
// number of cycles the battery has to be below BATT_VOLTAGE_SHUTDOWN
// before we shut down
// used to filter battery voltage sensing
#define BATT_UNDER_VOLTAGE_COUNTER_THRESHOLD 10

// if defined, the TAM tries to resolve the coordinator via network
on boot
// if undefined, uses hard-coded coordinator address defined below
#define RESOLVE_COORDINATOR
// maximum number of resolve commands, if it fails it uses the
// hard-coded
// coordinator address defined below
#define RESOLVE_COORDINATOR_MAX_TRIES 500

// minimum length of main control cycle in ms
// cycle might be slightly longer due to function calls
#define CYCLE_LENGTH 10
// interval at which we send our status to the coordinator, expressed
in milliseconds
#define HEARTBEAT_INTERVAL 10000
// timeout for waiting for responses for the packets sent by us,
expressed in milliseconds
#define RESPONSE_TIMEOUT 1000

// interval at which we send the ircom data while the robot is inside the TAM, expressed in ms
#define IRCOM_SEND_INTERVAL 200

/**
 * Definition of hardware pins. Do NOT change this.
*/
#define XBEE_WAKE_PIN 12 // xbee sleep control
#define XBEE_CTS_PIN 13 // xbee serial cts (unused at the moment, use for large data)
#define XBEE_RTS_PIN 18 // xbee serial rts (unused at the moment, use for large data)
#define XBEE_RST_PIN 15 // xbee reset
#define XBEE_NET_ID_0_PIN 2 // bit 0 of xbee PAN id
#define XBEE_NET_ID_1_PIN 4 // bit 1 of xbee PAN id
#define NETWORK_LED_PIN 19
#define POWER_OFF_PIN 3 // all-mighty power-off pin for PSU
#define VBATT_SENSE_PIN A1 // signal VBATT_SENSE
#define PROX_SENSOR_PIN 14 // ICP pin fixed to internal Timer
#define PROX_EMIT_PIN 1
#define IRB_SENSOR_PIN A2 // signal IR_SENS_L
#define IRF_SENSOR_PIN A0 // signal IR_SENS_R

enum TlcLedPins {
    LED_IR_R,
    LED_RED_R,
    LED_BLUE_R, // right side leds (IR + RGB)
    LED_GREEN_R,
    LED_RED_M,
    LED_BLUE_M,
    LED_GREEN_M, // middle led (RGB)
    LED_RED_L,
    LED_BLUE_L, // left side leds (IR + RGB)
    LED_GREEN_L,
    LED_IR_L,
    LED_BO, // unused (breakout 0)
LED_B1,  // unused (breakout 1)
LED_BLUE_T,
LED_GREEN_T,
LED_RED_T  // top led (RGB)
};

// adapt this however your board is populated
const uint8_t NUM_LEDS = 4;
const uint8_t LEDS_RED[NUM_LEDS] = {
    LED_RED_R,
    LED_RED_M,
    LED_RED_L,
    LED_RED_T
};
const uint8_t LEDS_GREEN[NUM_LEDS] = {
    LED_GREEN_R,
    LED_GREEN_M,
    LED_GREEN_L,
    LED_GREEN_T
};
const uint8_t LEDS_BLUE[NUM_LEDS] = {
    LED_BLUE_R,
    LED_BLUE_M,
    LED_BLUE_L,
    LED_BLUE_T
};

// type definition for the state of the TAM
typedef struct {
    uint8_t redLed;
    uint8_t greenLed;
    uint8_t blueLed;
    boolean robotPresent;
    uint8_t robotData;
} state_t;

/**
 * Global variables
 */

// current state (RGB LED value and robot presence)
state_t currentState;
// state as reported last to coordinator via Xbee
state_t lastReportedState;
// if true, the TAM reports the state to the coordinator in the next

51
cycle
boolean reportState = false;

// main objects initialization
// if you don’t to this here the bloody TAM resets
//
TLC59116 tlc = TLC59116(); // TLC that controls the PWM leds
XBee xbee = XBee();  // Xbee library

// Variables for reading light barriers analog
int irReadPreF;
int irReadPreB;
// stages of checking light barriers
byte irStage = 0;
// this can be considered as 8 booleans, used to filter the IR light
barrier reading
uint8_t irBoolsF = 0;
uint8_t irBoolsB = 0;
uint8_t irBit = 1;

// voltage check variables
int battVoltage = 0;
uint8_t battUnderVoltageCounter = 0;

// Xbee variables
uint32_t xbeeCoordinatorAddressHigh; // 64bit coordinator address, HSBs
uint32_t xbeeCoordinatorAddressLow; // 64bit coordinator address, LSBs
// fallback coordinator address, MSB and LSB
const uint32_t xbeeHardcodedCoordinatorAddressesHigh = 0x0013A200;
const uint32_t xbeeHardcodedCoordinatorAddressesLow[4] = {
0x408C04A4, 0x408C04C5, 0x408D6EB4, 0x408D6EA2
};
XBeeAddress64 coordAddr64; // struct for address
DMTxRequest statePacket; // packet for sending TAM state
DMTxStatusResponse statePacketResponse; // response for sending TAM state packet
uint16_t nodeDiscoveryTimeout;

// variables used for AT requests (management commands for Xbee)
uint8_t atCmd[2];
AtCommandRequest atRequest;
AtCommandResponse atResponse;
// count of control cycles, each cycle = 0.1 secs
int cycle = 0;

// set to true if the last packet sent is still awaiting response
// used for getting packet ACKs asynchronously
boolean lastPacketAwaitingResponse = false;

// frame id of the last packet sent
uint8_t lastPacketFrameId = 0;

// timestamp of the last packet sent
unsigned long lastPacketSentAt = 0;

// timestamp of last heartbeat
unsigned long lastHeartbeatSentAt = 0;

// payload of the state packet
uint8_t statePayload[8];

// IrCom variables
#ifdef ENABLE_IRCOM
int irComDataToWrite = 0;
unsigned long IrComLastSendAt = 0;
#endif

// current time
unsigned long now = 0;

/*********************************************************************
******************************/
/*********************************************************************
******************************/

/**
 * Small helper function that emulates printf().
 * Disabled by DEBUG_ENABLE_SERIAL_OUTPUT.
 * @param fmt format string, followed by possible parameters
 */
void p(const char *fmt, ... )
{
#ifdef DEBUG_ENABLE_SERIAL_OUTPUT
    char tmp[128]; // resulting string limited to 128 chars
    va_list args;
    va_start(args, fmt );
    vsnprintf(tmp, 128, fmt, args);
    va_end (args);
    Serial.print(tmp);
#endif
}
```c
/**
 * Swaps the endianness a 16bit unsigned integer.
 * @param val pointer to integer to swap
 */
void swapEndian16(uint16_t &val)
{
    val = (val<<8) | (val>>8);
}

/**
 * Swaps the endianness a 32bit unsigned integer.
 * @param val pointer to integer to swap
 */
void swapEndian32(uint32_t &val)
{
    val = (val<<24) | ((val<<8) & 0x00ff0000) | ((val>>8) & 0x0000ff00)
        | (val>>24);
}

/**
 * Prepare serial and set necessary digital pins to output.
 * Also initializes the Xbee module and resolves the address of the
 * coordinator.
 */
void setup()
{
    Serial.begin(57600);
    Serial.println("Starting IRIDIA TAM");

    #ifdef DEBUG_TAM_INFO
    p(\"Firmware version: %s\n\r", TAM_FIRMWARE_VERSION);
    #endif

    // currently unused
    pinMode(XBEE_WAKE_PIN, OUTPUT);
    digitalWrite(XBEE_WAKE_PIN, LOW);
    pinMode(XBEE_CTS_PIN, OUTPUT);
    digitalWrite(XBEE_CTS_PIN, LOW);
    pinMode(XBEE_RTS_PIN, OUTPUT);
```
digitalWrite(XBEE_RTS_PIN, LOW);

// sensing for front and back IR barrier
pinMode(IRF_SENSE_PIN, INPUT);
pinMode(IRB_SENSE_PIN, INPUT);

// setup psu pins
pinMode(VBATT_SENSE_PIN, INPUT);
pinMode(POWER_OFF_PIN, OUTPUT);
digitalWrite(POWER_OFF_PIN, LOW);

// emit and sensing for proximity sensor
pinMode(PROX_EMIT_PIN, OUTPUT);
digitalWrite(PROX_EMIT_PIN, LOW);
pinMode(PROX_SENSE_PIN, INPUT);

//IRcom initialization and start
#ifdef ENABLE_IRCOM
    ircomStart();
e_init_sampling();
    ircomEnableContinuousListening();
    ircomListen();
#endif

// configure all pins of TLC to be used as PWM pins
tlc.begin(B1100000, true, false);
for (int i = 0; i < NUM_LEDS; i++)
{
    tlc.setPinMode(LED.SeleniumRed[i], PM_PWM);
    tlc.setPinMode(LED.SeleniumGreen[i], PM_PWM);
    tlc.setPinMode(LED.SeleniumBlue[i], PM_PWM);
}

// breakout pins currently not used
tlc.setPinMode(LED_BO, PM_OFF);
tlc.setPinMode(LED_B1, PM_OFF);

// set IR leds to full power but currently disabled
tlc.setPinMode(LED_IR_R, PM_OFF);
tlc.setPinMode(LED_IR_L, PM_OFF);
tlc.setPinPWM(LED_IR_R, 55);
tlc.setPinPWM(LED_IR_L, 55);

// XBee pan id selector
pinMode(XBEE_NET_ID_0_PIN, INPUT);
digitalWrite(XBEE_NET_ID_0_PIN, HIGH); // enable internal pullup
pinMode(XBEE_NET_ID_1_PIN, INPUT);
digitalWrite(XBEE_NET_ID_1_PIN, HIGH); // enable internal pullup

// Xbee initialization
//
p("Initializing network");
pinMode(NETWORK_LED_PIN, OUTPUT);
digitalWrite(NETWORK_LED_PIN, HIGH);

// start the xbee
Serial1.begin(9600); // xbee serial
xbee.begin(Serial1);

P_XBEE("\nResetting XBee, please wait.\n");
pinMode(XBEE_RST_PIN, OUTPUT);
digitalWrite(XBEE_RST_PIN, HIGH);
delay(120);
digitalWrite(XBEE_RST_PIN, LOW);

P_XBEE("Waiting for XBee to become ready.");
// xbee will send MODEM_STATUS frame (0x8a) on power-up
do {
    xbee.readPacket();
delay(100);
p(".");
} while (!xbee.getResponse().isAvailable() &&
xbee.getResponse().getApiId() != MODEM_STATUS_RESPONSE);

P_XBEE("OK\n");

// setup AT requests
atRequest = AtCommandRequest(atCmd);
atResponse = AtCommandResponse();

// set and read node discover timeout
// we do this in a loop because the module sometimes isn't there yet
P_XBEE("Set/read node discover timeout.\n");
uint8_t ntValue = 0x82; // default 0x82
atCmd[0] = 'N';
atCmd[1] = 'T';
atRequest.setCommandValue(&ntValue);
atRequest.setCommandValueLength(1);
atRequest.setCommand(atCmd);
uint16_t nodeDiscoveryTimeout = 0;
do {
    sendAtCommand(100, &ntValue);
nodeDiscoveryTimeout = ntValue * 100;
p(".");
} while (nodeDiscoveryTimeout == 0);
P_XBEE("OK\n")

P_XBEE("Set/read PAN ID.\n")
uint8_t coordinatorNumber = (digitalRead(XBEE_NET_ID_1_PIN) << 1) +
digitalRead(XBEE_NET_ID_0_PIN);
uint16_t panId = 0x01b2 + coordinatorNumber;
swapEndian16(panId);
atCmd[0] = 'I';
atCmd[1] = 'D';
atRequest.setCommandValue((uint8_t*)&panId);
atRequest.setCommandValueLength(2);
atRequest.setCommand(atCmd);
sendAtCommand(50, (uint8_t*)&panId);
swapEndian16(panId);
P_XBEE("OK\n")

#ifdef RESOLVE_COORDINATOR
// The code below resolves the coordinator addresss by a DN command.
uint8_t coordinatorId[5] = {
   'C', 'O', 'O', 'R', 'D'
};
xBeeCoordinatorAddressHigh = 0;
xBeeCoordinatorAddressLow = 0;
atCmd[0] = 'D';
atCmd[1] = 'N';
atRequest.setCommand(atCmd);
atRequest.setCommandValue(coordinatorId);
atRequest.setCommandValueLength(5);
uint8_t tries = 0;
boolean networkLed = true;
boolean sendRequest = true;
unsigned long lastPacketAt = 0;
P_XBEE("Resolving XBee coordinator address. (this might take a few
seconds)")

  do {
    if (sendRequest) {
      P_XBEE_AT("Sending AT command %c%c to the XBee\n\r",
atRequest.getCommand()[0], atRequest.getCommand()[1]);
      xbee.send(atRequest);
      sendRequest = false;
lastPacketAt = millis();

// check for timeout
if (millis() - lastPacketAt >= nodeDiscoveryTimeout) {
    sendRequest = true;
    P_XBEE_AT("Answer timeout!\n\r");
}

// check if we received a new packet
xbee.readPacket();
if (xbee.getResponse().isAvailable()) {
    if (xbee.getResponse().getApiId() == AT_COMMAND_RESPONSE)
    {
        xbee.getResponse().getAtCommandResponse(atResponse);
        if (atResponse.getCommand()[0] == 'D' &&
            atResponse.getCommand()[1] == 'N')
        {
            sendRequest = true;
        }
    }
}

networkLed = !networkLed;
digitalWrite(NETWORK_LED_PIN, networkLed);
delay(500);
p(".");
tries++;
} while (tries < RESOLVE_COORDINATOR_MAX_TRIES &&
(atResponse.getStatus() != 0x40 || atResponse.getValueLength() != 10));

if (atResponse.getStatus() == 0x40 && atResponse.getValueLength() == 10)
{
    memcpy(&xbeeCoordinatorAddressHigh, atResponse.getValue()+2, 4);
    memcpy(&xbeeCoordinatorAddressLow, atResponse.getValue()+6, 4);

    // swap endianess of addresses
    swapEndian32(xbeeCoordinatorAddressHigh);
    swapEndian32(xbeeCoordinatorAddressLow);

    P_XBEE("SUCCESS.\r\n");
} else
{

// use the hardcoded coordinator address as fall-back
if (xbeeCoordinatorAddressHigh == 0 || xbeeCoordinatorAddressLow == 0)
{
  P_XBEE("Using hard-coded coordinator address.
        ");
  xbeeCoordinatorAddressHigh =
  xbeeHardcodedCoordinatorAddressesHigh;
  xbeeCoordinatorAddressLow =
  xbeeHardcodedCoordinatorAddressesLow[coordinatorNumber];
}
digitalWrite(NETWORK_LED_PIN, LOW);

// clear state and setup packets to coordinator
memset(&currentState, sizeof(state_t), 0);
memset(&lastReportedState, sizeof(state_t), 0);
coordAddr64 = XBeeAddress64(xbeeCoordinatorAddressHigh,
  xbeeCoordinatorAddressLow);
statePacket = DMTxRequest(coordAddr64, statePayload,
  sizeof(statePayload));
statePacketResponse = DMTxStatusResponse();
atRequest.setCommandValueLength(0);
atRequest.setCommandValue(NULL);
p("OK
");

#ifdef DEBUG_XBEE_INFO
p("Xbee variables:
        ");
p("Discovery time: 0x%4d\n", nodeDiscoveryTimeout);

uint8_t myName[26];
memset(myName, 0, 26);
atCmd[0] = 'N';
atCmd[1] = 'I';
atRequest.setCommand(atCmd);
sendAtCommand(50, myName);
myName[25] = 0;
p("TAM network ID: %s\n", myName);

uint32_t highAddress = 0;
atCmd[0] = 'S';
atCmd[1] = 'H';
atRequest.setCommand(atCmd);
sendAtCommand(50, (uint8_t*)&highAddress);
swapEndian32(highAddress);
p(" High address : 0x%08lx\n\r", highAddress);

uint32_t lowAddress = 0;
atCmd[0] = 'S';
atCmd[1] = 'L';
atRequest.setCommand(atCmd);
sendAtCommand(50, (uint8_t*)&lowAddress);
swapEndian32(lowAddress);
p(" Low address : 0x%08lx\n\r", lowAddress);

p(" PAN ID : 0x%04x\n\r", panId);
uint8_t channel = 0;
atCmd[0] = 'C';
atCmd[1] = 'H';
atRequest.setCommand(atCmd);
sendAtCommand(50, &channel);
p(" Channel : 0x%02x\n\r", channel);

// coordinator address
p(" Coordinator h.: 0x%08lx\n\r", xbeeCoordinatorAddressHigh);
p(" Coordinator l.: 0x%08lx\n\r", xbeeCoordinatorAddressLow);
}

int8_t sendAtCommand(int timeout, uint8_t* buffer)
{
  P_XBEE_AT("Sending AT command %c%c to the XBee\n\r", atRequest.getCommand()[0], atRequest.getCommand()[1]);
// send the command
xbee.send(atRequest);

unsigned long start = millis();
do {
    xbee.readPacket();
    if (xbee.getResponse().isAvailable()) {
        if (xbee.getResponse().getApiId() == AT_COMMAND_RESPONSE)
            xbee.getResponse().getAtCommandResponse(atResponse);
        //printResponseFrameData(xbee.getResponse());
        if (atResponse.getCommand()[0] == atRequest.getCommand()[0] &&
            atResponse.getCommand()[1] == atRequest.getCommand()[1])
            P_XBEE_AT("Got response for correct AT Command %c%c\n\r", atResponse.getCommand()[0], atResponse.getCommand()[1]);
        if (atResponse.isOk())
            P_XBEE_AT("AT command %c%c was successful!\n\r", atResponse.getCommand()[0], atResponse.getCommand()[1]);
        if (atResponse.getValueLength() > 0)
            P_XBEE_AT("AT command %c%c value length is %d\n\r", atResponse.getCommand()[0], atResponse.getCommand()[1],
                atResponse.getValueLength());
        if (buffer != NULL)
            memcpy(buffer, atResponse.getValue(), atResponse.getValueLength());
    }
    return 0;
} else {
    P_XBEE_AT("AT command %c%c return error code: 0x%02x\n\r", atResponse.getCommand()[0], atResponse.getCommand()[1],
                atResponse.getStatus());
    if (atResponse.getValueLength() > 0)
        P_XBEE_AT("AT command %c%c value length is %d\n\r", atResponse.getCommand()[0], atResponse.getCommand()[1],
                atResponse.getValueLength());
return atResponse.getStatus();
}
else
{
P_XBEE_AT("Got response for wrong AT Command %c%c, ignoring
\n\r", atResponse.getCommand()[0], atResponse.getCommand()[1]);
}
}
else
{
P_XBEE_AT("Expected AT response but got 0x%02x
\n\r", xbee.getResponse().getApiId());
}
} while ((millis() - start) < timeout);

P_XBEE_AT("No response from radio in %d ms\n\r", timeout);
return 255;

/**
 * Shutdown by switching of the PSU.
 * This is irreversible by software.
 * Called when we detect a low battery.
 */
void shutdown()
{
P_WARN("Shutting down now.\n");
delay(500);
digitalWrite(POWER_OFF_PIN, HIGH);
delay(500);
}

/**
 * Check for battery voltage and shut down the TAM
 * in case it’s too low for a couple of steps in a row.
 */
void checkBatteryVoltage()
{
battVoltage = analogRead(VBATT_SENSE_PIN) * 33;
// p("[DEBUG] Battery voltage: %d.%03d V\n", (battVoltage / 1000),
(battVoltage % 1000));

if (battVoltage <= BATT_VOLTAGE_SHUTDOWN) {
    P_WARN("Battery empty!\n");
    battUnderVoltageCounter++;

    if (battUnderVoltageCounter ==
BATT_UNDER_VOLTAGE_COUNTER_THRESHOLD)
    {
        P_WARN("Battery empty, shutting down!\n");
        delay(1000);
        shutdown();
    }
}
else if (battVoltage <= BATT_VOLTAGE_WARN) {
    P_WARN("Battery low!\n");
}
else {
    battUnderVoltageCounter = 0;
}
}

/**
 * Sets the RGB LEDs (all colors in a single function call).
 * TODO: maybe use the group feature of TLC somehow for these groups
 */
void setRgbLeds(uint8_t redLed, uint8_t greenLed, uint8_t blueLed)
{
// reduce brightness of reds slightly as red leds is different from
blue and green
    tlc.setPinPWM(LED_RED_L, redLed / 1.5);
    tlc.setPinPWM(LED_RED_M, redLed / 1.5);
    tlc.setPinPWM(LED_RED_R, redLed / 1.5);
    tlc.setPinPWM(LED_RED_T, redLed);
    currentState.redLed = redLed;

    tlc.setPinPWM(LED_GREEN_L, greenLed);
    tlc.setPinPWM(LED_GREEN_M, greenLed);
    tlc.setPinPWM(LED_GREEN_R, greenLed);
    tlc.setPinPWM(LED_GREEN_T, greenLed);
    currentState.greenLed = greenLed;

    tlc.setPinPWM(LED_BLUE_L, blueLed);
    tlc.setPinPWM(LED_BLUE_M, blueLed);
}
tlc.setPinPWM(LED_BLUE_R, blueLed);
tlc.setPinPWM(LED_BLUE_T, blueLed);
currentState.blueLed = blueLed;
}

/**
 * Checks if the light barrier is blocked by using a ANALOG read.
 * Returns true if yes.
 */
void checkLightBarriersAnalog()
{
    irStage++;

    // read value ambient light
    if (irStage == 1)
    {
        // read analog before we turn LED on
        irReadPreF = analogRead(IRF_SENSE_PIN);
        irReadPreB = analogRead(IRB_SENSE_PIN);
    }
    // turn IR LEDs on
    else if (irStage == 2)
    {
        tlc.setPinMode(LED_IR_R, PM_PWM);
        tlc.setPinMode(LED_IR_L, PM_PWM);
    }
    // read value when IR on
    else if (irStage == 3)
    {
        int irReadPostF = analogRead(IRF_SENSE_PIN);
        int irReadPostB = analogRead(IRB_SENSE_PIN);

        boolean frontIrBarrierBlocked = (abs(irReadPostF - irReadPreF) <
            IR_THRESHOLD);
        boolean backIrBarrierBlocked = (abs(irReadPostB - irReadPreB) <
            IR_THRESHOLD);

        /*
         * P_LIBA("Front IR barrier: pre=%d post=%d difference=%d
         * threshold=%d\n\r",
         *     irReadPreF, irReadPostF, abs(irReadPostF - irReadPreF),
         *     IR_THRESHOLD);
         * P_LIBA("Back IR barrier : pre=%d post=%d difference=%d
         * threshold=%d\n\r",
         */
irReadPreB, irReadPostB, abs(irReadPostB - irReadPreB),
IR_THRESHOLD);

/*

// Check if we detected the presence of a robot that is fully in
the TAM.
// For a robot to be fully in the TAM, the front light barrier
needs to be free
// and the back light barrier needs to be blocked (back = further
inside).
// The code below sets bits in a byte to the according reading.
There are
// 8 bits, thus, it takes 8 identical readings to switch state.
// This is done to filter the sensor reading.
if (frontIrBarrierBlocked)
{
    irBoolsF |= irBit;
}
else
{
    irBoolsF &= ~irBit;
}
if (backIrBarrierBlocked)
{
    irBoolsB |= irBit;
}
else
{
    irBoolsB &= ~irBit;
}

irBit = irBit << 1; // shift next-bit marker
if (irBit == 0)
    // and overflow if neccessary
{
    irBit = 1;
}

// if all 8 booleans are (un)set, we change state
if (currentState.robotPresent == false && irBoolsF == 0x00 &&
    irBoolsB == 0xff)
{
    currentState.robotPresent = true;
    P_LIBA("[DEBUG] IR changed to robot present!\n\r");
}
else if (currentState.robotPresent == true && irBoolsF == 0x00 &&

irBoolsB == 0x00)
{
    currentState.robotPresent = false;
P_LIBA("[DEBUG] IR changed to robot NOT present!\n\r");
}

// turn IR LED off after cycle
else if (irStage == 4)
{
    // turn IR LED off after cycle
tlc.setPinMode(LED_IR_R, PM_OFF);
tlc.setPinMode(LED_IR_L, PM_OFF);
}
else if (irStage == 5)
{
    irStage = 0;
}

/**
 * Send the current state of the TAM to the coordinator.
 * Uses global variable currentState for sending.
 * @return frame id of packet sent
 */
uint8_t sendStateToCoordinator()
{
    statePayload[0] = PACKET_TYPE_TC_CURRENT_STATE; // packet type
    statePayload[1] = currentState.redLed;
    statePayload[5] = battVoltage & 0xff;
    statePayload[6] = (battVoltage >> 8) & 0xff;
    statePayload[7] = currentState.robotData;

    statePacket.setFrameId(xbee.getNextFrameId());
xbee.send(statePacket);

    /*
     * P_XBEE_SEND("Sending packet (frame id %d):\n\r", statePacket.getFrameId());
     * P_XBEE_SEND(" Packet type: %02x\n\r", statePayload[0]);
     * P_XBEE_SEND(" State r,g,b: %02x,%02x,%02x\n\r", statePayload[1],
     * statePayload[2], statePayload[3]);
     * P_XBEE_SEND(" Robot there: %02x\n\r", statePayload[4]);
     */
P_XBEE_SEND(" Battery V : %02x, %02z\n\r", statePayload[5],
statePayload[6]);
*/

    return statePacket.getFrameId();
}

/*********************************************************************
*****************
*
* Main loop. Loops every CYCLE_LENGTH ms.
*
***********************************************************************/
void loop()
{
    // Check battery voltage and light barrier state. DO NOT swap the
two lines
    checkBatteryVoltage();
    checkLightBarriersAnalog();

#ifdef ENABLE_IRCOM
    // IRCOM Send data. We send every IRCOM_SEND_INTERVAL ms in order
to have enough time for receiving
    now = millis();
    if (currentState.robotPresent && ((IrComLastSendAt +
IRCOM_SEND_INTERVAL) < now))
    {
        IrComLastSendAt = now;
        ircomSend(irComDataToWrite);
        while (ircomSendDone() == 0);
        P_IRCOM("Send done. Value: %d\n\r", irComDataToWrite);
    }
#endif

    // report it to the coordinator if
    // - HEARTBEAT_INTERVAL passed, no matter what
    // - status changed and hasn’t been sent yet
    now = millis();
    if (!lastPacketAwaitingResponse &&
        (((lastHeartbeatSentAt + HEARTBEAT_INTERVAL) < now) ||
        (reportState == true) ||
        (memcmp(&lastReportedState, &currentState, sizeof(state_t)) !=
{lastPacketFrameId = sendStateToCoordinator();
lastPacketSentAt = now;
lastPacketAwaitingResponse = true;
P_XBEE_SEND("Sent state packet at cycle %d frame id %d\n\r", cycle, lastPacketFrameId);

if ((lastHeartbeatSentAt + HEARTBEAT_INTERVAL) < now)
{
    lastHeartbeatSentAt = now;
P_XBEE_SEND("  -> heartbeat\n\r");
}
else if (reportState == true)
{
P_XBEE_SEND("  -> reply after command\n\r");
reportState = false;
}
else
{
P_XBEE_SEND("  -> state change:
currentState.robotPresent=%d\n\r", currentState.robotPresent );
}

// check if we last packet sent timed out
now = millis();
if (lastPacketAwaitingResponse && (lastPacketSentAt + RESPONSE_TIMEOUT) < now)
{
P_XBEE_SEND("Error: didn’t get a response for previous packet frame id %d.\n\r", lastPacketFrameId);
lastPacketAwaitingResponse = false;
}

// try to read a packet and treat it if we did
xbee.readPacket();
if (xbee.getResponse().isAvailable())
{
// we got a reply to one of our previous packets; see if it’s expected
if (xbee.getResponse().getApiId() == DM_TX_STATUS_RESPONSE)
{
    xbee.getResponse().getDMTxStatusResponse(statePacketResponse);

    // get the delivery status, the fifth byte

if (statePacketResponse.getDeliveryStatus() == SUCCESS)
{
    if (lastPacketAwaitingResponse &&
        statePacketResponse.getFrameId() == lastPacketFrameId)
    {
        P_XBEE_SEND(" -> success, got response for frame id
%d\n\r", lastPacketFrameId);
        // update state as coordinator received it
        memcpy(&lastReportedState, &currentState, sizeof(state_t));
        lastPacketAwaitingResponse = false;
    } else
    {
        P_XBEE_SEND(" -> wtf? got response for unexpected frame id
%d\n\r", lastPacketFrameId);
        lastPacketAwaitingResponse = false; // TODO: check if this
        // is ok?
    }
} else
{
    P_XBEE_SEND(" -> FAIL! The remote XBee did not receive our
packet. Is it powered on?\n\r");
}
} else if (xbee.getResponse().getApiId() == DM_RX_RESPONSE)
{
    DMRxResponse rxResponse = DMRxResponse();
    xbee.getResponse().getDMRxResponse(rxResponse);
    P_XBEE_RECV("Received packet DM_RX_RESPONSE with data length =
%d\n\r", rxResponse.getDataLength());

    // check first byte if
    if (rxResponse.getData(0) == PACKET_TYPE_CT_SET_LEDS)
    {
        P_INFO("Got command SET_LEDS r=%d, g=%d, b=%d\n\r",
            rxResponse.getData(1), rxResponse.getData(2),
            rxResponse.getData(3));
            setRgbLeds(rxResponse.getData(1), rxResponse.getData(2),
                rxResponse.getData(3));
            reportState = true; // always reply with a state report after
getting a SET_LEDS command
    }
#endif ENABLE_IRCOM
    else if (rxResponse.getData(0) == PACKET_TYPE_CT_WRITE_ROBOT)
    {
        P_INFO("Got command WRITE_ROBOT value=%d\n\r",
            rxResponse.getData(1));
    }
}
rxResponse.getData(1));
    irComDataToWrite = rxResponse.getData(1);
}
#endif
else if (rxResponse.getData(0) == PACKET_TYPE_CT_SHUTDOWN)
{
    P_WARN("Got shutdown command, powering off...
            
            ");
    shutdown();
}
else
{
    P_XBEE_RECV("Got unknown packet type!
            
        ");
}
else
{
    reportState = true;
    P_XBEE_RECV("Received packet with unexpected api id %d
            
            ",
        xbee.getResponse().getApiId());
}
else if (xbee.getResponse().isError())
{
    // 1 - CHECKSUM_FAILURE 2 - PACKET_EXCEEDS_BYTE_ARRAY_LENGTH 3 -
    // UNEXPECTED_START_BYTE
    reportState = true; // reply with report when package was garbled
    P_XBEE_RECV("Error reading packet. Error code: %d
            
            ",
        xbee.getResponse().getErrorCode());
}
#elifdef ENABLE_IRCOM
    // receive data from the robot through IRcom
    if (currentState.robotPresent)
    {
        IrcomMessage imsg;
        ircomPopMessage(&imsg);
        if (imsg.error == 0)
        {
            int val = (int)imsg.value;
            if (val != currentState.robotData)
            {
                reportState = true;
                currentState.robotData = val;
            }
            P_IRCOM("currentState.robotData : %d
            
            ",
            val);
        }
    }
#endif
currentState.robotData);
    P_IRCOM("Receive successful value : %d\n\r", val);
  }
  else if (imsg.error > 0)
  {
    int val = (int) imsg.value;
    P_IRCOM("Receive FAILED with value : %d, ERROR: %d\n\r", val, imsg.error);
  }
}
#endif

// we keep the cycle under 10000 for easy display
  cycle++;
  if (cycle == 10000)
  {
    cycle = 0;
  }
  delay(CYCLE_LENGTH);
}
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>A picture of the e-puck robot</td>
<td>5</td>
</tr>
<tr>
<td>3.1</td>
<td>Conceptual drawing of the TAM</td>
<td>8</td>
</tr>
<tr>
<td>4.1</td>
<td>Screenshot from the ARGoS 1 simulator: single array</td>
<td>11</td>
</tr>
<tr>
<td>4.2</td>
<td>Screenshot from the ARGoS 1 simulator: experiment scene</td>
<td>11</td>
</tr>
<tr>
<td>5.1</td>
<td>Photos of the TAM</td>
<td>13</td>
</tr>
<tr>
<td>A.1</td>
<td>Main board – bill-of-materials</td>
<td>35</td>
</tr>
<tr>
<td>A.2</td>
<td>Side board (right) – bill-of-materials</td>
<td>36</td>
</tr>
<tr>
<td>A.3</td>
<td>Side board (left) – bill-of-materials</td>
<td>37</td>
</tr>
<tr>
<td>A.4</td>
<td>Main board – circuit schematics – Power supply</td>
<td>38</td>
</tr>
<tr>
<td>A.5</td>
<td>Main board – circuit schematics – Side board connectors</td>
<td>39</td>
</tr>
<tr>
<td>A.6</td>
<td>Main board – circuit schematics – LED controller</td>
<td>40</td>
</tr>
<tr>
<td>A.7</td>
<td>Main board – circuit schematics – Arduino base</td>
<td>41</td>
</tr>
<tr>
<td>A.8</td>
<td>Main board – circuit schematics – Xbee module</td>
<td>42</td>
</tr>
<tr>
<td>A.9</td>
<td>Side board (right) – circuit schematics</td>
<td>43</td>
</tr>
<tr>
<td>A.10</td>
<td>Side board (left) – circuit schematics</td>
<td>44</td>
</tr>
<tr>
<td>A.11</td>
<td>Main board – Prototype circuit board layout</td>
<td>45</td>
</tr>
<tr>
<td>A.12</td>
<td>Side board (right) – Prototype circuit board layout</td>
<td>46</td>
</tr>
<tr>
<td>A.13</td>
<td>Side board (left) – Prototype circuit board layout</td>
<td>46</td>
</tr>
</tbody>
</table>
List of Publications using the TAM


Bibliography


