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 IRIDIA TAM: A device for task abstraction for the e-puck robot

Arne Brutschy, Giovanni Pini, Nadir Baiboun, Antal Decugnière, and Mauro Birattari
# Contents

1 Introduction .............................................. 2  
1.1 Note on updated version ................................. 2  
1.2 Using the IRIDIA TAM ................................ 2  

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

3 Concept ................................................... 7  
3.1 Basic concept .......................................... 7  
3.2 Usage examples ........................................ 8  

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

5 Real-World Implementation ............................... 12  
5.1 Hardware ................................................. 12  
5.2 Firmware ................................................ 13  

6 Conclusions and Future Work ............................. 15  
6.1 Published works ....................................... 15  

A Appendix .................................................. 16  
A.1 Simulation ............................................... 16  
A.1.1 ARGoS Entity ........................................ 16  
A.2 Real World ............................................. 26  
A.2.1 Hardware ............................................. 26  
A.2.2 Firmware ............................................. 33  

List of Figures ................................................ 40  
References .................................................... 40  
Bibilography .................................................. 42
1 Introduction

This technical report describes the IRIDIA 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 IRIDIA 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 IRIDIA TAM. 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 IRIDIA 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 IRIDIA 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 IRIDIA TAM

All necessary information to replicate the IRIDIA TAM can be found in this technical report and the supplementary on-line material (Brutschy et al., 2012).
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 IRIDIA TAM non-commercially, as long as you credit the IRIDIA TAM and its authors, and license your new creation under the identical terms. You are free to use the IRIDIA TAM or any reproduction thereof in your experiments.

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

```bibtex
@techreport{BruPinBai-etal2010:IridiaTAM,    
  Author = {Arne Brutschy and Giovanni Pini and Nadir Baiboun and Antal Decugni\'ere and Mauro Birattari},  
  Title = {The \textsf{IRIDIA TAM}: A device for task abstraction for the e-puck robot},  
  Institution = {IRIDIA, Universit\'e 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 IRIDIA TAM (see page 40).
2 Motivation

In this chapter, we introduce the reasoning that stands behind the creation of the IRIDIA 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 IRIDIA TAM has been designed. Finally, we will present classes of experiments and possible ideas that can be enabled by using the IRIDIA 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:

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

(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;
2 Motivation

Proximity sensors (8)
Wheels actuator (max. 8 cm/s)
Color camera
IR rays

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/
2 Motivation

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 IRIDIA 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 IRIDIA 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 IRIDIA 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 IRIDIA 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 IRIDIA TAM in the environmental setup.
3 Concept

![Diagram of IRIDIA TAM](image)

Figure 3.1: Conceptual drawing of the IRIDIA TAM. The light barrier detects a robot entering into the module. Upon the detection of a robot, the module reacts by changing the color of its LEDs following a user-defined logic. In our experiments, the color of the LEDs represent the different types of task.

In this chapter, we explain the concept of the IRIDIA 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 this device, referred to as IRIDIA TAM (task abstraction module).

3.1 Basic concept

Each module is equipped with two 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 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 IRIDIA 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 IRIDIA TAM given in this section is intended for ARGoS version 1. An updated version for ARGoS version 2 is in preparation.

The IRIDIA TAM has been implemented as actuated device in the ARGoS simulation framework (Pinciroli et al., 2011). 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 IRIDIA 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 IRIDIA 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 IRIDIA TAM can not be transferred 1:1 on the real world device. This is no major drawback as the controllers for the IRIDIA TAM are usually very simple.
Figure 4.1: Screenshot from the ARGoS simulator which shows a single IRIDIA 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 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 IRIDIA TAM. First, we will focus briefly on the hardware implementation, and then we will outline the low-level firmware that allows the IRIDIA TAM to be versatile and re-programmable depending on the experiment.

5.1 Hardware

The real-world, hardware implementation of the IRIDIA 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 IRIDIA 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 IRIDIA TAM directly via USB to a host computer. Controllers can be written on the microcontroller of the IRIDIA TAM directly using this connection, without the requirement of a

¹http://www.cadsoft.de/
5 Real-World Implementation

Figure 5.1: The prototype of the IRIDIA TAM. The light barrier detects a robot entering into the module. Upon the detection of a robot, the module reacts by changing the color of its LEDs following a user defined logic. In our experiments, the color of the LEDs represent the different types of task.

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 IRIDIA TAM can be found in Appendix A.2.1. The circuit boards, which form the electronics as well as the body of the IRIDIA 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 IRIDIA 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 IRIDIA 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 IRIDIA 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 IRIDIA 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 representation.

The device has been, to date, used with success in simulation experiments of various students and researchers (see page 40 for a list of publications using the IRIDIA TAM). The hardware device has been prototyped and successfully tested. The production of the device in numbers that allow extensive real-world experiments with the e-puck is currently in progress.

A future extension of the IRIDIA TAM might include the capability of communication between the robot in the module and the module itself. This communication would be realized by using the e-pucks proximity sensor. The necessary sensor is already included in the current prototype, and a suitable protocol has been developed already.

6.1 Published works

The IRIDIA TAM has been successfully used in several scientific experiment over the last years, be it in journal papers, conference papers, or master's thesis. See page 40 for a list of publications using the IRIDIA TAM. If you are using the IRIDIA TAM for your publications and they are not listed, please report your publications to Arne Brutschy (arne.brutschy@ulb.ac.be).
A Appendix

A.1 Simulation

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

A.1.1 ARGoS 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.
 *
 * @author Marco Frison - <mfrison85@gmail.com>
 */

16
#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 );

class CGateEntity( );
c~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;

};
}
#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
* 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.cpp>
* *
* @brief This file contains the implementation of the gate entity.
* *
* This file contains the class implementation of the gate entity.
* *
* @author Marco Frison - <mfrison85@gmail.com>
*/

#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;

    int nTmpDoor;
    s_doors = CExperimentConfiguration::GetNodeValue(
        t_configuration_tree, CONFIGURATION_GATE_DOORS);

    return 0;
}
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;
            break;
    }

    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 Appendix

A.2 Real World

A.2.1 Hardware

Partlist

Parts list for all three boards: Table A.1

Schematics

Circuit schematics of the main board (Arduino base circuit): Figure A.1
Circuit schematics of the side board (right): Figure A.2
Circuit schematics of the side board (left): Figure A.3

Circuit Boards

Prototype circuit board layout of the main board: Figure A.4
Prototype circuit board layout of the side board (right): Figure A.5
Prototype circuit board layout of the side board (left): Figure A.6
### Table A.1: Parts list for all three boards.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description/Venue</th>
<th>Value</th>
<th>Qty</th>
<th>Digi-Key Part#</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>CER RESONATOR 16.0MHZ SMD XTAL</td>
<td></td>
<td>1</td>
<td>490-193-1-ND</td>
</tr>
<tr>
<td>C1, C4, C5, C6, C7</td>
<td>CAP0805 CAP CERAMIC X7R 0805 0.1μF, 16V</td>
<td></td>
<td>5</td>
<td>311-1142-1-ND</td>
</tr>
<tr>
<td>C2</td>
<td>CAP0805 C2012X5R1C106M/1.25 10μF 16V 1</td>
<td></td>
<td>1</td>
<td>445-4115-1-ND</td>
</tr>
<tr>
<td>IC1</td>
<td>ATMEGA328P-AU microcontroller</td>
<td>TQFP32-08</td>
<td>1</td>
<td>A TMEGA328P-AU-ND</td>
</tr>
<tr>
<td>IC2</td>
<td>MCP1703CB IC REG LDO 3.3V 250MA SOT-223-3</td>
<td></td>
<td>3.3V</td>
<td>1 MCP1703-3302E/DB-ND</td>
</tr>
<tr>
<td>R1, R3, R4</td>
<td>RES 10.0K OHM 1/8W 1% 0805 SMD</td>
<td></td>
<td>10k</td>
<td>3 RHM10.0K CCT-ND</td>
</tr>
<tr>
<td>R2, R12</td>
<td>RES 47.0 OHM 1/8W 1% 0805 SMD</td>
<td></td>
<td>2</td>
<td>2 RHM47.0CCT-ND</td>
</tr>
<tr>
<td>FT2</td>
<td>RFM12B RFM12B radio module 868Mhz 1</td>
<td></td>
<td>1</td>
<td>751-1031-ND</td>
</tr>
<tr>
<td>SV1</td>
<td>CONN HEADER DUAL R/A PCB (ISP)</td>
<td></td>
<td>8V</td>
<td>1 609-2846-ND</td>
</tr>
<tr>
<td>SV2</td>
<td>CONN HEADER DUAL R/A PCB (ISP)</td>
<td></td>
<td>8V</td>
<td>1 609-2846-ND</td>
</tr>
<tr>
<td>SV3</td>
<td>RES 10K OHM 1/8W 0805 SMD</td>
<td></td>
<td>1  R324</td>
<td>751-1031-ND</td>
</tr>
<tr>
<td>SV4</td>
<td>RES 10K OHM 1/8W 0805 SMD</td>
<td></td>
<td>1  R324</td>
<td>751-1031-ND</td>
</tr>
<tr>
<td>SV5</td>
<td>RES 10K OHM 1/8W 0805 SMD</td>
<td></td>
<td>1  R324</td>
<td>751-1031-ND</td>
</tr>
<tr>
<td>SV6</td>
<td>RES 10K OHM 1/8W 0805 SMD</td>
<td></td>
<td>1  R324</td>
<td>751-1031-ND</td>
</tr>
<tr>
<td>SV7</td>
<td>RES 10K OHM 1/8W 0805 SMD</td>
<td></td>
<td>1  R324</td>
<td>751-1031-ND</td>
</tr>
<tr>
<td>SW1</td>
<td>SW TACTILE SPDT A/V A/NC Pcb</td>
<td></td>
<td>SW1</td>
<td>1 609-2846-ND</td>
</tr>
<tr>
<td>SW2</td>
<td>SW TACTILE SPDT A/V A/NC Pcb</td>
<td></td>
<td>SW1</td>
<td>1 609-2846-ND</td>
</tr>
<tr>
<td>LED1</td>
<td>LED-1206 CHIPS 3101-2 Led Green 1.206 SMD GREEN</td>
<td></td>
<td>1  TLED-1.206-2LED2 GREEN</td>
<td>475-1407-1-ND</td>
</tr>
<tr>
<td>LEDR</td>
<td>LED-1206 CHIPS 3101-2 LED2 GREEN 1206 Diff Pcb</td>
<td></td>
<td>2  TLED-1.206-2LED2 GREEN</td>
<td>475-2756-1-ND</td>
</tr>
<tr>
<td>LEDL</td>
<td>SFH 4258 IR LED</td>
<td></td>
<td>IRLED</td>
<td>1 475-1243-1-ND</td>
</tr>
<tr>
<td>IR TRANS</td>
<td>SFH 320 IR TRANSISTOR</td>
<td></td>
<td>IR TRANS</td>
<td>1 SFH 320 IR TRANS</td>
</tr>
<tr>
<td>R5, R8, R11</td>
<td>RES 18.0 OHM 1/8W 0805 SMD</td>
<td></td>
<td>18</td>
<td>3 RHM18.0CCT-ND</td>
</tr>
<tr>
<td>R6, R7, R9, R10</td>
<td>RES 2.20 OHM 1/8W 0805 SMD</td>
<td></td>
<td>2.2</td>
<td>4 RHM2.20CCT-ND</td>
</tr>
<tr>
<td>PRX</td>
<td>TCR T1000</td>
<td></td>
<td>PRX</td>
<td>1 751-1031-ND</td>
</tr>
</tbody>
</table>

**Note:** The table above is a part of Table A.1: Parts list for all three boards. This table lists the components used in the boards, including their types, quantities, and part numbers.
Figure A.1: Circuit schematics of the main board (Arduino base circuit).
Figure A.2: Circuit schematics of the side board (right).
Figure A.3: Circuit schematics of the side board (left).
Figure A.4: Prototype circuit board layout of the main board.
A Appendix

Figure A.5: Prototype circuit board layout of the side board (right).

Figure A.6: 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. Additional logic that sets task type and length must be inserted according to the experiment at hand. Please note that the battery voltage check must be kept in place as without it the battery might be damaged.

```c
#include <JeeLib.h>
#include <EEPROM.h>
#include <LowPower.h>
#include <stdarg.h>

/********************************************************************
* Example Arduino Controller for the IRIDIA TAM
* This sketchbook requires a JeeLink USB on the experimenters
* computed that acts as a central node for logging data and
* setting task ratios.
* 
* ***************************************************************************/
#define DEBUG 1

// configurable variables
#define LED_RED_PIN  3
#define LED_GREEN_PIN  5
#define LED_BLUE_PIN  6
#define LED_ACT_PIN  9
#define IR_WAIT_TIME  2
#define IR_THRESHOLD  60
#define IR_ENABLE_PIN A4
#define IR_EMIT_PIN A2
#define IR_SENSE_PIN A0
#define PROX_ENABLE_PIN A5
#define PROX_EMIT_PIN A3
#define PROX_SENSE_PIN A1
#define BATT_VOLTAGE_THRESHOLD 2900 // 2.9V, might be lower
```
#define BATT_UNDER_VOLTAGE_COUNTER_THRESHOLD 10

#define LOG_NODE_ID 1

// initialize variables
int irReadPre;
int irReadPost;
boolean irInterrupted = false;
int battVoltage = 0;
byte battUnderVoltageCounter = 0;
byte nodeId;

//MilliTimer timer;
//http://jeelabs.net/projects/11/wiki/Class_MilliTimer

/**
 * Small helper function that emulates printf()
 */
void p(char *fmt, ... )
{
    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);
}

/**
 * Set activity LED on/off
 */
void activityLed(byte state)
{
    if (state == 0)
        digitalWrite(LED_ACT_PIN, HIGH);
    else
        digitalWrite(LED_ACT_PIN, LOW);
}

/**
 * Sets the RGB LEDs (all colors in a single function call).
 */
void rgbLeds(byte red, byte green, byte blue)
{
    analogWrite(LED_RED_PIN, red);
    analogWrite(LED_GREEN_PIN, green);
/**
 * Prepare serial and set necessary digital pins to output.
 * Also initializes the RF module with pre-configured values (i.e., node id).
 */
void setup()
{
  Serial.begin(57600);
  pinMode(LED_RED_PIN, OUTPUT);
analogWrite(LED_RED_PIN, 0);
  pinMode(LED_GREEN_PIN, OUTPUT);
analogWrite(LED_GREEN_PIN, 0);
  pinMode(LED_BLUE_PIN, OUTPUT);
analogWrite(LED_BLUE_PIN, 0);
  pinMode(LED_ACT_PIN, OUTPUT);
analogWrite(LED_BLUE_PIN, blue);
  activityLed(true);
  pinMode(IR_ENABLE_PIN, OUTPUT);
digitalWrite(IR_ENABLE_PIN, HIGH);
  pinMode(IR_EMIT_PIN, OUTPUT);
digitalWrite(IR_EMIT_PIN, LOW);
  pinMode(PROX_ENABLE_PIN, OUTPUT);
digitalWrite(PROX_ENABLE_PIN, LOW);
  pinMode(PROX_EMIT_PIN, OUTPUT);
digitalWrite(PROX_EMIT_PIN, LOW);

  // initialize RF module with values read from eeprom
  // use rf12demo sketch to configure
 NodeId = rf12_config(true);
}

/**
 * Shutdown all sensors, LEDs and the RF module to save energy.
 * Then, put the uC to sleep.
 * Called when we detect a low battery.
 */
void shutdown()
{
// power off all sensors and their associate leds
digitalWrite(IR_ENABLE_PIN, LOW);
digitalWrite(IR_EMIT_PIN, LOW);
digitalWrite(PROX_ENABLE_PIN, LOW);
digitalWrite(PROX_EMIT_PIN, LOW);

// switch off RGB LEDs
rgbLeds(0, 0, 0);

// power off RF module
rf12_sleep(0);

// switch off activity LED
activityLed(false);

// power down the ATmega
LowPower.powerDown(SLEEP_FOREVER, ADC_OFF, BOD_OFF);

/**
 * Reads the vcc on the ATmega by comparing the input
 * voltage to the 1.1V reference voltage.
 */
long readVcc()
{
    long result;

    // Read 1.1V reference against AVcc
    ADMUX = _BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
delay(2); // Wait for Vref to settle
    ADCSRA |= _BV(ADSC); // Convert
    while (bit_is_set(ADCSRA, ADSC));
    result = ADCL;
    result |= ADCH << 8;
    result = 1126400L / result; // Back-calculate AVcc in mV

    return result;
}

/**
 * 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 = readVcc();
//p("[DEBUG] Battery voltage: %d\n", readVcc());
if (battVoltage < BATT_VOLTAGE_THRESHOLD)
{
    battUnderVoltageCounter++;
    if (battUnderVoltageCounter == BATT_UNDER_VOLTAGE_COUNTER_THRESHOLD)
    {
        p("[WARNING] Battery voltage is too low, powering down!\n");
        delay(100);
        shutdown();
    }
}
else
{
    battUnderVoltageCounter = 0;
}
}

/**
 * Checks if the light barrier is blocked.
 * Returns true if yes.
 */
boolean isLightBarrierBlocked()
{
    // read analog before we turn LED on
    irReadPre = analogRead(IR_SENSE_PIN);

    // turn IR LED on
    digitalWrite(IR_EMIT_PIN, HIGH);

    // wait for some short time so that the transistor has the time to react
    delay(IR_WAIT_TIME);

    // then read out analog value from receiver
    irReadPost = analogRead(IR_SENSE_PIN);

    // turn IR LED off after cycle
    digitalWrite(IR_EMIT_PIN, LOW);

    // debug output
    //p("[DEBUG] Reading analog difference: pre=%d post=%d diff=%d\n",
        irReadPre, irReadPost, abs(irReadPost - irReadPre));

    return (abs(irReadPost - irReadPre) < IR_THRESHOLD);
}
```c
boolean wantToSend = false;
struct packet_t {
  byte header;
  byte data[RF12_MAXDATA];
  byte len;
};
packet_t packet;

/**
 * Main loop.
 */
void loop()
{
  if (Serial.available())
  {
    char c = Serial.read();
    p("IRIDIA TAM: ", nodeId);
    nodeId = rf12_config(true);
  }

  checkBatteryVoltage();

  if (rf12_recvDone() && rf12_crc == 0)
  {
    activityLed(false);
    p("Receive OK ");
    for (byte i = 0; i < rf12_len; ++i)
      p("%c", rf12_data[i]);
    p("\n");
    delay(100); // otherwise led blinking isn't visible
    activityLed(true);
    wantToSend = false;
  }

  if (wantToSend && rf12_canSend())
  {
    activityLed(false);
    // p("[DEBUG] Sending...");
    rf12_sendStart(packet.header, packet.data, packet.len);
    delay(100); // otherwise led blinking isn't visible
    activityLed(true);
    wantToSend = false;
  }
```
// change RGB LEDs state if IR light barrier is interrupted
boolean lightBarrier = isLightBarrierBlocked();
if (!irInterrupted && lightBarrier)
{
    irInterrupted = true;
    rgbLeds(50, 0, 0);

    //p("[DEBUG] Interrupted!
    packet.header = LOG_NODE_ID;
    packet.data[0] = nodeId;
    packet.len = 1;
    wantToSend = true;
}
else if (!lightBarrier)
{
    irInterrupted = false;
    rgbLeds(0, 50, 0);
}

//p("[DEBUG] Reading PROX value: %d\n", analogRead(PROX_SENSE_PIN));

// if we iterate too fast logmessages garble the serial out
delay(200);
List of Figures

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. 5

3.1 Conceptual drawing. 7

4.1 Screenshot from the ARGoS simulator: single array. 11
4.2 Screenshot from the ARGoS simulator: experiment scene. 11

5.1 Photo of the IRIDIA TAM prototype. 13

A.1 Main board – circuit schematics – Arduino base. 28
A.2 Side board (right) – circuit schematics. 29
A.3 Side board (left) – circuit schematics. 30
A.4 Main board – Prototype circuit board layout. 31
A.5 Side board (right) – Prototype circuit board layout. 32
A.6 Side board (left) – Prototype circuit board layout. 32
List of Publications using the IRIDIA TAM


