Viktor Kozma and David Broman. MORAP: a Modular Robotic Arm Platform for Teaching and Experimenting with Equation-based Modeling Languages. In Proceedings of 7th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT). Milano, Italy, ACM, 2016. DOI: DOI: http://dx.doi.org/10.1145/2904081.2904085

MORAP: a Modular Robotic Arm Platform for Teaching and Experimenting with Equation-based Modeling Languages

[Work in Progress]

Viktor Kozma
KTH Royal Institute of Technology
vkozma@kth.se

David Broman
KTH Royal Institute of Technology and
University of California, Berkeley
dbro@kth.se

ABSTRACT

Equation-based object-oriented (EOO) modeling and simulation techniques have in the last decades gained significant attention both in academia and industry. One of the key properties of EOO languages is *modularity*, where different components can be developed independently and then connected together to form a complete acausal model. However, extensive modeling without explicit model validation together with a real physical system can result in incorrect assumptions and false conclusions. In particular, in an educational and research setting, it is vital that students experiment both with equation-based models *and* the real system that is being modeled. In this work-in-progress paper, we present a physical experimental robotic arm platform that is designed for teaching and research. Similar to EOO models, the robotic arm is *modular*, meaning that its parts can be reconfigured and composed together in various settings, and used for different experiments. The platform is completely open source, where electronic schematics, CAD models for 3D printing, controller software, and component specifications are available on GitHub. The vision is to form a community, where new open source components are continuously added, to enable an open and freely available physical experimental platform for EOO languages.

CCS Concepts

•**Computing methodologies** → *Model development and analysis;* •**Computer systems organization** → *Embedded systems; Real-time systems;* Robotic components;

Keywords

Modeling; simulations; equations; robotic arm

1. INTRODUCTION

Equation-based object-oriented (EOO) languages, such as Modelica [8], various research languages (Hydra [6], Modelyze [4], Sol [12] etc.), and hardware description languages with mixed-signal extensions (e.g., VHDL-AMS and Verilog-AMS) support acausal modeling both at the equation-level and component level. Compared to causal modeling (e.g., block diagrams in MATLAB/Simulink), the strength of the acausal modeling paradigm is rapid prototyping where the topology of connected model components directly corresponds to a physical system. Acausal components enable *modular* modeling, where model components can be developed independently and then be composed together rapidly.

However, in an educational environment, it is not only important to learn about rapid component-based modeling, but also to learn about equation-level modeling, embedded systems design, parameter estimation, and model validation of real systems. Moreover, in a modeling language research context, it is important that language features are designed and motivated based on modeling of real systems, not only by synthetic models and benchmarks. Developing physical experiment equipment is, however, both expensive and time consuming. Standard robotic arms, such as robots used in industrial automation, can be used in many experiment scenarios and can be directly modeled using model libraries. Purchasing a real industrial robot is, however, extremely expensive, resulting in that many researchers focus on pure software based modeling and simulation. Within the robotic research community, extensive research has been performed within modular reconfigurable robots [11] and biologically inspired robots [10]. Although such robots are modular, their autonomous characteristics of selfreconfiguration do not match the needs of a modular experiment platform.

In this paper, we present a preliminary solution of a simple *MOdular Robotic Arm Platform (MORAP)* that is designed as a test platform for experimenting with EOO languages. The platform is available as open source[1] and contains standard components that can be configured by the experiment designer into different experiment scenarios. More specifically, we present the following:

- We discuss the overall idea, objective, and the design of the modular platform, both from a physical system and an EOO modeling perspective (Section 2).
- We briefly describe the components that are developed so far, including CAD files for 3D printing, electronics, and software API (Section 3).
- We outline two work-in-progress case studies, demonstrating the foundation of the platform (Section 4).

[1] http://www.github.com/modelyze/morap
(using the simplified BSD license)

EOOLT '16, April 18 2016, Milano, Italy

DOI: http://dx.doi.org/10.1145/2904081.2904085

2. PLATFORM OVERVIEW

In this section, we explain the objective of the platform, followed by a brief overview of the design principles, both from a system and a modeling perspective.

2.1 Objective

The main objective of MORAP is to be a simple teaching and research platform. From a teaching perspective, the platform should enable students to learn about equation-based modeling, control, embedded systems skills [1] and how it directly relates to the real physical systems. From a research perspective, the platform can be used for testing when developing equation-based languages, with new features and capabilities. More specifically, important properties of the platform are:

- all components (3D model CAD files , electronic schematics, software etc.) are freely available as open source using the simplified BSD license.
- the physical components have clear physical interfaces, to enable simple assembling and reconfigurations of the robotic arm.
- there is a simple and well documented software API for controlling the system.
- there are simple and reusable equation-based modeling libraries, which match the physical components.

2.2 Modular Hardware System Design

Consider Figure 1 that depicts the three main classes of modules in a physical modular robotic arm: i) joints, ii) links, and iii) end effectors. The *joints* can contain bearings and motors for controlling the arm, or only bearings and sensors (see the case study with the inverted pendulum). It is important that they have the same physical interface to the *links* that are connecting the joints together, and the same electrical interface, which enables several joints to be controlled from the same controller platform. An *end effector* can be attached to the end of the arm, to enable interaction with the environment. The figure illustrates two examples of end effectors: a pen and a gripper.

The different modules can then be combined into various configurations with different degrees of freedom. Figure 2 shows examples of configurations in 2 and 3 dimensions, with different number of joints, link lengths, and end effectors.

To enable a modular design, the electrical system is designed as a distributed system using a standardized interface. All components are equipped with microcontrollers that communicate over a central data bus. A master controller can then communicate with each motor node, writing commands, and reading data. Other components like end effectors and sensors can also be placed on the data bus, allowing extensible sensing and manipulation.

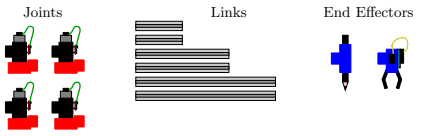

Figure 1: The main components used to construct a working robotic arm.

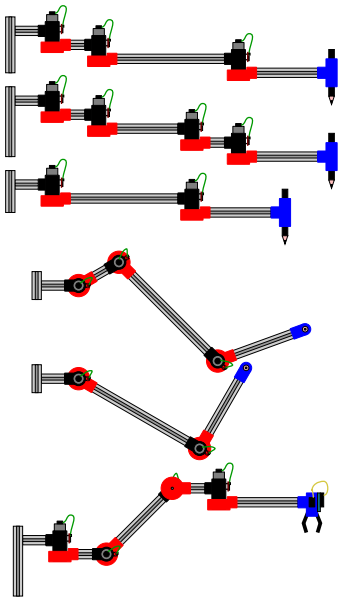

Figure 2: Different configurations, both in 2 and 3 dimensions.

2.3 Modular Equation-Based Modeling

As an educational and research platform, any available EOO language with a multibody library can be used for modeling and simulation of the robotic arm. We have so far focused on developing models in the research language Modelyze [2, 4], an experimental host language for developing equation-based domain specific languages (DSLs). The following listing shows a 2D model of a robotic arm, with two links and two joints.

```
1  def 2d_arm_system() = {
2    def f1,f2,f3,f4,f5 : Mechanical2D;
3    def u1,u2,sth1,sth2 : Signal;
4
5    Fix(f1,f5);
6    JointType1(u1,f1,f2);    // Joint 1
7    Link20x20mm(0.4,f2,f3); // 0.4m link
8    JointType2(u2,f3,f4);    // Joint 2
9    Link20x20mm(0.2,f4,f5); // 0.2m link
10
11   SineVoltage(12.0,0.6,u1); // 12V, 0.6Hz
12   SineVoltage(6.0,1.2,u2);  // 6V, 1.2Hz
13
14   RotSensor(f2,sth1); // rotation sensor
15   RotSensor(f4,sth2); // rotation sensor
16 }
```

The current 2D library is based on the existing approaches developed in existing Modelica libraries [9, 13].

Note how the joints, the links, and the electrical signals are connected together by node-based semantics [3], meaning that nodes (e.g.. `f1` and `f2`) are used to connect components together. `Link20x20mm` models a general 20x20 mm aluminum profile and `JointType1` and `JointType2` are two different joints driven by two different DC motors. Both are based on the same equations, but contain different parameters for the different motors.

Modeling this fairly simple setup is possible in an EOO

language such as Modelica. From a modeling language research point of view, possible physical extensions can make the model expressiveness more challenging, thus making it more interesting from a language research point of view. For instance,

- if the arm can grip heavy objects, the inertia of the arm changes dynamically, and thus the dynamics of the arm.
- if for instance two different arms are throwing objects to each other, the modeling language needs to handle structurally dynamic systems.
- if sensing and actuation are performed on different platforms, modeling and realization of a combined *cyber-* (network timing and embedded computation) *physical* (plant model together with sensors and actuator models) *system* (CPS) can result in new interesting language aspects.

3. CURRENT COMPONENT DESIGN

Figure 3 shows the currently available joint [7], which consists of a DC motor, an individual electronic circuit-board, bearings, and 3D printed parts that hold the joint together.

3.1 3D CAD Models

All components are 3D printed, which allows the attachment of 20x20mm aluminum profile links. Each joint contains two angular contact ball bearings, allowing the joint to take loads in all mounting directions. Figure 4 shows the 3D CAD model of the different parts of the joint.

3.2 Electronics

Each motor node is controlled by a PIC24f microcontroller on a printed circuit board (PCB) containing a motor driver and connectors to power and the central data bus.

Each motor is equipped with a controller that enables both position and speed control. The controllers are capable of tuning themselves to three different types of controllers (PD and PID position control and PI speed control) based on the working environment (load inertia) and requested performance (closed loop pole).

Currently, we use a PIC32 microcontroller mounted on a ChipKIT UNO32 development board as a master controller, which also supplies different supply and logic voltages and allows the user to interact with the system. The system is easy to extend: both the master controller and the joints can potentially be replaced by other components, such as a Raspberry Pi-based master controller, or Arduino-based servos.

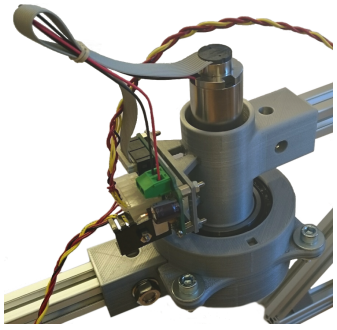

Figure 3: A hardware joint driven by a Maxon DC motor equipped with an encoder.

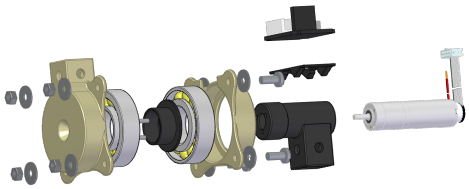

Figure 4: Exploded view of a joint to visualize the construction process.

3.3 Controller API

The I^2C serial communication protocol is used for communication between the master controller and the different joints and end effectors. Each motor node has an unique address and several available commands with their unique identifiers. These commands include setting position and velocity setpoints, calibrating encoders, enabling/disabling the joints, and tuning their controllers. Information, such as angle and status, can also be read from the joints and end effectors, allowing the user to build feedback into their systems. The most important I^2C commands are listed in the following table.

ID	Message Type
0	disable motor
1	set position reference (angle)
2	set speed reference
3	set voltage
16	calibrate encoder to zero radians
31	set encoder calibration status unknown
34	tune control parameters
128	next read operation will return the angle

4. CASE STUDIES

This section briefly describes two work-in-progress case studies using the MORAP platform.

4.1 Two-Linked Arm with End Effector

Figure 5 shows an application that uses two joints and one end effector that can be moved in two dimensions. It demonstrates the possibility to place the end effector at any place within its reach through the use of inverse kinematics.

Practical applications of this system involves evaluating the model of the controller and providing a test platform for motion planning algorithms.

4.2 Furuta Pendulum

Figure 6 shows the physical setup of a Furuta pendulum [5], a pendulum mounted at the end of a single jointed link. This experiment is a classic inverted pendulum problem, where a control algorithm has to be developed in order to keep the pendulum in an upright stabilized position.

The physical setup consists of a motor joint providing the main actuation, and an orthogonal free-running joint, where the pendulum is attached. The angle of the pendulum is measured by an MPU-9150 inertial measurement unit, con-

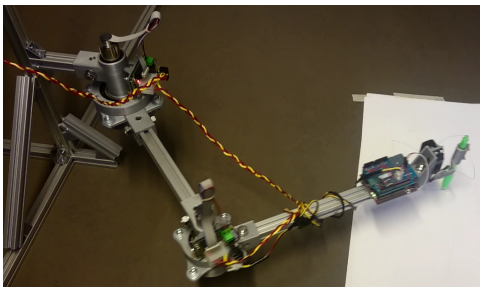

Figure 5: Hardware setup for a two-linked robotic arm with a pen end effector.

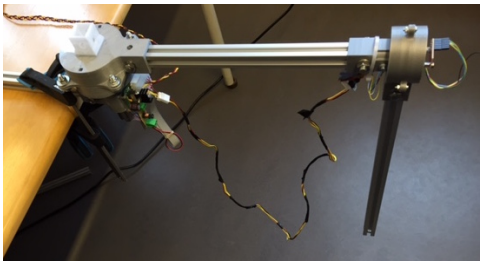

Figure 6: Hardware setup for a Furuta pendulum.

nected on the main data bus through a slip ring, allowing for full 360° continuous rotation. As part of the case study, we model the pendulum in Modelyze using the explicit equations described by Cazzolarto et al. [5], simulate it using the Modelyze simulator, and animate it in MATLAB (see Figure 7). Equation-based modeling showed that controlling the pendulum worked in simulation. In this work-in-progress prototype, the pendulum can be physically controlled and stand up only for a short while. The backlash in the motor and the measurements of the angle introduce potential sources of errors. However, the intention is to further refine the components and to improve the open platform over time.

5. CONCLUSIONS

In this work-in-progress paper, we describe the initiative of providing an open source modular robotic arm platform. The platform has been developed during the past year, resulting in a standardized motor joint, a free-running joint, and one end effector. The intention is to continuously extend the platform with new modules. Our hope is that it will be used both for teaching and as a test platform for equation-based modeling language research.

6. REFERENCES

[1] S. Behere and M. Törngren. Educating embedded systems hackers: A practitioner's perspective. In *Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education (WESE)*, pages 1:1–1:8, New York, NY, USA, 2015. ACM.

[2] D. Broman. *Meta-Languages and Semantics for Equation-Based Modeling and Simulation*. PhD thesis, Department of Computer and Information Science, Linköping University, Sweden, 2010.

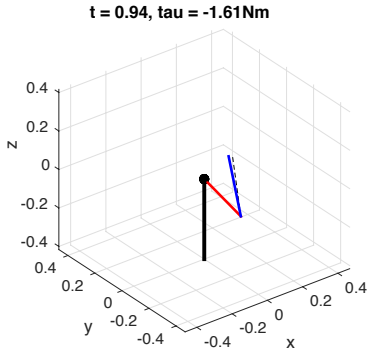

Figure 7: Modelyze model of a Furuta pendulum, animated in MATLAB.

[3] D. Broman and H. Nilsson. Node-Based Connection Semantics for Equation-Based Object-Oriented Modeling Languages . In *Proceedings of the Fourteenth International Symposium on Practical Aspects of Declarative Languages (PADL)*, volume 7149 of *LNCS*, pages 258–272. Springer, 2012.

[4] D. Broman and J. G. Siek. Modelyze: a gradually typed host language for embedding equation-based modeling languages. Technical Report UCB/EECS-2012-173, EECS Department, University of California, Berkeley, June 2012.

[5] B. S. Cazzolato and Z. Prime. On the dynamics of the furuta pendulum. *Journal of Control Science and Engineering*, 2011:3:1–3:8, Jan. 2011.

[6] G. Giorgidze and H. Nilsson. Embedding a functional hybrid modelling language in Haskell. In *IFL 2008, Revised Selected Papers*, volume 5836 of *LNCS*, pages 138–155. Springer, 2011.

[7] V. Kozma. Design of Modular Robotic Arms With High Model Fidelity. Master's thesis, KTH Royal Institute of Technology, 2015.

[8] Modelica Association. *Modelica - A Unified Object-Oriented Language for Physical Systems Modeling - Language Specification Version 3.3 Revision 1*, 2014. Available from: http://www.modelica.org.

[9] M. Otter, H. Elmqvist, and S. E. Mattsson. The New Modelica MultiBody Library. In *Proceedings of the 3rd International Modelica Conference*, 2003.

[10] R. Pfeifer, M. Lungarella, and F. Iida. Self-organization, embodiment, and biologically inspired robotics. *Science*, 318(5853):1088–1093, 2007.

[11] M. Yim, Y. Zhang, and D. Duff. Modular robots. *Spectrum, IEEE*, 39(2):30–34, 2002.

[12] D. Zimmer. Introducing Sol: A General Methodology for Equation-Based Modeling of Variable-Structure Systems. In *Proceedings of the 6th International Modelica Conference*, pages 47–56, 2008.

[13] D. Zimmer. A Planar Mechanical Library for Teaching Modelica. In *Proceedings of the 9th International Modelica Conference*, pages 681–690, 2012.