# Challenges in Physical Modeling for Adaptation of Cyber-Physical Systems

Ivan Ruchkin, Selva Samuel, Bradley Schmerl, Amanda Rico, and David Garlan
*Institute for Software Research, Carnegie Mellon University, Pittbsurgh, PA, USA*
*Email: {iruchkin,ssamuel,schmerl,garlan}@cs.cmu.edu*
*arico1@andrew.cmu.edu*

***Abstract*—Cyber-Physical systems (CPSs) are systems that mix software and physical control, with equal prominence. Typically, models at run time have concentrated on managing and controlling the cyber (or software) aspects of these systems. However, to fully realize the goals of CPS, physical models have to be treated as first class models in these systems. This gives rise to three main challenges: (a) identifying and integrating physical and software models with different characteristics and semantics; (b) obtaining instances of physical models at a suitable level of abstraction for control; and (c) using and adapting physical models to control CPSs. In this position paper, we elaborate on these three challenges and provide insight into how we are addressing these challenges in the context of power adaptation for a robotic system.**

## I. Introduction

State-of-the-art self-adaptation relies on multiple layers of models at run time to enable sophisticated decision-making and adaptation. It is common to separate the system under adaptation from the adaptation layers that govern control of increasingly higher-level concerns, from specific adaptive strategies like adding a server to managing long-term goals of the system like user satisfaction or mission achievement. In particular, models of the software being adapted are used to reason about and manage software-intensive systems. Common models used include architecture models [1], [2], [3], [4], [5], goal and requirements models [6], [7], and models of tasks [8].

Most self-adaptive approaches predominantly consider sophisticated cyber models but consider physical characteristics of the the system with less sophistication, and mostly only insofar as modeling resources available to the software. In architectural models, for instance, physical elements may be abstracted as components (e.g., a motor or a gripper component [9]), making adaptation reasoning discrete and symbolic in most cases.

Cyber-physical systems (CPS) add physical phenomena and interactions, such as power and timing, which makes adaptation difficult. If physical dynamics are not accounted for properly, the models may fail to predict the outcome of adaptation, potentially making an adaptive system less effective than a non-adaptive one. For example, in a robotics context if a decision is made to reconfigure the software of the robot to get better accuracy in localization of the robot without considering physical characteristics like increased power consumption of the hardware and software, then the robot may not have enough power to achieve its task.

Fully managing CPSs requires using models of all parts of the system - the cyber (software) *and* the physical models. In trying to incorporate physical models into adaptation, we face several challenges. First, we need to pick appropriate formal representations for physical models. The models need to be sufficiently expressive to represent the target behavior of the system and the environment. Different formalism choices may enable or disable certain kinds of analyses and affect possible adaptations. Another challenge is obtaining actual instances of physical models. They cannot be obtained simply by abstraction of software or by design (as is the case for cyber models) - often they require some experimentation and data collection to build. The final challenge is connecting physical models with proper adaptation layers and MAPE loops. Many physical models (e.g., power) have a systemic nature and can apply to many decisions about software. Therefore, an important choice in the design of self-adaptive cyberphysical systems is what parts read and write such physical models. Also, the relationship with cyber models is not clear - does one only adapt cyber models, or is it an option to adapt physical models as well? E.g., constraining the search space, or being as first-class as cyber models? How does this relate to functional/behavioral vs quality/configuration adaptation?

In this paper, we first present an example that illustrates several of these challenges. The example relates to our work of obtaining power usage model for a service robot based on the TurtleBot platform. We then discuss each of the above challenges in detail. The rest of this paper is organized as follows. Section 2 surveys existing work on model-based self-adaptation, focusing on work in the self-adaptive robotics domain. Section 3 presents an example that illustrates several difficulties that arise when physical models are used for implementing self-adaptive behaviour. In particular, it relates our work in building a service robot that adapts its mission in accordance with its remaining battery capacity. Section 4 contains a detailed discussion of the challenges associated with using physical models in combination with software models in CPS.

## II. Related Work

Models at run time have been mostly used to guide self-adaptation or verification. In the self-adaptive community, the models have focused on various models of the software that is being adapted (e.g., requirements, goals, architecture, tests). There has been some work where these models are applied to example systems that can be considered cyberphysical systems.

There has been quite a lot of work on applying self-adaptive techniques to robotics systems, which have physical characteristics such as batteries, physical movement, interaction with humans, etc. The works of Sykes et al. [3], [8], [9] include examples from the domain of robotics, but the models that are used at run time focus on the architecture or structure (both behavioral and extra-functional) of the system and the tasks of the system. Physical models are limited to simple models of resources and environment. The most sophisticated model of the physical aspects of the robots are described in [9], which describes an approach for updating environment models of a robot which conforms to Gat's three-layer architecture [10]. The reasons for which an environment model might have to be updated include unmodeled conditions, probabilistic factors, unmodeled transitions and unmodeled historical dependencies which manifest as failure of the robot to reach its goal. The model is encoded as a logic program that describes the actions that can be taken when certain conditions hold. To achieve a goal, this model is used to generate a plan. As the robot executes different plans, it collects execution traces for each of them. When a success is observed, a set of hypotheses that might explain it are generated. The traces are then examined to determine the most likely hypothesis. The model is then updated with a probability value that states the likelihood of success when the conditions in the hypothesis are true. This updated model is used by the planner to generate plans which lead to a higher probability of success. Many of the approaches are multi-model in that the represent and manipulate models of tasks or policies (e.g., [11]) of a robot, in addition to the software architecture. However, these models are not physical models.

Physical models are used in [12] to reason about optimal configurations of software in an unmanned underwater vehicle (UUV). In this work, the software configuration has to be sensitive to both the speed of the UUV and the amount of power that sensors can use, while fulfilling requirements on the number of measurements that should be taken during a mission. They use some modifications of run-time quantitative verification [13] to verify that quality attributes are being met and to choose appropriate reconfigurations if not. The quality attributes include physical characteristics (such as power consumption).

None of the approaches describe a methodology or framework for treating physical models as first class alongside software models.

## III. Motivating Example

We will illustrate physical modeling challenges on the example of TurtleBot [1] – a personal robot that can be used for indoor delivery, notification, and escort tasks. TurtleBot is equipped with a Kinect depth camera to visually sense its surroundings, navigate and avoid collisions with obstacles in its path. Control software is based on the Robot Operating System (ROS), which in turn executes on an on-board computer – the Intel Next Unit of Computing (NUC) [2] – that runs a Linux OS. TurtleBot moves using a two-wheeled base capable of driving in straight lines and arcs, and turning in place. The base also houses a battery that powers all system devices.

Since it runs on a battery, TurtleBot also has a limited amount of power and thus cannot perform tasks of indefinite duration. Therefore, it can run out of power and shut down in the middle of a task, which is undesirable. To prevent such situations, offline and online power adaptation is necessary. The former would determine, before starting a task, the system configuration and an action plan that would satisfy energy constraints (e.g., not running out of power in the middle of the task). The latter would react to unanticipated changes (e.g., driving drained more energy than expected) in-progress and would change the configuration or the plan.

To enable power-aware adaptation for TurtleBot, two power models are necessary:

- A *power consumption model* that determines for a given task, how much energy it takes to accomplish this task. This model is characterized by accuracy and whether it predicts for the average, worst, or best case.
- A *power state estimation model* that determines for a given the battery voltage reading, how much energy remains in the battery. This model is characterized by accuracy.

Building a power consumption model requires knowledge of how much power each physical part of TurtleBot uses for a given task. From the power perspective, TurtleBot has three components: a base, a Kinect, and a NUC. The power consumption of the base depends on the motion type (driving straight and rotating drain different power), velocity (translational and rotational), the floor material (e.g., carpet or tile), and the slope. NUC power consumption depends on the amount of processing that the NUC does, determined by multiple parameters of vision and navigation algorithms (e.g, number of processes points in a sensed set of points provided by Kinect). Power consumption of the Kinect sensor is mostly fixed since it delivers fixed data.

Consider a scenario that we will use to illustrate the challenges. TurtleBot needs to navigate to a target spot

[1] http://www.turtlebot.com

[2] http://www.intel.com/content/www/us/en/nuc/overview.html

through a building. Two paths are available: a shorter path through a crowded hallway, and a longer path through a mostly empty hallway. Using the shorter path would lead to the robot driving with slower speed and potentially stopping to let humans clear the path, whereas for the longer path can be driven with a faster speed and fewer if any interruptions. Depending on the specific parameters of this scenario, either of the two paths may turn out to be energy-optimal and (independently) time-optimal.

## IV. Challenges

Our position is that all models (physical and software) should be treated equally in the running system for verification and adaptation. Existing techniques for models at run time tend to favor software models, but physical models also need to be defined, monitored, and manipulated in similar ways to software models. Furthermore, these models and analyses of them need to be integrated to provide a holistic view of the run time state of the cyberphysical system.

To achieve this vision, there are a number of challenges that need to be addressed. First, identifying and integrating the required models needs to consider the vastly different modeling characteristics between software and various physical domains. Second, deriving and defining the physical models is challenging because of uncertainty and evolution over time because of changes in physical properties. Third, physical and software models need to be adapted and consider in any adaptation decisions that need to be made on the system at run time.

In this section, we discuss these three major challenges, giving some insight into how we are addressing them in our research.

### A. Selecting Modeling Formalism

Modern engineering methods for cyber-physical systems offer a multitude of possible formalisms for physical aspects. These formalisms vary in degree of their mathematical sophistication. Some are based on differential equations (e.g., Modelica), often used to describe continuously timed systems, for instance, physical movement. Signal-flow models (e.g., Simulink) can be used to approximate and simulate differential equations. For systems with multiple modes, one can use variations of automata such as timed automata or probabilistic automata. Formalisms are usually accompanied by tool environments that support model manipulation and analysis.

Not all formalisms are, however, equally useful to model all physical aspects of a system. The chosen formalism needs to fit the purpose and context in which it is used. One of the issues is to do with expressiveness. For example, if power predictions for a wheeled robot should output real numbers, formalisms that only use integers (e.g., Promela) would not fit this purpose.

A formalism's expressiveness determines what objects and dynamics can be described. For physical models, important formalism properties include linearity, treatment of continuity, and the underlying mathematical theories (which constrain permitted functions and numbers). For instance, if a formalism is restricted to polynomial functions (e.g., the input language of KeYmaera [**?**]), it may be challenging or even impossible to accurately describe system dynamics that are traditionally modeled with transcendental functions (such as movement along a curve written down with trigonometric functions).

Merely modeling a physical system is rarely the final goal; rather, one usually aims to execute analyses on a model to provide some value in the engineering process. For the power model of TurtlebBot, the goal of analysis is predicting whether the robot has enough energy to complete a given task. That is an example of an online analysis, but many analyses are done offline (i.e., at design time). For instance, determining the center of mass of a physical structure is a common offline analysis. Formalism choice is often dominated by the tradeoff between expressiveness and feasible analyses: one aims to use the least expressive formalism possible, given that it captures the needed phenomena. Otherwise, if the formalism's expressiveness is high, analysis may be infeasible and costs of usage may be prohibitive due to complexity.

In addition to the requirement of different formalisms for different physical aspects, there is also the challenge that different physical characteristics require different expertise to model. For example, modeling the mechanical aspects of the TurtleBot requires understanding physical mechanics, whereas modeling power consumption requires electrical engineering expertise. As cyber-physical systems scale and incorporate different aspects of physicality, the likelihood that a domain expert who understands all the modeling characteristics of the system decreases.

These challenges lead us to the position that: *It is a fruitless exercise to develop a unified modeling formalism that covers all aspects of a cyberphysical system.* Rather, we have been exploring the approach that allows engineers to work within the formalism required for their domain and within their area of expertise, and to integrate those models through abstraction and relation. Abstraction factors out crucial commonalities of models into a simplified space (in a metamodel language, a universal format for all models traces, etc.). Once that space is constructed, some elements of models are related through it. That relation is checked for consistency properties on the structure [14] and semantics [15]. Furthermore, beyond integrating the models through abstraction and relation, the analyses that may be performed on the different models needs to be coordinated. We need to ensure that an analysis is never applied to a model that violates the assumptions of the analysis. Since such violation can originate from updating a model by

another analysis, analyses must be executed in the correct order. In [16] we developed an analysis integration approach that uses contracts (assumptions required for the analysis to be valid, and guarantees provided by the analysis) to specify dependencies between analyses, determine their correct orders of application, and specify and verify applicability conditions in multiple domains. In [17] we list some of the ways in which we can deal with the inevitable dependency loops that occur in complex cyber-physical systems with many models and analyses.

Because of the dynamics of a cyber-physical systems, the fidelity of the models, and uncertainty in operating environments, these models and relationships need to be maintained at run time, as we discuss in the coming sections.

### B. Obtaining Physical Models

Once a formalisms for modeling various aspects of a system are decided, they are used to build models for the specific system at hand. To use a physical model at run time successfully, the model has to rank well enough on several *model value factors* – characteristics of a model that determine how much value it provides in run-time adaptation. The concrete factors may vary by system, but we distinguish three categories of model value factors:

1) *Analytical power* – the capacity of the model to provide conclusions (estimates, predictions) with required precision, granularity, and horizon. Analytical power is in part determined by the formalism, and partially by the model itself. For instance, a power model learned from a single run of TurtleBot would not accurately predict another run's power consumption.
2) *Fragility* – dependency of the model's analytical power on the model's assumptions, and how difficult it is to carry over the model to a new context from the original target context. For example, if a power model does not easily carry over from one surface material (e.g., carpet) to another (e.g., tiles), it is said to be fragile with respect to surface material. If a model is to be used across multiple contexts, its fragility needs to be mitigated.
3) *Computational cost* – the amount of computation needed to perform the required analyses. Again, although a formalism sets the overall expectations of computational costs, the model itself may contribute to them. For example, a state model with redundant states may slow down its traversal by a model checker. Computational costs often correlate with time needed for analysis, which is often critical during a self-adaptive system's execution.

Ideally, requirements on these value factors would determine the approach to building a physical model. Unlike cyber models that are often created through abstraction and (re-)engineering the modeled software, physical models may be built with various approaches. For the purposes of this paper, we distinguish two ways of building physical models:

- From *first principles*: the phenomena are modeled directly using an appropriate physical theory. For example, a robot wheel may be modeled mechanically to derive the power it takes to move [18]. First-principles models may be calibrated to suit the target context, but the primary emphasis is on the theoretical abstractions.
- *Data-driven*: the phenomena are studied indirectly, by collecting data and creating statistical models of this data. Data-driven models do not necessarily use or reflect the physical theories about the reality. To build a model for TurtleBot power consumption from data, one could multiple experiments with repetitive movement tasks (e.g., spinning on the floor for 15 minutes) and use regression to extract the model from the experimental data [**?**].

It is challenging to align model building decisions to ensure that model value factors meet the requirements. Before a theoretical model is constructed or data is collected, it is difficult to tell what margins of errors and prediction horizons a model can tolerate. It is also hard to set an expectation regarding how a model would respond to context switches.

Our position is that *for physical models, approaches to building models significantly affect the resulting value of the model*. Therefore, model-building decisions need to be carefully considered in the light of model value factors. There is little formal guidance on making model-building decisions in this way, and more such guidance is needed.

We followed this approach in the case of robot power model by examining the implications of the two model-building approaches on the specific power model values. We concluded that predictions over a high horizon (up to an hour of operation) with relatively low precision (error up to 10% is acceptable) can be accomplished by a data-driven model. A first-principles model, on the other hand, could have highly precise and granular predictions on the order of minutes, but their error would accumulate for long horizons. Thus, a data-driven model was more preferable from the perspective of analytical power.

The fragility of our data-driven power model left much to be desired: we could not predict power consumption in new contexts (e.g., a different surface material) without additional data collection. However, we can reuse data collection protocols and equipment to rapidly re-build models as needed. Moreover, our team did not possess the expertise in electrical, mechanical, and chemical engineering to build first-principles models for wheels, motors, and the battery of TurtleBot – especially parameterized models that could be rapidly adapted to new situations.

Due to the discussed reasons, we chose to pursue a data-driven power model for TurtleBot. We collected power consumption data from multiple motor tasks, executed up to

30 minutes. Then this data was cleaned from outliers and fed into linear regression. Although the result was satisfactory, more guidance on how model building decisions affect model value would have helped streamline the decisions and perhaps build a higher-precision model. However, it is unclear whether this is the right approach. Having systematic processes for guiding model choice will be crucial going forward, especially as the size and complexity of these systems increases.

### C. Using Physical Models in Adaptation

Traditionally, self-adaptive behavior has been implemented by reconfiguring software components of a system. This is not enough for CPS: we need to make adaptation decisions regarding the physical aspects of the system, too. For instance, for TurtleBot, physical variables like speed can be adapted along with cyber variables such as the number of iterations in the localization algorithm.

The constructed physical models may be used in several ways to make adaptation decisions in the target system. First, and most commonly, they might be used to *estimate* the current state of the system. Second, physical models can be used for *prediction* of physical actions. For example, there are no on-board devices on the TurtleBot to measure the remaining battery capacity directly. But, the output voltage from the battery can be measured, and an estimate of the remaining battery capacity can be made based on that. Third, the models might be used to search for an adaptation strategy that satisfies the given constraints. In this case, the search needs to be sensitive to the relationships between the different models.

Aside from adapting the system, physical models themselves might have to be updated because of changing physical conditions. For instance, if the power model is consistently giving optimistic projections, it may be that wear and tear on the robot reduced the efficiency of movement. There are multiple ways to respond to this situation: (re-)learning, calibration, updating ontology, and so on.

Thus, using physical models in adaptation is a two-part challenge:

- How to organize decision-making to adapt the system based on cooperative use of its models?
- How to organize decision-making to adapt models based on their model value (as described in the previous subsection)?

Our position is that *physical models should be treated as first-class entities in adaptation, thus given equal importance as cyber models*. The value of each physical model needs to be tracked throughout the system execution, and the models should be adapted as needed.

The way we approach implementing this position is by giving the system self-awareness of some aspects of model-building, described in the previous subsection. Since the system needs to manipulate its models, it needs to operate under the same concerns as do the engineers that built the original version of the model. In TurtleBot's case, the robot should continuously monitor how good predictions are and evaluate the precision of the models, adapting when the precision becomes unacceptable. It is, however, not advisable to constantly improve the models due to the risk of overfitting.

## V. Conclusion

Physical models are as important as cyber models when it comes to model-based adaptation of cyber-physical systems. However, we face several significant challenges in using physical models in adaptation. One needs to carefully select a formalism. The method of obtaining the model influences the scope of applicability and the analyses we can run on the model. As our experience with power modeling suggests, sophisticated experimental protocols can help collect and abstract data properly. A set of guidelines for selecting formalisms and picking experiments would help develop custom models for systems.

Another set of challenges is associated with using physical models in adaptation. Contextual sensitivity, information hiding, and maintaining models at runtime are among the prominent challenges in adapting via physical models. Depending on the type of model, different model qualities need to be prioritized for successful adaptation.

We thus come to a conclusion that modern model-based self-adaptive systems lack in treatment of physical phenomena. To improve the state-of-the-art of self-adaptive CPS, first, new approaches are needed that focus on bringing together cyber and physical models – not blindly but by taking into accounts their characteristics and features. Furthermore, we envision model management environments that help construct, relate, and maintain heterogeneous CPS models more systematically, as opposed to having individual tools for each kind of model.

## Acknowledgment

The authors would like to thank Rick Goldstein, Joydeep Biswas, and Manuela Veloso for their assistance with setting up and operating TurtleBot technology.

This material is based on research sponsored by NSF under Grant No. 1560137 and by AFRL and DARPA under agreement number FA8750-16-2-0042. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation therein. The views, opinions, findings, conclusions, and recommendations contained herein are those of authors and should not be interpreted as necessarily representing or reflecting the official policies or endorsements, either expressed or implied, of NSF, AFRL, DARPA, of the U.S. Government.

## REFERENCES


[1] P. Oreizy, N. Medvidovic, and R. N. Taylor, "Architecture-based runtime software evolution," in *Proceedings of the 20th International Conference on Software Engineering*, ser. ICSE '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 177–186.

[2] S.-W. Cheng, D. Garlan, and B. Schmerl, "Architecture-based self-adaptation in the presence of multiple objectives," in *Proceedings of the 2006 International Workshop on Self-adaptation and Self-managing Systems*, ser. SEAMS '06. New York, NY, USA: ACM, 2006, pp. 2–8.

[3] D. Sykes, W. Heaven, J. Magee, and J. Kramer, "Plan-directed architectural change for autonomous systems," in *Proceedings of the 2007 Conference on Specification and Verification of Component-based Systems: 6th Joint Meeting of the European Conference on Software Engineering and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, ser. SAVCBS '07. New York, NY, USA: ACM, 2007, pp. 15–21.

[4] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg, "Models run.time to support dynamic adaptation," *Computer*, vol. 42, no. 10, pp. 44–51, Oct. 2009.

[5] N. Huber, A. Hoorn, A. Koziolek, F. Brosig, and S. Kounev, "Modeling run-time adaptation at the system architecture level in dynamic service-oriented environments," *Serv. Oriented Comput. Appl.*, vol. 8, no. 1, pp. 73–89, Mar. 2014.

[6] V. E. Souza, A. Lapouchnian, K. Angelopoulos, and J. Mylopoulos, "Requirements-driven software evolution," *Comput. Sci.*, vol. 28, no. 4, pp. 311–329, Nov. 2013.

[7] N. Bencomo, A. Belaggoun, and V. Issarny, "Dynamic decision networks for decision-making in self-adaptive systems: A case study," in *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 113–122.

[8] D. Sykes, D. Corapi, J. Magee, J. Kramer, A. Russo, and K. Inoue, "Learning revised models for planning in adaptive systems," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 63–71.

[9] V. Braberman, N. D'Ippolito, J. Kramer, D. Sykes, and S. Uchitel, "MORPH: A reference architecture for configuration and behaviour self-adaptation," in *Proceedings of the 1st International Workshop on Control Theory for Software Engineering*, ser. CTSE 2015. New York, NY, USA: ACM, 2015, pp. 9–16.

[10] E. Gat, "Artificial intelligence and mobile robots," D. Kortenkamp, R. P. Bonasso, and R. Murphy, Eds. Cambridge, MA, USA: MIT Press, 1998, ch. Three-layer Architectures, pp. 195–210.

[11] J. C. Georgas and R. N. Taylor, "Policy-based self-adaptive architectures: A feasibility study in the robotics domain," in *Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems*, ser. SEAMS '08. New York, NY, USA: ACM, 2008, pp. 105–112.

[12] S. Gerasimou, R. Calinescu, and A. Banks, "Efficient runtime quantitative verification using caching, lookahead, and nearly-optimal reconfiguration," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS 2014. New York, NY, USA: ACM, 2014, pp. 115–124.

[13] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-adaptive software needs quantitative verification at runtime," *Commun. ACM*, vol. 55, no. 9, pp. 69–77, Sep. 2012.

[14] A. Y. Bhave, B. Krogh, D. Garlan, and B. Schmerl, "View consistency in architectures for cyber-physical systems," in *Proceedings of the 2nd ACM/IEEE International Conference on Cyber-Physical Systems*, April 2011.

[15] A. Rajhans, A. Y. Bhave, I. Ruchkin, B. Krogh, D. Garlan, A. Platzer, and B. Schmerl, "Supporting heterogeneity in cyber-physical systems architectures," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3178–3193, December 2014.

[16] I. Ruchkin, D. De Niz, S. Chaki, and D. Garlan, "Contract-based integration of cyber-physical analyses," in *Embedded Systems Week*, 12-17 October 2014.

[17] I. Ruchkin, B. Schmerl, and D. Garlan, "Analytic dependency loops in architectural models of cyber-physical systems," in *Proceedings of the 8th International Workshop on Model-based Architecting of Cyber-Physical and Embedded Systems*, Ottawa, Canada, 28 September 2015.

[18] J. Morales, J. L. Martnez, A. Mandow, A. Pequeo-Boter, and A. Garca-Cerezo, "Simplified power consumption modeling and identification for wheeled skid-steer robotic vehicles on hard horizontal ground," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, Oct 2010, pp. 4769–4774.