High Assurance Controller of a Self-Balancing Robot

Amanda Voegtlin, Ethan Lew, Patrick Gmerek, Justin Patterson / Dr. Marek Perkowski / Sponsored by Galois Inc

OUR TASK

Our team was tasked with researching, designing, and formally verifying the control system of a Two-Wheeled Invert Pendulum (TWIP) robot. To achieve these tasks, we had to mathematically model the dynamics of the TWIP robot, select an appropriate control method, implement a controller in software from our model, and verify our implementation. For our implementation to be considered high assurance, we needed to continuously study our model, address any shortcomings, verify our improvements through model checkers and formal verification tools, and validate that improvements in simulation matched the observed dynamics of the TWIP robot.

MODEL

The robot consists of two wheels, two actuators, a chassis and a motion control unit. The kinematics of the robot were formulated by producing generalized coordinates that describe a configuration space. Next, the robot's nonholonomy was exploited by adding a Pfaffian constraint to the kinematics, which allowed for the robot's joints to be fully expressed by the robot's overall motion.

$$K = M_w v^2 + I_w \left(\frac{v}{r}\right)^2 + 2\left(M_w + \left(\frac{I_w}{r^2}\right)\right) d^2\omega^2$$

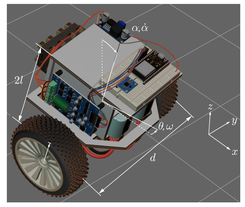
$$+ \frac{1}{2} M v^2 + \frac{1}{2} I_M \dot{\alpha}^2 + \frac{1}{2} I_p \omega^2$$

$$+ \frac{1}{2} m(v + l\cos(\alpha)\dot{\alpha})^2 + \frac{1}{2} m(-\sin(\alpha)\dot{\alpha})^2$$

$$U = mgl\,(1 - \cos(\alpha))$$

Dynamics were derived using the Lagrange-Euler formulation. This is done by find the Lagrange equations for the system.

The dynamics yield state space with five state variables of interest, three of which matter for stability analysis: tilt angle, tilt angular rate and forward/backward velocity.



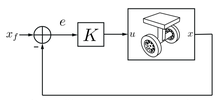
Robot Model Parameters.

The equations for a DC motor are given by

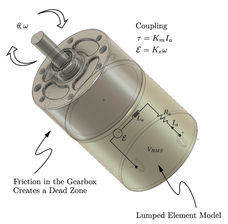
$$\begin{cases} V_{RMS} = I_a R_a + L_a \dot{I}_a + K_e \omega \\ \dot{\omega} = \frac{K_m I_a}{J} - \frac{b\omega}{J} \end{cases}$$

where we assume the motor shaft is rigid and the excitation flux is constant.

DESIGN



A simplified feedback diagram with a control matrix.

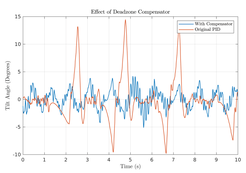


Motor Model Parameters.

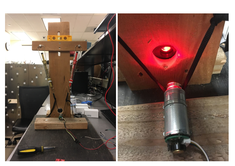
Due to inherent non-linearities in controlling a moving robot, we found that using a Jacobian matrix would linearly approximate the dynamics of our system. As a result, a digital PID controller was implemented with weights of 6.7 (K_p), 65.6 (K_i), and 0.171 (K_d).

By taking the next step and modeling our components mathematically, we were able to improve our simulations as well as the performance of the real-world robot. By implementing improvements to the basic PID controller, we were able to overcome things like derivative kick, motor dead zone, steady state oscillation, and motor saturation. Using sensor data from a gyroscope and accelerometer, our controller determined a balance point for its desired setpoint, and then torque was delivered to the motors to achieve the setpoint. A bluetooth joystick steered the robot by changing the setpoint, and the motors balanced it as it changed. Our controller was sensitive to pitch and yaw, and had the ability to correct the robot's facing as well as its tilt.

VALIDATION



Comparison of performance with and without deadzone compensation.



Prony Brake setup (left) and phototachometer used to measure the motor speed (right).

Validation was performed chiefly with comparisons of telemetry; the output of the simulator model and the output of the actual hardware, and changes in those outputs as adjustments to the controller were made.

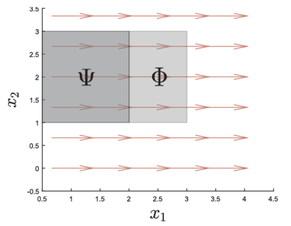
Using the telemetry dashboard, the PID weights were tuned using the Ziegler-Nichols method.

FORMAL VERIFICATION

Model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model.

CTL-A (COMPUTATION TREE LOGIC FOR ANALOG)

Computation Tree Logic (CTL) is a branching time logic that is based on path quantifiers and temporal operators. An extension to the language, analog operators, add inequalities that can describe regions of a dynamical system's state space.



$$\Phi = (x_{(1)} > 2) \wedge (x_{(1)} < 3) \wedge (x_{(2)} > 1) \wedge (x_{(2)} < 3)$$

$$\Psi = \mathsf{EF}\,(\Phi)$$

CTL-A allows for a rigorous description of common verification questions, namely

Safety	Does instability exist?
Liveness	Can the system stabilize from a condition?
Fairness	How can stability repeatedly occur?
Real-time	Does the stability occur fast enough?

These problems all address the stability problem in different ways. Safety addresses whether the robot is globally asymptotically stable (GAS), whereas liveness determines the region of attraction that results in a stable robot. Fairness addresses what conditions could occur where stability cannot be achieved again.

CTL is well-suited to address discrete finite automata, so a system discretization method must be created to utilize this verification technique.

SYMBOLIC DYNAMICS

Symbolic dynamics relates a discretized dynamical system to a sequence of abstract symbols corresponding to system state. Transitions between states are created from a shift operation that arises from system evolution.

Symbolic encoding allows continuous time, continuous state behavior to be characterized by a labeled transition system (LTS). LT systems are commonly used by verification tools and allow analog and dynamic behavior to be verified by tools designed for digital systems.

The Ikeda Map with its Chain Recurrent Set Components Labeled

Structural Graph of Ikeda Map

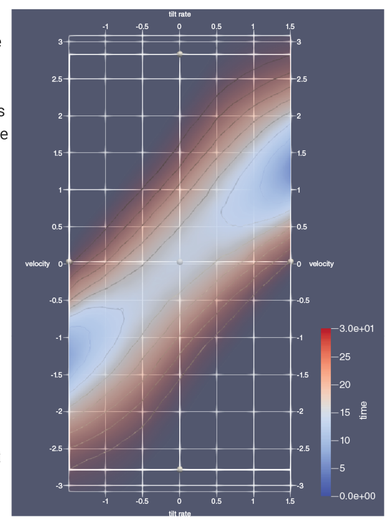
$$\begin{array}{c|cccc} & \infty & H & A & F \\ \hline \infty & 1 & 1 & 1 & 1 \\ H & 0 & 1 & 1 & 1 \\ A & 0 & 0 & 1 & 0 \\ F & 0 & 0 & 0 & 1 \end{array}$$

Structural Matrix of Ikeda Map

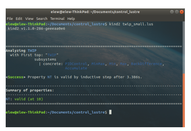
CONTROLLER VERIFICATION

Being able to associate coverings in state space to a region of attraction means that there exists a volume which contains all stable points. Assume all other state variables are at rest, verification found that the robot build should be able to stabilize within **26 degrees of the equilibrium point.**

Velocity and tilt angular rate affect the robot's ability to stabilize. In fact, there is found a large region where the rates can be sizable, but because of their direction, are still controllable.



The stability convergence time between linear velocity and tilt angular rate. Note that the robot has enhanced stability when the tilt angular rate and velocity are similar, able to converge quickly to the stable region.

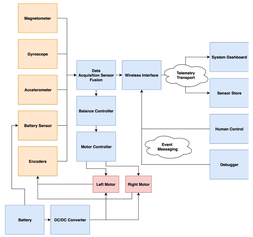


A SMT-based automatic model checker called Kind2 was used to the system. As Kind2 doesn't support transcendental functions, the system was partitioned into covering and a linear system was assigned to each state

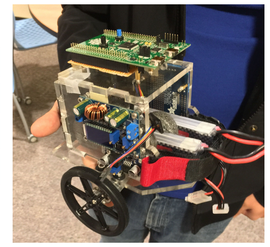
MATLAB/Simulink was utilized as a design, simulation and verification tool. The robot dynamics were added to a Simulink block and the controller design workflow can be performed after linearizing the feedback path.

BUILD

Rendering of the acrylic chassis.



Block diagram of the build.



Assembly of the acrylic robot, unwired.

The cardboard robot, our first prototype, followed the philosophy of rapid-prototyping; it was constructed quickly out of cardboard and used components that were already on-hand. This robot used an ESP32 as the processor and the controller was implemented in C. It also included a Bluetooth module to provide wireless telemetry and remote control.

Our second robot was made from acrylic and was modeled in AutoDesk Fusion360. The rough dimensions of the chassis were derived from the first robot. The specific dimensions for the mounting holes of the Buck/Boost converters, motor driver, and fusebox were determined by reviewing the part datasheet (where applicable) or by measuring the components. Once the modeling in Fusion360 was complete, the panels were exported to .dxf files and cut using a laser cutter. To facilitate easy assembly and disassembly, the edges of the chassis panels were joined by means of fingerjoints and hot glue. This provided a sturdy-yet-reversible bond between the panels.

CONCLUSION

The primary accomplishment of this team was implementing a way to use tools designed for digital system verification to verify hybrid systems. This was accomplished by discretizing the continuous time, continuous state behavior using symbolic encoding. The results extracted using symbolic methods closely aligned with brute force simulation. This capstone project represented a major research project for all members. In addition to verification research and documentation, our team succeeded in creating a functioning TWIP robot that can be remotely controlled, and implemented a PID library in Rust. We learned to how to load, debug, and emulate embedded Rust code for the STM32 Discovery microcontroller, as well as write code in the Lustre, Haskell, Python, C and MATLAB programming languages. Furthermore, a parameter-driven simulator was developed to show the robot dynamics as the controller, model, and hardware evolved. Further work is needed to develop an embedded Rust, verifiable, nonlinear controller to be compared against our C-based, modified PID controller.

ACKNOWLEDGEMENTS

Dr. Marek Perkowski and the Maseeh College of Engineering and Computer Science, for overseeing the project. Michal Podraedsky and Matt Clark at Galois Inc, for giving us this project and advising us along the way. Github and all the many authors of Rust, for their many tutorials and documention.

REFERENCES

Jorge Aparic, A self-balancing robot coded in Rust, (2018), Github Repository, https://github.com/japaric/zen

Stefan Kroboth, Rust on a Microcontroller, (2018), Github Repository, https://github.com/stefan-k/STM32F303DISCOVERY.

A. Greig, A. Richter, J. Munns, J. Pallant, D. Egger, The Embedded Rust Book, (2019), Github Repository, https://rust-embedded.github.io/book

[Various Authors], The Cargo Book, (2019), Github Repository, https://doc.rust-lang.org/stable/cargo/index.html

[Various Authors], Template to develop bare metal applications for Cortex-M microcontrollers, (2019), Github Repository, https://github.com/rust-embedded/cortex-m-quickstart

[Various Authors], Generate Rust register maps (`struct`s) from SVD files, (2019), Github Repository, https://github.com/rust-embedded/svd2rust

[Various Authors], Jorge Aparic, Board Support Crate for the STM32F3DISCOVERY, (2019), Github Repository, https://github.com/japaric/f3

Rust By Example, https://doc.rust-lang.org/rust-by-example/

Crate: Cortex-M-Quickstart; A template for building applications for ARM Cortex-M microcontrollers, (2018), github repository, https://rust-embedded.github.io/cortex-m-quickstart/cortex_m_quickstart/

Brett Beauregard. *Improving The Beginner's PID Controller*. Project Blog. http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/. Accessed May 02, 2019.

S. N. Vukosavic, *Digital Control of Electrical Drives*. Springer, 2017.

G. E. Fainekos, S. Sankaranarayanan, K. Ueda, and H. Yazarel, "Verification of automotive control applications using s-taliro," *American Control Conference*, 2012.

H. Abbas, G. Faniekos, S. Sankaranarayanan, F. Ivancic, and A. Gupta, "Probabilistic temporal logic falsification of cyber-physical systems," 2011.

T. Kahsai, P. L. Garoche, H. Bourbouh, C. Pagetti, T. Loquen, and E. Noulard . (2017) Cocosim. [Online]. Available: https://coco-team.github.io/cocosim/

Baier, C., & Katoen, J. P. (2008). *Principles of model checking*. MIT press.

Li, Z., Yang, C., & Fan, L. (2012). *Advanced control of wheeled inverted pendulum systems*. Springer Science & Business Media.

Osipenko, G. (2006). *Dynamical systems, graphs, and algorithms*. Springer.

Drechsler, R. (Ed.). (2004). *Advanced formal verification* (Vol. 122). Norwell: Kluver Academic Publishers.

Kevin M., Lynch, & Park, F. C. (2017). *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press.

Team 26, ECE Capstone 2019!

Portland State UNIVERSITY

Maseeh College of Engineering and Computer Science