Maseeh College of Engineering and Computer Science

PORTLAND STATE UNIVERSITY

HIGH-ASSURANCE CONTROLLER OF A SELF-BALANCING ROBOT

PRODUCT DESIGN SPECIFICATION

Version *1.0*

01/26/2019

A. Voegtlin, E. Lew, P. Gmerek, J. Patterson

VERSION HISTORY

Version #	Implemented By	Revision Date	Approved By	Approval Date	Reason
1.0	*All*	*1/26/2019*			Initial Design Definition draft
1.1	*P. Gmerek*	*2/15/2019*			Added feedback from the 2/8 meeting with Matt and Michel
1.2	P. Gmerek & A. Voegtlin	3/2/2019			Added details about verification tools and design verification theory

UP Template Version: 12/31/07

TABLE OF CONTENTS

1 INTRODUCTION

1.1 PURPOSE OF THE PRODUCT DESIGN SPECIFICATION DOCUMENT

The primary purpose of this document is to outline the requirements and stretch goals of the High Assurance Controller of Self-Balancing Robot capstone project. This is a living document and will be periodically updated throughout the course of the project. The deliverable of this project is a complete design process for creating a robust software-based controller for a self-balancing inverted pendulum robot.

2 GENERAL OVERVIEW AND DESIGN GUIDELINES/APPROACH

This section describes the principles and strategies to be used as guidelines when designing and implementing the system.

2.1 ASSUMPTIONS / CONSTRAINTS / STANDARDS

We develop here a list, in no particular order, of things the design of our robot "must" include, things it "should" include, and then things that would be "nice to have" if there is time. In this way we attempt to isolate the most important underpinnings of our proposed design.

Our design…

MUST

- Be well-documented with a repeatable design process, especially with respect to hardware
- Be formally verifiable[1]
- Be an inverted pendulum robot
- Be able to measure its improvement over a "simple PID" controller
- which balances on two wheels unassisted
- It will utilize the embedded Rust programming language for its principle control program,
- Investigate different model checkers such as Kind2 or Alloy
- Output telemetry data to enable debugging
- Implement PID controller and document accuracy/performance characteristics
- Have all components designed and implemented with verification in mind

SHOULD

- Use Octave as a mathematical model checker
- Be wireless

[1] This requirement is, by nature, very broad. It will be refined as the verification processes are developed and verification tools are investigated.

- A controller that is not PID-based in the prototype, for comparison to other control types
- Explore the applicability of alternatives to PID such as H-Infinity Control, Back-stepping, LMI Based Infinity Control, Non-Linear Controllers, Intelligent Control, Neural Network Feedback Control, for the final version of the robot controller
- Abstract I/O so that switching boards is not difficult
- Perform better in simulation than a standard model (such as a Segway, or similar) in terms of ability to adapt to environmental changes and sudden shocks
- Be able to wirelessly transmit its telemetry
- Use a microprocessor that shares characteristics with the HighFive1 or STM32 processor family

NICE TO HAVE

- Image tracking
- Turning
- Remote controllable while still being self-balancing

3 ARCHITECTURE DESIGN

This section outlines the system and hardware architecture design of the system that is being built.

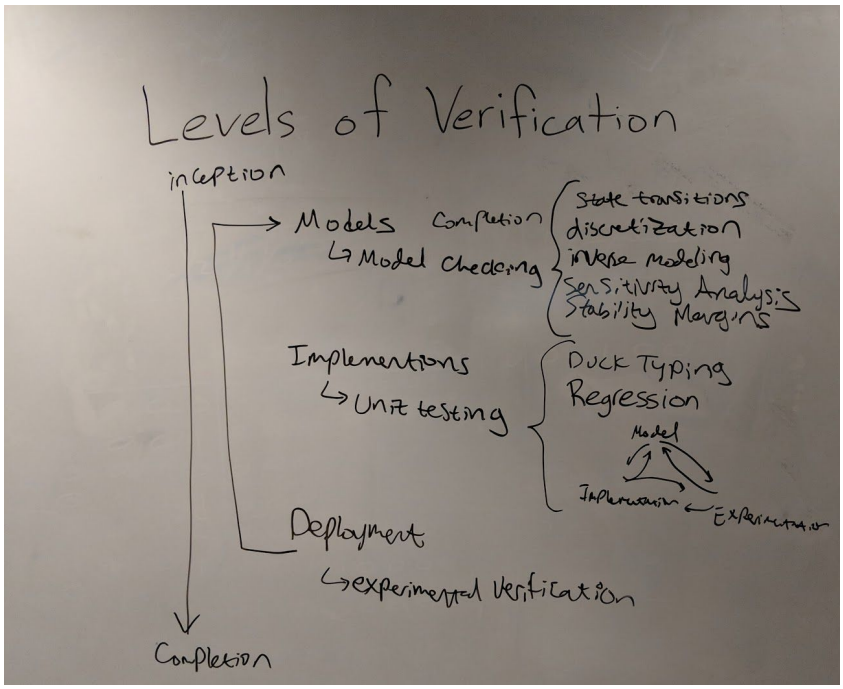
3.1 DESIGN METHODOLOGY

Given that the verifiability is the most important aspect of this project, our design methodology consists of describing a TWIP as a system, investigating the interaction of the forces within this system, and to make a controller based off of these properties.

From the image below, there are primarily three levels of verification: model, implementation, and deployment. In this project, we will primarily be concerned with the model and implementation stages of verification.

Since we already know the properties of our system from our investigation, we can verify our each component using a model checker. For instance, we could verify our motor model independently of the rest of the system and ensure that this aspect is performing as we expect. Knowing that our motor model is safe, we can integrate it with the rest of the subsystems and develop a hierarchical interactions of models that represent our robot. Furthermore, once we develop a continuous mathematical model of our system and subsystems, we can then discretize our model, as components of our system are discrete in nature.

After we refine the models of our components, and subsequently the model of the aggregate robot itself, we will write the Rust source files. At this point, we will be at the implementation level of verification. This stage is primarily concerned with isolating the "Software Under Test" (SUT, henceforth) and analyzing the SUT's properties.



3.2 DESIGN VERIFICATION THEORY

The purpose of the first prototype is to enable rapid testing of a control program. Our approach will have two basic components: physical, and simulation.

On the physical side of things is the prototype robot. We interact with the robot by means of a handheld controller, which should provide an intuitive way to demonstrate the prototype's ability to self-correct its balance in a variety of turns, spins, speeds, and changes in velocity. A web-based dashboard to monitor robot data and telemetry (including tilt angle, change in self-sensed speed and direction, etc) should allow for easy interpretation of the robot's state, graphed over time.

On the simulation side of things is the very same controller applied to a simulated set of robot characteristics, meant to be mathematically as similar as possible to our real robot; there will be mathematical models of every component to (as closely as possible) simulate the expected behavior of a robot of given dimensions, weight, engine power, torque, wheel size, etc. The performance of the simulated robot and the telemetry data of the real prototype can then be compared for expected versus actual performance in synchronized graphs to observe how the simulation differs from the actual performance, to discover deficiencies in the model.

The challenge of verifying this system lies in selecting a verification method which allows us to verify all domains of the input signals and the controller's ability to respond to them in a reasonable amount of time and in a predictable way. To that end, our team will be investigating analog verification of non-linear continuous time-based systems; candidates include CTL*, Continuous Time Markov Chains, TTL and SAT-verification.

3.3 GENERAL HARDWARE LOGICAL OVERVIEW

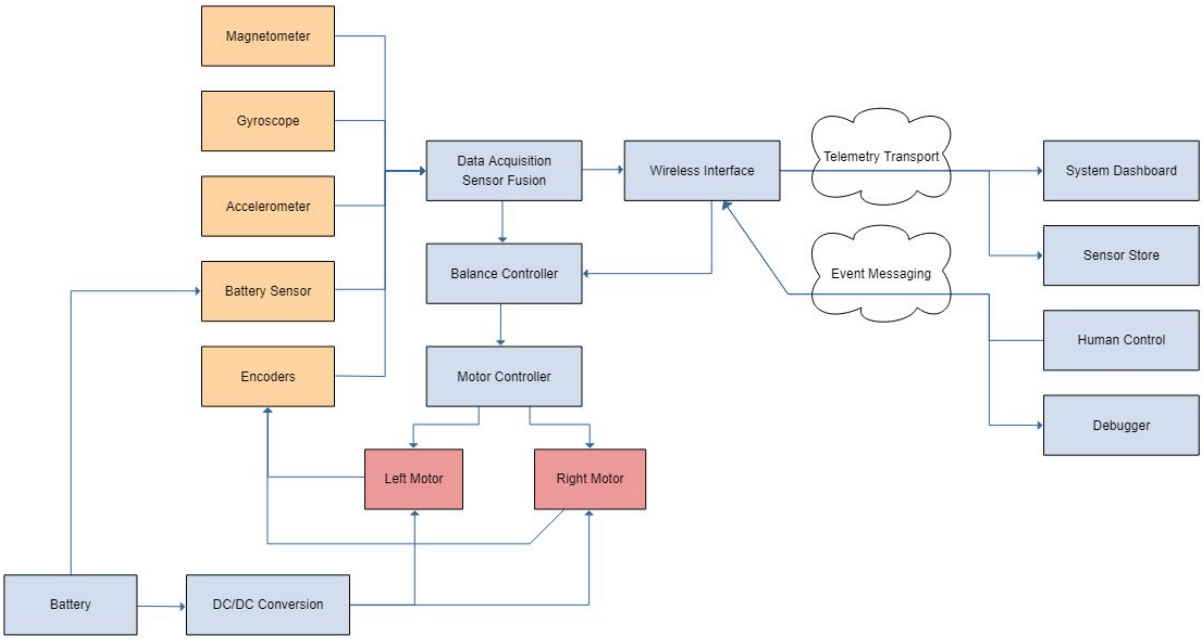
The L1 block diagram of the TWIP robot, shown at the end of this section, consists of 5 sensors and 2 actuators. The sensors critical for determining the error signal must return meaningful output (de-noised) at a frequency greater than 200 Hz in order to provide reasonable loop delay, thus the "at least 5 times per second" sampling requirement in our "must" list. Some of the sensors, like the encoders, are redundant and are present to be filtered through a sensor fusion algorithm in order to reduce the uncertainty inherent in readings. The other critical sensors, which reside within the IMU, include the magnetometer, accelerometer, and gyroscope.

The design of the two-wheeled inverted pendulum robot is an underactuated system, meaning that there are fewer control inputs than generalized coordinates. A control parameter we will use (which is often used in this kind of robot) is the "tilt angle", the angle between the payload orientation and the set point orientation. Sensing a tilt angle is not a trivial task, however, and various methods exist to achieve this. In its current state, the TWIP will utilize a 9 DoF Inertia Measurement Unit (IMU) in order to calculate tilt angle: 3 degrees from an accelerometer, 3 degrees from a gyroscope and 3 from a magnetometer. Traditionally, is is difficult to interpret gyroscope data due to the offset calibration error present--deviations of the output when the device is fixed at a single level. Further, gyroscopes can be permanently biased under the presence of a constant magnetic field, making it necessary to compensate for any magnetic distortion. Accordingly, a magnometometer's output can be factored in with a gyroscope's angle measurement to correct magnetic distortion; this process is known as sensor fusion.

The TWIP robot, capable of locomotion, will not be able source power from any external source, meaning that a battery is required onboard. Further, the battery should be able to supply power for the motors on the order of amps. A battery life of 2-5 hours is a reasonable target and should last the period of time to give several demonstrations. Power conditioning is also of critical importance as it directly affects the system's ability to be treated as time-invariant; identical pulse widths will achieve the same effect now as it will in several hours. Accordingly, a high efficiency DC/DC converter will be used to correct for the battery's variable voltage before powering the motors. Further, battery cost and maintenance isn't trivial, so the health of the devices should be considered. A battery sensor will alert the user when it's time to recharge and may help to troubleshoot power draw issues.

The support for event messaging and telemetry is essential as the robot is designed to further controller research. Further, such communication should take place over a wireless protocol as wiring the robot would tether it to a fixed range of motion. Raw sensor telemetry can be forwarded to help with the development and verification of sensor fusion algorithms. Data collected with these means can be stored in a time series database (TSDB) so it can be managed and used in the future. Some controller platforms, like the Simulink real-time kernel, even allow a device to use a wireless interface in the control loop itself. TSDBs track improvements as changes are made, and also provides an aggregate of all data. This has an adverse effect on the system loop delay. This architecture would allow for similar premise to be added later if needed.

Being able to change control strategies or controller parameters of the robot without rebuilding and re-uploading source code will make it easier to iterate over the design and modeling process. A wireless protocol capable of doing this is necessary to allow this to occur over a message queue. Also, if trajectory algorithms are added later, position commands can sent over the same queue. Lastly, in-circuit emulation (ICE) of the balance controller is possible with this setup. The processor running on the wireless interface can serve a debugger stub over a remote protocol, acting as a transceiver between the target hardware and the host computer. This provides a means for our programmers to debug the robot remotely and without the use of wires.



3.4 FIRST PROTOTYPE HARDWARE ARCHITECTURE

The initial hardware design of our practice prototype is heavily based on an

instructables tutorial. A link to this project can be found here. While similar, our plan deviates from this tutorial in several key areas. First, we will be incorporating a STM32 Discovery board as our controller because it's ability to run Rust code. Second, we chose different wheels since wheels that large aren't necessary and smaller wheels will require a more robust controller.

Any other deviations from the design will be made based on the availability of parts. The final prototype will use an updated parts list determined by the weaknesses of the initial prototype and should represent improvements in the design. The parts used will be outlined in a separate bill of materials.

3.5 SOFTWARE ARCHITECTURE

The controller of the robot shall be written in the Rust programming language to leverage the language's inherit memory safety and speed. The controller shall use code generated using Kind2's Rust code generation capability.

4 SYSTEM DESIGN

4.1 USE-CASES

A reliable controller for implementing two-wheeled pendulum robots across a variety of hardware implementations.

4.2 APPLICATION PROGRAM INTERFACES

The validation software needs to be easily reprogrammable to accommodate testing of the software controller on diverse hardware setups.

The controller software should be made portable such that it can run on different microprocessors and hardware setups.

4.3 USER INTERFACE DESIGN

Time permitting, adding the ability to interact with the robot by means of a handheld controller should provide an intuitive way to demonstrate our project. Furthermore, a web-based dashboard to monitor robot data and telemetry should allow for easy interpretation of the robot's state.

5 PRODUCT DESIGN SPECIFICATION APPROVAL

The undersigned acknowledge they have reviewed the Team 26 **Product Design Specification** document and agree with the approach it presents. Any changes to this Requirements Definition will be coordinated with and approved by the undersigned or their designated representatives.

Signature: _____ Date: _____

Print Name: _____

Title: _____

Role: _____

Signature: _____ Date: _____
Print Name: _____
Title: _____
Role: _____

Signature: _____ Date: _____
Print Name: _____
Title: _____
Role: _____