RapiBaBot: A Solution To The Inverted Pendulum Using a Raspberry Pi and its GPIO

Christian Micklisch and Hala ElAarag
Department of Mathematics and Computer Science
Stetson University
DeLand, Florida
{cmicklis, helaarag}@stetson.edu

***Abstract*— In this paper we present a manipulation of the Kalman Filter and Proportional Integral Derivative (PID) controller to generate an Inverted Pendulum. A Kalman Filter recursively receives noisy data and produces an estimate of the system's state. A PID Controller determines the rate at which the system reaches the set state from the derived estimate. An Inverted Pendulum, an inherently unstable system, attempts to sustain an upright position. The utilizations of the Kalman Filter in conjunction with the PID Controller will allow the system to maintain that position. While many Inverted Pendulum applications exist, in this research we propose the RapiBaBot; a robot that employs low-cost, easily obtained electronics to formulate a fairly stable inverted pendulum system. In this paper we show how we solve issues that usually arise in inverted pendulum applications, namely, vibration and lack of mobility. These issues have to be solved effectively for the stability and mobility of the system. We also explain how to construct the Kalman Filter and PID Controller into an effective and understandable form for the electronics to easily integrate with.**

Keywords*—*Raspberry Pi, GPIO, Inverted Pendlum, Kalman Filter.

I. INTRODUCTION

Robotics has usually been an integral part of our world nowadays. It is keen to be utilized in the palm of our hands and the actions of our fingertips. It is all around us when we include the simple gadgets that make up our daily lives (thermostat, microwave, lighting, etc.). These gadgets use the basics of what robotics forms, sensor input for interaction and the controllers formulating a decision from the input.

The inverted pendulum problem is a statement in which an object with a high center of gravity right above the pivot point has to balance itself using sensory information. The RapiBaBot takes in an input from a sensor like a Gyro or an accelerometer (or both) and decides which way to turn its motors based off of that input, just as Huang, Wang, and Chiu had done for their inverted pendulum design [1]. This kind of design becomes useful when one needs to transport an object from one point to another receiving inputs along the way either to avoid or interact with the objects around it autonomously or via a user.

The purpose of creating the RapiBaBot, an autonomous robot capable of balancing itself, is to develop a prototype that utilizes a linear estimator and a linear controller and can be used in future projects that have the same needs. For the RapiBaBot, we use the Kalman Filter as our linear estimator, which is capable of estimating past, present, and future states of the model [2]. For our linear controller we use a PID controller, similar to the one used by Jia-Jun Wang, [3] reason being the controller has been thoroughly tested and proven to be a solid contender in the industrial world [4]. The problem with the already existing systems is that they have commercial like price tags associated with them. Our objective was to create a simple Inverted Pendulum that would focus on balancing itself and work around a budget that can easily be accessible for anyone to purchase. The price of such a system would allow students to learn about the algorithms that control Inverted Pendulums in a more interactive and cost effective manner.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 describes the hardware that is utilized in the process of creating a low-cost inverted pendulum. Section 4 explains the software application of the inverted pendulum. It also covers how we use the Kalman Filter in building a useable design for our RapiBaBot. It also explores the PID Controller and how we use it for the Inverted Pendulum. In section 5 we cover the issues we encountered during testing. Finally section 6 concludes the paper.

978-1-4799-6585-4/14/$31.00

II. Related Work

There have been many other projects that utilize the problem of the inverted pendulum in order to find a solution that fits their need. One such example is the basic two-wheeled robot named the nBot made by David P. Anderson [5]. The two-wheeled robot was originally a three-wheeled robot project in an attempt to balance an orange ball on top of a wooden pole. After a couple more revisions the balancing act was discarded for a plain two-wheeled robot capable of moving around and balancing itself. It allows itself to move forward and backwards by changing the set point of the system to stay in a specific tilt-like position, this then forces the robot to stay in constant motion and keep its balance [5].

The nBot itself has similar qualities to that of a Segway, which is a two-wheeled transportation vehicle. Introduced in 2001 the vehicle is capable of moving by allowing the user to control the set point of the system, letting the user tilt forward or backward to set the direction of the system [6]. The Segway uses a total of five different Gyroscopic sensors and two tilt sensors to control the balance of the system [6].

Another application, and a more impressive one, is the BallBot, a completely autonomous system capable of balancing itself on a ball using several different sensors to achieve its goal. The real goal of the BallBot is to have "mobile robots that are safe; dynamically agile and capable of graceful motion; slender enough to easily maneuver in cluttered, peopled environments; and which readily yield when pushed around" [7].

There have been several different approaches for the BallBot by several different Universities around the world. The first of which was Carnegie Mellon University and was controlled by four different spinning rods all at 90 degree angles from each other [8]. The Swiss Federal Institute of Technology in Zurich (ETH) utilized a similar design in which the BallBot would be controlled by a 3 wheel system, and the wheels themselves would be movable on the side to allow a completely frictionless motion when moving the BallBot forward using two moving wheels, and one stationary one [9]. The BallBot from ETH, named Rezero, was one of the first ever BallBot's that utilized and was able to perform all the capabilities that a BallBot ought to have, such as a safe, graceful, and agile movement [9].

III. The Hardware

For the creation of an inverted pendulum, one would need a sensor for detecting the change in the state of the system that the inverted pendulum is in, a way to move the system from one position to another in a specific fashion, and a method of calculating the different movements that the system has to undergo. One such method is calculating the using a PID controller to manipulate the system from the output of the Kalman filter.

A model B Raspberry Pi (as shown in Figure 1) is utilized for computation. It has a 700Mhz processor that is capable of reading the output from the sensor and computing the input for the drive system in a timely manner. The Raspberry Pi also contains a copious number of GPIO (General-Purpose input/output) in which one can easily read from or control a device using simple programs such as wiringPi [10] or i2c-dev library[11] . The Raspberry Pi is running the Raspbian [12] operating system

Figure 1: Raspberry Pi

We used a Pololu MinIMU v2 sensor as shown in Figure 2. The MinIMU is an inertial measurement unit (IMU) that has a Gyro, Accelerometer, and Compass [13]. This sensor can easily be integrated with the Raspberry Pi's GPIO and the i2c-dev software. The sensor receives an input from all three axes. Having both the Gyro and Accelerometer, allows to get the speed of the rotation from the Gyro, and the force of the system in motion [14]. These two will allow us to get the angle at which the system is currently in, and generate an appropriate response from that output.

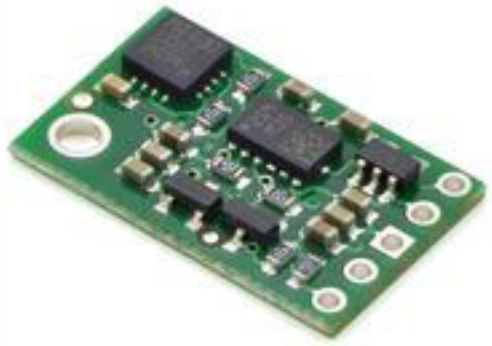

Figure 2: Pololu MinIMU v2 sensor

For movement of the inverted pendulum two NEMA -17 bipolar stepper motors (as shown in Figure 3) with about 0.4 N*m holding torque were usedThe motors could take about 500V for about a minute and needs a current of ~1.3A [15]. The motor system required a "high-voltage" setup, in which the minimum voltage would be that of ~6V. The drivers need to be able to take in that amount of current and not overheat to the point of melting (which was a slight issue). We used the Sparkfun EasyDriver Stepper Motor Driver [16], which allowed a minimum of 7V and a maximum of 30V, which fit our purpose fairly well. The drivers for the motors were hooked up directly to the GPIO to control the direction of the stepper motor and the steps that the stepper motor would take. All of this was done by utilizing the wiringPi library to easily set the pins on the Raspberry Pi to output pins, and then call

978-1-4799-6585-4/14/$31.00 ©2014 IEEE

out a high or low voltage depending on the scenario. Figure 4 shows a snapshot of the completed design of the RapiBaBot.

Figure 3: NEMA 17 Stepper motor

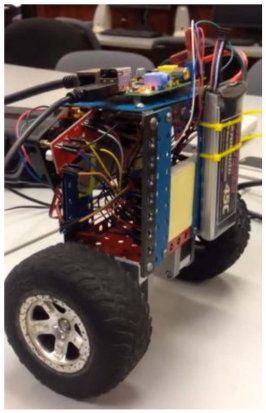

Figure 4: The RapiBaBot

IV. THE SOFTWARE

Our software implements two phases, the initialization and the balancing phases. The initialization phase allows the system to find its set point, which the system will maintain while running. The balancing phase postulates a decision based on that output to control its motors in a specific direction. For efficiency, the code for the balancing phase will first generate an output from the PID controller and then control the motors during the time the system is waiting for the sensor to get a new input.

The initialization phase is utilized to allow the system to gain a good understanding of where the set point should be once it starts to attempt to balance itself. During the initialization phase, the system simply executes a loop for a designated period of time (INIT_TIME) which needs to be tuned by the user. During this looping interval the system receives an input from the Gyro, feeds it into the Kalman Filter, which in turn produces an output to the PID controller. Finally the PID controller's output will be used as the systems set point for the remainder of its operation. A looping interval (LOOP_TIME) is used in order for the system to gauge accurate update to its positioning. This time would allow for the calculation of the angle related to the accelerometer. The pseudo code of the initialization phase is shown in Figure 5.

```
INIT_TIME = 4000;
LOOP_TIME = 20;
initializeTime();
while(totalRunTime() < INIT_TIME)
{
        startInt = getCurrentTime();
        acc_raw = readAcc();
        gyro_raw = readGyro();
        kalmanPredict(kalmanFilterInfo, gyro_raw);

        PID = calculatePID(kalmanFilterInfo);
        systemPoint = PID.getPID();

        kalmanUpdate(kalmanFilterInfo, acc_raw);
        while(getCurrentTime () - startInt < LOOP_TIME)
        {
                sleep(1);
        }
}
```

Figure 5: Initialization Phase of the RapiBaBot

The time it takes for the system to initialize is controlled by the amount of time it takes for the PID output to stabilize. For our RapiBaBot the best initialization time was around three to four seconds such that the system would be able to get a constant value from the Kalman Filter.

The balancing phase allows the RapiBaBot to properly base a decision on the output of the PID controller. The setup for the flow structure is similar to the initialization stage. The difference between the two is the output of the PID controller relates to the systems set point. A decision is then made during the waiting period as to whether the motors should spin in one direction or the other. The setup is structured so that the output from the PID controller will be then either decreased or increased as the motor is oscillated in either a positive or negative direction. This is to bring the output back to zero such that the motors will spin their desired amount. The balance phase of the code shares the same LOOP_TIME as the initialization phase. This is only to properly setup the set point. The pseudo code that is used to spin the motors is shown in Figure 6.

```
while(myMillis() - startInt < LOOP_TIME)
{
        if(output < 0)
        {
                spinMotorsForward();
        }
        if(output > 0)
        {
                spinMotorsBackward();
        }
        sleep(1);
}
```

Figure 6: Balancing Phase of the RapiBaBot

A. Kalman Filter

978-1-4799-6585-4/14/$31.00 ©2014 IEEE

The Kalman filter can be utilized to filter out some of the noise or random fluctuations that comes with any type of sensor. The algorithm can then be thought of as a continuous cycle throughout the runtime where two instances of the filter are used. The first instance is when the current state and error covariance are used to calculate the possible future instance of the state of the system. The second instance is when the state of the previous calculation is adjusted by the actual variables at that time. These two instances are called the time update and measurement update or prediction and correction moments, respectively [2].

i. Time Update

The instance of the time update or prediction moment is controlled by two equations. The first equation explains the current state of the system [2].

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \quad (1)$$

Where x_k is a function of its previous value from the correction moment, its control signal u_k, and the mean deviation w_k. A is the relation between the current and previous states. In our case it is a $2x2$ matrix. B is the relation between the control signal and the current state. In this case the B matrix is also a $2x2$ [2]. Equation 2 shows the calculation for the error covariance.

$$P_k = AP_{k-1}A^t + Q \quad (2)$$

Here P_k is the error covariance which is equal to its previous state multiplied by the relation of the previous state A, the time instance of that relation plus the process noise covariance or Q [2]. Usually the process noise covariance is calculated using the Auto-covariance Least Squares technique or ALS. The ALS technique is a "correlation between routine operating data to form a least-squares problem to estimate the covariance for the disturbances" [17]. In our design, the ALS method is computationally expensive for the hardware to calculate in a timely manner. Instead we used a self-tuned constant of 0.6 [18].

ii. Measurement Update

For the instance of the measurement update or correction moment it is necessary to find the Kalman Gain, the update in the error covariance and the new state of the system [2]. Equation 3 shows one of the three portions of the measurement update, the Kalman Gain.

$$K_k = \frac{P_k H^t}{(HP_kH^t + R)} \quad (3)$$

Where K_k is the Kalman Gain, which is a $2x1$ matrix, equal to the instance of the covariance error multiplied by the time instance of the $2x1$ matrix H (the relation of both the measurement noise and the state of the measurement) [2]. This is then divided by the error estimate, or the quantity of the relation of the measurement noise by its state, the time instance of that relation, and the error Covariance plus the measurement error covariance, R [2]. After computing the Kalman Gain, we compute x_k, the state of the system, as shown in equation 4.

$$x_k = x_k + K_k(z_k - Hx_k) \quad (4)$$

In this case x_k is equal to the solution found in equation 1 added by the Kalman Gain, which is multiplied by the measurement residual, as shown above. The residual is the "difference between the predicted measurement Hx_k and the actual measurement z_k"[2]. The actual measurement can be produced from the combination of both the multiplication of the predicted measurement and the measurement noise; v is the standard deviation of the measurement noise as shown in equation 5 [2].

$$z_k = Hx_k + v_k \quad (5)$$

After we conclude the above equations, what's left is to find the correction error covariance such that P_k is equal to the multiplication of the prediction error covariance. Then by the quantity of the constant I subtracted by the multiplication of the Kalman Gain by H. This is shown in equation 6.

$$P_k = P_k(I - K_kH) \quad (6)$$

ii. Utilization Of Kalman Filter

Now in the case of utilizing the Kalman Filter with the RapiBaBot we would need to implement the accelerometer and Gyro output into x_k [19]. From a programming point of view the basic calculations for the prediction moment would then be in equations 7 and 8. Where equation 7 represents the current state of the system, and equation 8 represents error covariance.

$$\begin{bmatrix} angle \\ bias \end{bmatrix}_k = \begin{bmatrix} 1 & -dt \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} angle \\ bias \end{bmatrix}_{k-1} + \begin{bmatrix} dt \\ 0 \end{bmatrix} \cdot u_{k-1} \quad (7)$$

$$\begin{bmatrix} P00 & P01 \\ P10 & P11 \end{bmatrix}_k = \begin{bmatrix} P00 & P01 \\ P10 & P11 \end{bmatrix}_{k-1} \cdot \begin{bmatrix} 1 & -dt \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} P00 & P01 \\ P10 & P11 \end{bmatrix}_{k-1} \cdot \begin{bmatrix} 1 & -dt \\ 0 & 1 \end{bmatrix}^{-1} + \begin{bmatrix} Q\_angle \\ Q\_gyro \end{bmatrix} \quad (8)$$

The calculations for the correction measurement are shown in equations 9, 10, and 11. Where 9 is the representation of the Kalman Gain K_k from equation 3, 10 is the representation of the recalculation of the current state of the system x_k as in equation 4, and 11 is the recalculation of the error covariance P_k as in equation 6.

978-1-4799-6585-4/14/$31.00 ©2014 IEEE

$$\begin{bmatrix} K0 \\ K1 \end{bmatrix}_k = \frac{\begin{bmatrix} P00 & P01 \\ P10 & P11 \end{bmatrix}_k \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix}}{\begin{bmatrix} 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} P00 & P01 \\ P10 & P11 \end{bmatrix}_k \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} R\_angle \\ 0 \end{bmatrix}} \quad (9)$$

$$\begin{bmatrix} angle \\ bias \end{bmatrix}_k = \begin{bmatrix} angle \\ bias \end{bmatrix}_k + \begin{bmatrix} K0 \\ K1 \end{bmatrix}_k \cdot \left(\begin{bmatrix} z\_angle \\ 0 \end{bmatrix}_k - \begin{bmatrix} 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} angle \\ bias \end{bmatrix}_k \right) \quad (10)$$

$$\begin{bmatrix} P00 & P01 \\ P10 & P11 \end{bmatrix}_k = \begin{bmatrix} P00 & P01 \\ P10 & P11 \end{bmatrix}_k \cdot \left(\begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} K0 \\ K1 \end{bmatrix}_k \cdot \begin{bmatrix} P00 & P01 \\ P10 & P11 \end{bmatrix}_k \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right) \quad (11)$$

The variables that stay constant are H and I. I itself was originally a constant of no effect in the original equation and H is also a constant, since we can ignore the relation of both the measurement noise and the state of the measurement. The matrix z_k is never really calculated out since it is a given value from the accelerometer. Last but not least the matrixes Q and R are initialized constants as well, and A is a matrix used to include the dt, or the LOOP_TIME, into the calculations of the angle in x_k.

B. PID Controller

In a regular control system you would have a state that the system is in, the set point of what the system should be in, and then the change that needs to occur at a certain rate when the state is not in a specific range of the set point [4]. A PID follows the same principle as the basic format of the control system, only using more aggregate functions to allow the system to bring itself to the set point in different environments, being the case of an autonomous self-balancing robot having different "wheel size and traction" [18].

In this instance our robot could actually utilize a basic control system, where it has a specific position in the environment and a designated position. This is actually a P controller, which calculates the difference between the set point and the actual point to see what adjustments are necessary for that occasion [18]. One problem that might occur is when the P controller overshoots its set point because of a large difference between the environment its in and the one it was calibrated for.

The solution then would be to calculate the rate of change in the difference of the set point and the actual point. This is where a PD controller comes in, the P portion checking the system for a change in the difference of the set point and actual point, and the D portion being the first derivative of P (proportional) [18].

All the system needs to do then is account for all of the errors that occur from D (derivative). These errors are the deviations, where D is either under or over its expected value. This is where I, or the integral, becomes important. I represents the "sum of all the errors over time," such that it creates the PID controller taking into account a change in its environment and adjusting to it in a fashion of recognizing the errors resulting from the difference. The final form of the PID controller is then the proportional change of the system with the rate of the change and the sum of all the errors in that rate shown in equation 12.

$$Output = K_p \cdot e(x) + K_i \cdot \int_0^x e(x)dx + K_d \cdot \frac{de(x)}{dx} \quad (12)$$

In the above equation x is the output of the Kalman Filter, and K_p, K_i, and K_d are the gains used to tune the system so that it may function in almost any environment properly.

V. TESTING

A number of tests were completed to test the system performance and the issues that arise during runtime. During operation the stepper motors created a vibration, which was then received by the sensor. This caused distortion in the output of the sensor that could not be filtered out by the Kalman Filter. To remedy this problem we placed some anti vibration reduction tape between the Gyro and the mounting place on the RapiBaBot.

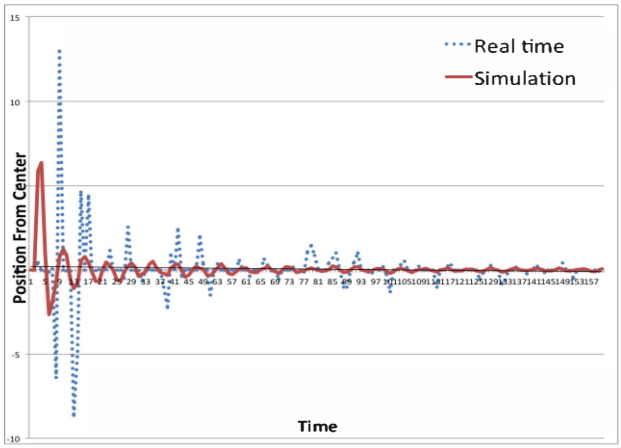

Figure 7: Results

After a certain period of testing we noticed that the RapiBaBot functions perfectly well as an inverted pendulum wherein it balances itself in an upright position autonomously. The result of this testing phase compared to its simulation is shown in Figure 7. In the test shown in Figure 7, the RapiBaBot was initialized and attempted to balance itself. The graph displays the change in closeness to the upright position over time, where the center of the RapiBaBot is at 0. In the test there were also moments where the RapiBaBot moved a bit off course, this might have been from the slight vibration caused by the motors.

978-1-4799-6585-4/14/$31.00 ©2014 IEEE

The RapiBaBot's tires allowed for good traction on almost any household surface. A flaw of the RapiBaBot is that it cannot smoothly travel across a set path. After a while it will start to tip over if it attempts to move in a specific direction at a quickening pace. This is an issue with the motors themselves as they can only turn at a set pace. A solution to this problem would be to use motors that can easily control their direction position and speed, such as servomotors, or to use stepper motors with more poles and current as ours only ran on 2.5 A on a bipolar system.

VI. Conclusion

In this research, we have designed a robot that utilizes a Kalman filter in association with a PID controller. The hardware used in our design is pretty affordable and the robot works fairly well as an inverted pendulum. In this paper we have shared our experience in designing this robot and provided details of the hardware and software used. Our system is not only useful for inverted pendulum applications, but also for any application that utilizes a sensor with Gaussian noise and needs to gauge an actual focal point and change it to a set point.

Acknowledgment

The authors would like to thank Kyle Campbell, Colin Carter, Ingolf Micklisch and Bill Ball for their contribution to this project.

References

[1] Cheng-Hao Huang, Wen-June Wang, and Chih-Hui Chiu. "Design and implementation of fuzzy control on a two-wheel inverted pendulum." Industrial Electronics, IEEE Transactions on 58.7 2011. 2988-3001.

[2] Greg Welch, Gary Bishop. July 24, 2006. *An introduction to the kalman filter*. Chapel Hill, NC: University of North Carolina at Chapel Hill, 1995.

[3] Jia-Jun Wang. "Simulation studies of inverted pendulum based on PID controllers." Simulation Modeling Practice and Theory 19.1 2011. 440-449.

[4] National Instruments. "PID theory explained". Internet: http://www.ni.com/white-paper/3782/en/, March 29, 2011 [November 27, 2013]

[5] David P. Anderson. "nBot balancing robot". Internet: http://www.geology.smu.edu/dpa-www/robo/nbot/index.html, September 14, 2013 [December 1, 2013]

[6] Internet: http://www.segway.com/business/products-solutions/i2.php [December 1, 2013]

[7] Tom Lauwers, George Kantor, and Ralph Hollis. "One is enough!." in *Robotics Research*, vol. 28. Springer Berlin Heidelberg, 2007. 327-336.

[8] Ankit Bhatia, Carl Curren, Mark Dudley, Jodi Forlizzi, Jared Goerner, Ralph Hollis, George Kantor, Kalicharan Karthikeyan, Byungjun Kim, Masaaki Kumagai, Tom Lauwers, Jun Xian Leong, Anish Mampetta, Umashankar Nagarajan, Suresh Nidhiry, Kenneth Payson, Kathryn Rivard, Eric Schearer, Michael Shomin, Bhaskar Vaidya, and Luo (Jack) Yi."Dynamically-stable mobile robots in human environments". Internet: http://www.msl.ri.cmu.edu/projects/ballbot/ [December 1, 2013]

[9] ETH Zurich. October 22, 2010. *Stability and dynamics – and this all on a ball?* Switzerland: Maxon Motor, 2010, pp. 1-6.

[10] Gordon Henderson. "Wiring Pi GPIO interface library for the Raspberry Pi". Internet: http://wiringpi.com/, July 2013 [November 27, 2013]

[11] http://www.i2cdevlib.com

[12] Internet: http://www.raspbian.org, 2013 [December 1, 2013

[13] Polulu. "MinIMU-9 v2 gyro, accelerometer, and compass (L3GD20 and LSM303DLHC carrier)". Internet: http://www.pololu.com/product/1268/, 2013 [December 1, 2013]

[14] Shane Colton. June 25, 2007. *The balance filter. A simple solution for integrating accelerometer and gyroscope measurements for a balancing platform.* [Online Paper] Available: http://web.mit.edu/scolton/www/filter.pdf [December 1, 2013]

[15] KEEDOX. "57oz-in 1Nm NEMA 17 stepper motor 1.3A 40mm for CNC router or mill" Internet: http://www.amazon.com/57oz-NEMA-Stepper-Motor-router/dp/B00C4P382G [December 1, 2013]

[16] Sparkfun. "EasyDriver stepper motor driver" Internet: https://www.sparkfun.com/products/10267 [December 1, 2013]

[17] Murali Rajaa Rajamani. 2007. *Data-based techniques to improve state estimation in model predictive control*. Ann Arbor, MI: ProQuest Informtion and Learning Company, 2008, pp. 34-42.

[18] Rich Chi Ooi. (2003, November 3). *Balancing a two-wheeled autonomous robot*. [Online Paper] Available: http://robotics.ee.uwa.edu.au/theses/2003-Balance-Ooi.pdf [November 27, 2013].

[19] José Luis Corona Miranda . (Spring 2009). *Application of Kalman filtering and PID control for direct inverted pendulum control*. [Online Paper] Available: http://csuchico-dspace.calstate.edu/bitstream/handle/10211.4/145/4%2022%2009%20Jose%20Miranda.pdf?sequence=1 [December 1, 2013]