APPLICATIONS OF
MODELLING AND SIMULATION

http://arqiipubl.com/ams eISSN 2600-8084 VOL 3, NO. 2, 2019, 64-73

A Tutorial on Modelling and Control of Two-Wheeled Self-Balancing Robot with Stepper Motor

Fabian Kung

Faculty of Engineering, Multimedia University, 63100 Cyberjaya, Selangor, Malaysia

*Corresponding author: wlkung@mmu.edu.my

Received 2 May 2019, Revised 3 June 2019, Accepted 4 June 2019.

Abstract: In this paper we describe an approximate mathematical model for a stepper motor based two-wheeled self-balancing (TWSB) robot. We show the usage of this model in computer aided design of a digital controller to balance and to steer the robot. TWSB robots can be implemented using geared DC motors and stepper motors. The latter version is popular in recent years after the appearance of open-hardware design for stepper motor driver module, which was originally created for desktop 3D printers and CNC machines, and the ease with which stepper motors can be controlled with pulsed electrical signals. Even though many individuals had shared the design and computer codes for TWSB robots with stepper motors, to date there is still a lack of mathematical model for such machines being reported, even in academic publications. Almost all reported works on stepper motor based TWSB robots rely on empirical or trial-and-error approach to obtain a suitable feedback controller for balancing the machine. This paper intends to fill the gap, and the model and techniques described will be useful for engineers, researchers, educators and others who are building their own machines.

Keywords: Digital control; Modelling; Robot; Self-balancing; Scilab; Stepper motor.

1. INTRODUCTION

Since the reported mobile two-wheeled self-balancing machine in [1], many parties had successfully brought this concept to smaller autonomous machines for serious research, for fun and for commercial applications [2-10]. The reader can refer to [4], which provided a nearly exhaustive review of the modelling and control methods for two-wheeled robots from year 2000 to 2012. Reference [4] also gave a detailed account of various two-wheeled robot configurations in tackling environmental challenges such as working with inclined or uneven surface, obstacles avoidance, controlling body roll to having manipulator or limbs. From 2011 onwards, there is an explosion of projects involving two-wheeled self-balancing robots in both academic, commercial and hobbyist domains, mainly due to the emergence of open-source and open-hardware movement, cheap and powerful embedded computing platforms, compact electric motors with reasonably priced solid-state motor drivers and motion sensors. Many people took to sharing their works on social media, bringing rapid improvement to the open-source computer codes for the on-board controller and algorithms. For the rest of this paper we will just use the term "TWSB robot" to refer to such machines.

Most of the robots reported in [1-10] used DC geared motors to provide locomotion to the machine, except for [8-9] which used stepper motors. Around 2011, TWSB robots using stepper motors for locomotion emerged and has been popular with many enthusiasts. References [8-9] are indicative of this trend, with many more examples reported online. Mathematical models for TWSB robots have been derived for machines using geared DC motors in academic publications [1,2,4,5]. However, to date it seems no detailed model have been proposed yet for TWSB robots using stepper motor, with the works reported in non-academic settings usually relied on empirical or trial-and-error approach to find the controller coefficients for the feedback control. Executing secondary motions, such as linear velocity control (i.e. making the robot move forward/backward at constant speed), steering (i.e. turning left/right) while balancing upright have been described in [1-2] and works cited in [4], again for DC motor based two-wheeled robots. There is also a lack of detailed discussion of the characteristics of executing secondary motions for stepper motor based TWSB robots. Therefore, this paper intends to fill the gaps by:
(a) Providing an approximate linear model for the TWSB robot platform with stepper motors.
(b) Proposing techniques for executing secondary movements and steering.

To support our idea, we have built a basic stepper motor TWSB robot as shown in Figure 1. The custom on-board computer of our test robot uses a 16-bits DSP (digital signal processing) capable micro-controller running at 140 MHz clock frequency [11].

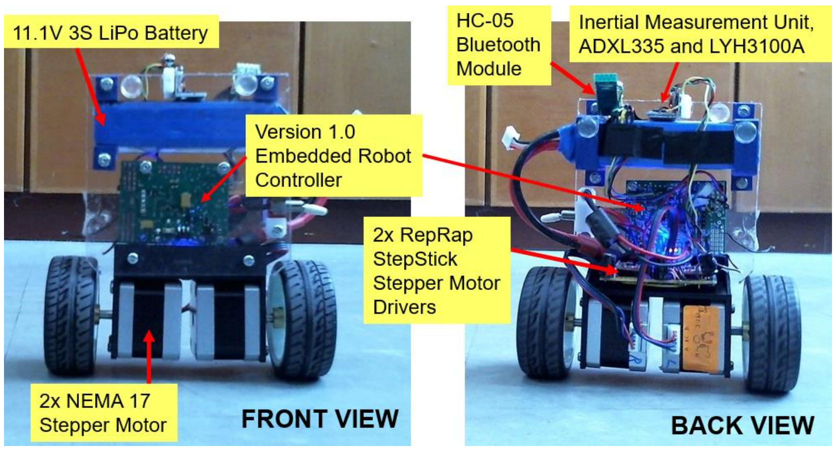

11.1V 3S LiPo Battery
Version 1.0 Embedded Robot Controller
2x NEMA 17 Stepper Motor
FRONT VIEW
HC-05 Bluetooth Module
Inertial Measurement Unit, ADXL335 and LYH3100A
2x RepRap StepStick Stepper Motor Drivers
BACK VIEW

Figure 1. Front and back views of the stepper motor based TWSB robot

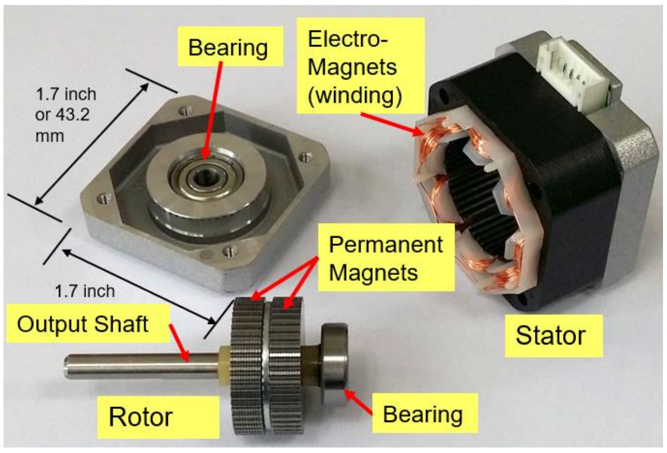


Figure 2. Disassembled NEMA-17 size bipolar stepper motor

2. MODELLING

2.1 Simplified Stepper Model

Construction and operating principles of the stepper motor can be found in [12-13]. Typical stepper motor has high output torque and can be attached directly to a wheel without the use of a gearbox. Here we will focus on two-phase (i.e. two windings) bipolar stepper motors as oppose to unipolar (i.e. single windings) type stepper motors for reasons of compactness and higher power efficiency. Figure 2 shows a disassembled NEMA-17 size stepper motor popular with small TWSB robots. The holding torque at the output shaft generally depends on the construction and current rating of each winding. Typical current rating per windings is 0.5 A to 2.0 A, with output shaft holding torque ranging from 30 N·cm to 60 N·cm.

To make a stepper motor rotates clockwise or anti-clockwise, electric current with the correct sequence needs to be injected into the windings. Usually the current in each winding is a series of steps resembling a sine wave, with the waveforms between winding 1 and winding 2 differs by 90^o in phase [12], [14-15]. An integrated circuit (IC) incorporating variable current sources and transistor switches is often used to drive the windings of the stepper motor. The driver IC takes an input in the form of periodic voltage pulses. With each pulse the correct sequence of electric current will be imposed on the windings, turning the stepper motor shaft a fix amount, typically 1.8^o for full-step or 0.9^o for half-step operation (1/4 to 1/32 micro-stepping modes are also possible). Usually the driver IC also contains another input that determines the rotation direction. This concept is illustrated in Figure 3 for a commercial driver IC A4988 [15] for half-step operation, with STEP and DIR being the inputs for step and direction control. For instance, assuming half-step operation, to rotate the output shaft of the stepper motor by 45^o, we merely need to send 50 pulses to the motor driver circuit. If these 50 pulses are sent within 1 second (50 Hz), we achieve a rotation speed of 0.125 revolution/second (or 45^o/second). If this is done within 0.25 second (200 Hz), we achieve a rotation speed of 0.5 revolution/second (180^o/second) as long as the load on the motor shaft is within the stepper motor 'Pullout Torque' limit. Thus, the rotation speed of the stepper motor output shaft is proportional to the frequency of the step control.

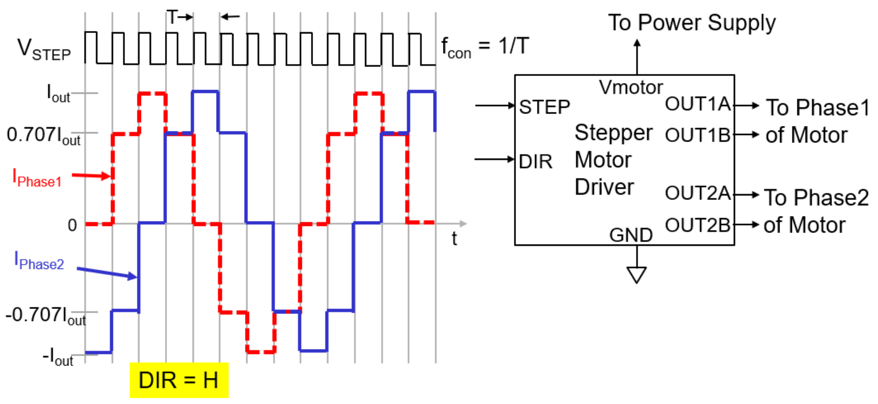


Figure 3. The relationship between input signal STEP (V_{STEP}) and winding currents for half-step (0.9°) operation, with DIR input set to logic high, for A4988 Stepper Motor Driver IC. For each V_{STEP} pulse the stepper motor shaft will rotate by 0.9°. When DIR input is set to logic low, Phase 2 current will lead Phase 1 current instead

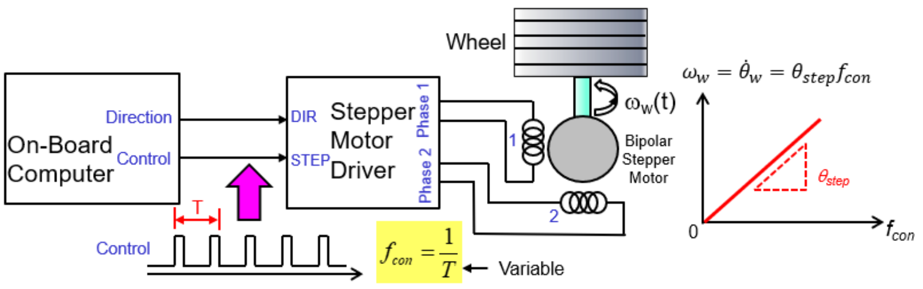


Figure 4. The first order input-output relationship for stepper motor.

Using the NEMA-14 and NEMA-17 size bipolar stepper motors from [12] as a benchmark, for slow to moderate rotational speed of below 600 rpm or 10.0 revolutions per seconds, the pullout torque of the stepper motor can be approximated as constant (the back EMF induced in the windings is not significant at low rotational speed). Most TWSB robots never require wheel rotation speed above 5 revolutions per second (300 rpm) to keep the robot in upright position, so this approximation is valid in most cases.

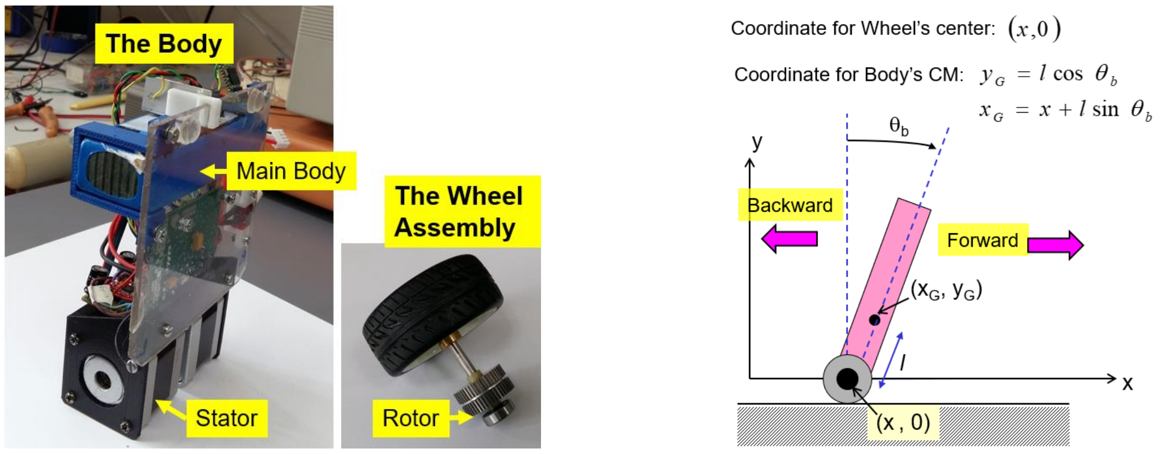
From the discussion above, it is thus helpful to consider the stepper motor to a first order a rotational actuator whose output torque is constant, and rotational velocity is proportional to the frequency of the control signal. Let θ_{step} be the step angle, and f_{con} the frequency of the voltage pulses applied to the STEP input, θ_w is the wheel angle and ω_w is the wheel angular velocity. The wheel is connected to the stepper motor output shaft. Then the input-output relationship for stepper motor can be approximated as:

$$\omega_w = \frac{d\theta_w}{dt} \cong \theta_{step} f_{con} \tag{1}$$

Step angle θ_{step} will be positive when rotating in clockwise direction and negative in counter-clockwise dircetion. This relationship is illustrated in Figure 4.

2.2 Robot Platform Transfer Function

The TWSB robot can be considered as a mechanical system with two coupled parts: (1) The Body, consisting of the main body and the stators of the stepper motors and (2) The Wheel Assembly, consisting of the wheels and rotors, as shown in Figure 5. The coordinate system and notation for the physical properties of our TWSB robot are also illustrated in Figure 5. Here θ_b is the angle made by the line intersecting the main body center of mass (CM) and wheel axle, with the y-axis. The CM for the main body can be determined using the bifilar suspension method. Mass can be measured using an electronic scale, while the individual component's moment inertia is calculated and parallel axis theorem is used to find the total moment inertia. Note that moment inertia J_w is referenced to the center of each wheel, and J_b is referenced to the rotational axis on CM.


The Body
Main Body
The Wheel Assembly
Stator
Rotor
Coordinate for Wheel's center: $(x,0)$
Coordinate for Body's CM: $y_G = l\cos\theta_b$
$x_G = x + l\sin\theta_b$
θ_b
y
Backward
Forward
(x_G, y_G)
l
x
(x , 0)

R	Radius of wheel	0.031	m
m_{wa}	Mass of wheel assembly	0.104	kg
J_w	Moment inertia of wheel assembly	0.0000241	kgm^2
l	Distance between wheel axis to CM of the body	0.04	m
m_b	Mass of the body	0.660	kg
J_b	Moment inertia of the body	0.001304	kgm^2
g	Earth gravitational acceleration	9.807	ms^{-2}

Figure 5. Physical parameters of the robot platform and the coordinate system

The wheel is assumed to follow non-slip motion. Derivation of the equations of motion can be found in [1] using Newtonian mechanics. What one needs to bear in mind is the form of the equations of motion will vary slightly depending on the coordinate system chosen. The inputs to the robot platform are the torque T_L and T_R applied to the left and right wheel assemblies, which are further assumed to be similar, e.g. $T_w = T_L = T_R$. Using the notation $\ddot{x} = \frac{d}{dt}\dot{x} = \frac{d}{dt}\left(\frac{d}{dt}x\right)$ and $\ddot{\theta}_b = \frac{d}{dt}\dot{\theta}_b = \frac{d}{dt}\left(\frac{d}{dt}\theta_b\right)$, we can derive the following time domain equations of motion:

$$\ddot{x} = \left[m_b + 2\left(m_{wa} + \frac{J_w}{R^2}\right) - \frac{(m_b l\cos\theta_b)^2}{m_b l^2 + J_b}\right]^{-1} \cdot \left\{\begin{matrix} (m_b l\sin\theta_b)\dot{\theta}_b^{\,2} - \dfrac{(m_b l)^2 g\cos\theta_b\sin\theta_b}{m_b l^2 + J_b} + \\ 2\left(\dfrac{1}{R} + \dfrac{m_b l\cos\theta_b}{m_b l^2 + J_b}\right)T_w \end{matrix}\right\} \tag{2a}$$

$$\ddot{\theta}_b = \left[J_b + m_b l^2 - \frac{(m_b l\cos\theta_b)^2}{m_b + 2\left(m_{wa} + \frac{J_w}{R^2}\right)}\right]^{-1} \cdot \left\{\begin{matrix} (m_b l\sin\theta_b)\left[g - \dfrac{m_b l\cos\theta_b\left(\dot{\theta}_b\right)^2}{m_b + 2\left(m_{wa} + \frac{J_w}{R^2}\right)}\right] - \\ 2\left[R + \dfrac{m_b l\cos\theta_b}{m_b + 2\left(m_{wa} + \frac{J_w}{R^2}\right)}\right]\dfrac{T_w}{R} \end{matrix}\right\} \tag{2b}$$

Equation (2a) is obtained from consideration of the wheel assemblies while Equation (2b) is from the robot body. Both can be linearized at $\theta_b = \dot{\theta}_b = 0$ (since we assume the robot is upright and stationary initially), using $\sin\theta_b \cong 0$, $\cos\theta_b \cong 1$ and $\dot{\theta}_b \cong 0$:

$$\ddot{x} = -C_1\theta_b + C_2 T_w \tag{3a}$$

$$\ddot{\theta}_b = C_3\theta_b - C_4 T_w \tag{3b}$$

where we introduce the followings coefficients:

$$C_1 = \frac{1}{M}\left(\frac{(m_b l)^2 g}{m_b l^2 + J_b}\right) \tag{4a}$$

$$C_2 = \frac{2}{M}\left(\frac{1}{R} + \frac{m_b l}{m_b l^2 + J_b}\right) \tag{4b}$$

$$C_3 = \frac{m_b g l}{J} \tag{4c}$$

$$C_4 = \frac{2}{JR}\left[R + \frac{m_b l}{m_b + 2\left(m_{wa} + \frac{J_w}{R^2}\right)}\right] \tag{4d}$$

$$M = m_b + 2\left(m_{wa} + \frac{J_w}{R^2}\right) - \frac{(m_b l)^2}{m_b l^2 + J_b} \tag{4e}$$

$$J = J_b + m_b l^2 - \frac{(m_b l)^2}{m_b + 2\left(m_{wa} + \frac{J_w}{R^2}\right)} \tag{4f}$$

Now suppose we use stepper motor to provide locomotion and assume the output torque from the stepper motor to be constant. What we can control is the stepper motor output shaft angular velocity ($\dot{\theta}_w$), which is related to *x* by:

$$R\dot{\theta}_w = \dot{x} \tag{5}$$

We can eliminate the term T_w from Equations (3a) and (3b), converting the resulting equation into frequency domain and replacing $sX(s)$ with $sR\theta_w(s)$ to arrive at the transfer function for the robot platform:

$$\frac{\theta_b(s)}{s\theta_w(s)} = \frac{-sRC_4}{C_2 s^2 + (C_1 C_4 - C_2 C_3)} \tag{6}$$

Equation (6) implies that with a proper adjustment of stepper motor output shaft angular velocity $\dot{\theta}_w$, we can then maintain θ_b close to zero degree, i.e. making the robot upright.

3. COMPUTER MODELLING AND CONTROL

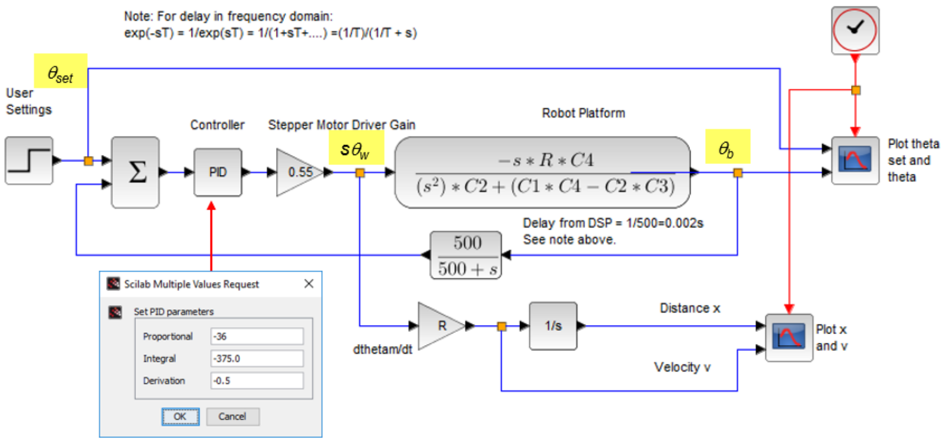
Putting the physical values of our test robot in Figure 5 into (6), we can work out the open-loop poles of the TWSB robot transfer function.

$$C_1 = 4.6645, C_2 = 139.6449, C_3 = 162.0168, C_4 = 2410.3201$$

$$\text{Poles}_{\text{open-loop}} = \sqrt{\frac{-(C_1 C_4 - C_2 C_3)}{C_2}} = \pm 9.028 + j0$$

One of the poles fall on the right half complex plane and the system is intrinsically unstable. We shall now demonstrate the use of PID (proportional, integration and differentiation) controller to implement a stable close-loop system, maintaining the robot platform upright with θ_b close to 0°. Although the commercial software *MATLAB* [16] with its *Simulink* graphical programming environment remain the favourite, here we will be using the free and open source *Scilab* [17] to carry out the computer analysis. *Scilab* and *MATLAB* share many similarities in their syntax and library functions. *Scilab* also supports graphical programming capability with its *XCOS* environment, albeit less polished and capable than its commercial counterpart. The complete stepper motor based TWSB model in *XCOS* is shown in Figure 6. The model in Figure 6 assumes continuous time, whereas in actual implementation the controller will be in discrete time. Thus, any delay due to sample-and-hold, analog-to-digital conversion and digital signal processing (DSP) needs to be taken into account. Based on the performance of our on-board computer, we set the sampling interval to 1 millisecond. The coefficients for the PID controller can be found using analytical or other experimental techniques such as Ziegler-Nichols [18]. Once the PID coefficients are fixed we can determine the closed-loop transfer function and estimate the bandwidth of the closed-loop continuous-time system. From this bandwidth, we will cross check to make sure that the chosen sampling interval is suitable for implementation on a digital computer.
A few items of interest should be mentioned here:

- The output of the PID controller is converted into a periodic square wave using a hardware timer in the on-board computer. This periodic pulse is then used to drive the stepper motor driver IC. The *Stepper Motor Driver Gain* block after the PID controller in Figure 6 accounts for the conversion between PID controller output and the actual angular velocity obtained using Equation (1).



Note: For delay in frequency domain:
exp(-sT) = 1/exp(sT) = 1/(1+sT+....) =(1/T)/(1/T + s)

Delay from DSP = 1/500=0.002s
See note above.

Scilab Multiple Values Request

Set PID parameters

Proportional	-36
Integral	-375.0
Derivation	-0.5

Figure 6. Block diagram of the stepper motor TWSB robot model with PID controller for balancing upright

This value of the gain block depends on the resolution of the hardware timer, the step angle of the stepper motor and the non-linear equation used to map the PID controller output to the timeout period of the hardware timer. It is hardware and algorithm dependent.

- The measured body tilt angle θ_b fed into the PID controller is delayed by two cycles, or two sampling intervals before being processed by the PID controller. This is due to the analog-to-digital conversion delay and DSP routines used to estimate θ_b from the accelerometer and gyroscope outputs of the inertia measurement unit (IMU). Assume τ is the delay period and small, using the transfer function for ideal delay [19] and Taylor's series expansion:

$$H_{delay}(s) = e^{-s\tau} = \frac{1}{1+s\tau+\frac{1}{2}s^2\tau^2+\cdots} \cong \frac{1}{1+s\tau} = \frac{\frac{1}{\tau}}{s+\frac{1}{\tau}} \tag{7}$$

With a sampling interval of 1 millisecond and delay of 2 cycles, $\tau = 0.002$, hence

$$H_{delay}(s) \cong \frac{500}{s+500}$$

- The PID coefficients in Figure 6 (K_p = -36, K_d = -0.5, K_i = -375) are determined using Root Locus design technique [19] with the delay network ignored. In *Scilab* the root locus plot can be generated with the *evans()* command. We can then compute the closed-loop poles of the system, which indicate that this closed-loop system is stable:

$$\text{Poles}_{\text{closed-loop}} = -564.376, \; -4.598 \pm j2.102$$

The dominant poles of the system are $-4.598 \pm j2.1021$. The natural frequency ω_n of this pair of poles is given by [19]:

$$\omega_n = \sqrt{4.598^2 + 2.102^2} \cong 5.06 \text{ or } f_n = \frac{5.06}{2\pi} = 0.805 \text{ Hz}$$

The system bandwidth is approximately proportional to the natural frequency of the dominant pole [19], thus, the system bandwidth is on the order of 1 Hz. In order to discretise the PID controller to a digital controller using zero-order hold (ZOH), a sampling rate at least 30 times larger than the closed-loop system bandwidth is needed so that the system response does not deviate too much from the continuous time analysis [20]. Hence a minimum sampling rate for our discrete PID controller should be at least 30 Hz. In our implementation we are using a sampling rate of 1 kHz so deviation in performance due to discretisation should be small.

3.1 Balancing Experiment 1 (Stable Operation)

We will run two numerical experiments in *Scilab XCOS* to illustrate the usage of the model. First, we force the robot to maintain θ_b = 2° (0.035 radian) with K_p = -36, K_d = -0.5, K_i = -375. The robot platform is initially maintained at body tilt angle θ_b = 0°, with the user setting θ_{set} also set to 0°. At time t = 3.0 seconds, θ_{set} is abruptly changed to 2°. The simulation is run up to t = 15.0 seconds. As seen in Figure 7, the robot responded by moving forward with a velocity that increases linearly. This keeps the body tilt angle θ_b at a constant 2° thereafter. An important observation in Figure 7 is the velocity magnitude will keep on increasing if we maintain θ_{set} at non-zero value. Ultimately the robot will tip over when it can no longer move any faster. Conversely if θ_{set} is set to negative; the robot will move backward to maintain a negative body tilt angle.

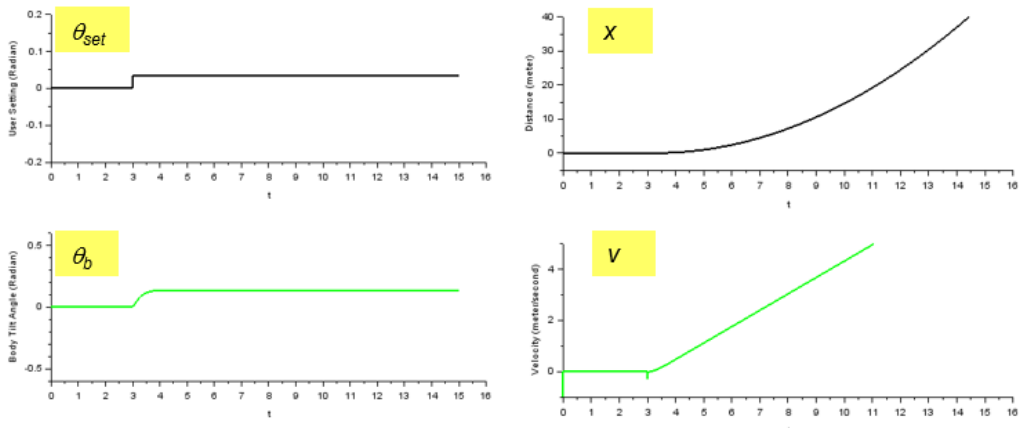


Figure 7. Simulated results for K_p = -36, K_d = -0.5, K_i = -375, stable operation, robot is set to maintain a tilt angle of θ_b = 2°

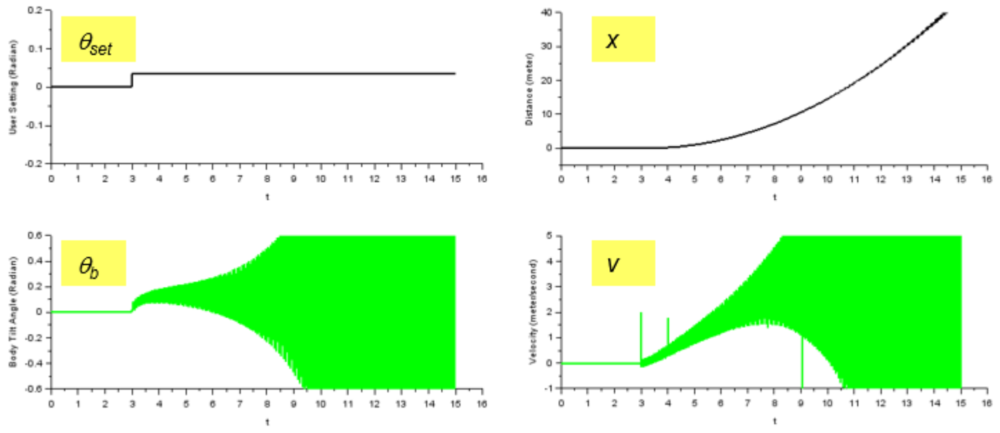


Figure 8. Simulated results for K_p = -36, K_d = 3.4, K_i = -375, unstable operation, robot is set to maintain a tilt angle of θ_b = 2°

3.2 Balancing Experiment 2 (Unstable Operation)

In this experiment we again force the robot to maintain θ_b = 2° (0.035 radian) at time t = 3.0 seconds, however, with different PID coefficients K_p = -36, K_d = 3.4, K_i = -375. The calculated closed-loop poles of the system are:

$$\text{Poles}_{\text{closed-loop}} = -2.761,\ 1.52 \pm j72.262$$

which represent unstable condition. Figure 8 shows that the system state blows up.

3.3 Linear Motion and Steering – Higher Order Control Loop

Based on the observation in Section 3.1, to make the robot moves forward with constant velocity, we need to make the robot lean forward at a small positive θ_b. Once the robot picks up sufficient speed, we can reduce θ_b to zero or even negative to slow down the robot. This suggests another control loop running "on top" of the basic balancing control loop, e.g. a higher order control loop. We will call this the *Velocity Control Loop* (VCL). The input to the VCL is the wheel rotational velocity or linear velocity, while the output will be the θ_{set}. A PI control structure can be employed for the VCL. The VCL is also a discrete time controller, and has a corresponding sampling interval. Suppose we are using half-step (0.9°) approach to drive the stepper motors, and we wish to make the wheel turn at an average rotational speed of 0.25 rotation/second. This means the robot will maintain almost upright position while moving forward at average wheel speed of 0.25 rotation/second (If the wheel radius is 3.2 cm, this translates to around 5 cm/second linear speed). The stepper motor thus needs to turn an average of 100 steps per second (0.25 rotation or 90°, which contain 100 x 0.9° steps), or 10 milliseconds between each step. In this case we can set the sampling interval for the VCL to be between 1 millisecond (the sampling interval for the basic balancing control loop) to 10 milliseconds.

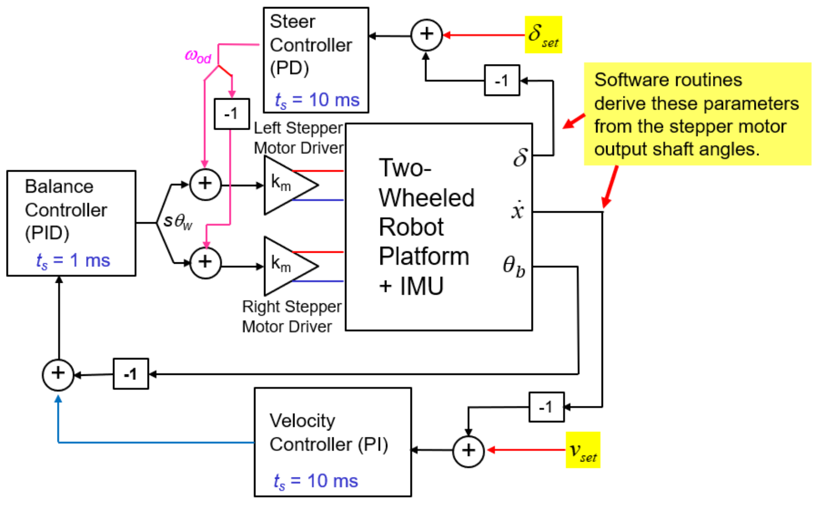


Figure 9. Block diagram of the system including higher order control blocks

The upper limit is due to the fact that the software routine in the on-board computer needs to update the current speed of the wheel after every step, hence ideally the VCL also needs to be executed after a new wheel speed is obtained. In our robot we set the VCL control interval to 10 milliseconds.

Finally, to turn the robot, a small differential velocity can be added to the input of each stepper motor drivers, making the robot turn to the left or right while balancing upright. The turning, or steering can also be implemented as another higher order loop running on top of the basic balancing control loop. The sampling interval of the steering control is also fixed at 10 milliseconds. We find that a PD control architecture is sufficient for the *Steering Control Loop*. This idea is illustrated in Figure 9, where δ is the yaw angle or heading of the robot [1], it can be calculated if we keep track of the distance traveled by both left and right wheels in the software.

The Balance, Velocity and Steering Control Loops run concurrently in the on-board computer software. Thus, one can make the robot move and turn at the same time. For example, if we set v_{set} = 10 cm/second and δ = 0°, the robot will attempt to move forward at 10 cm/second. If we v_{set} = -10 cm/second and δ = 45°, the robot will reverse at 10 cm/second while trying to turn 45 degrees to the left (Negative δ will make the robot turns right). Finally, if v_{set} = 0 and δ = 0°, the robot will just balance upright and remain still. Any attempt to push the robot or turn it forcefully, the control routines will resist such attempt and try to move the robot back to the initial position.

4. EXPERIMENTAL RESULTS

We implemented the Balance, Velocity and Steering Control Loops described in Figure 6 and Figure 9 in the firmware of our TWSB robot. Due to the characteristics of the IMU and stepper motors, and the need to reduce the computational loading on the on-board controller, θ_b is represented as floating point while $\dot{x}$ and δ are stored as signed integers. The Balance Control Loop algorithm is executed in floating points while Velocity and Steer Control Loops are executed in integers format to reduce the computational load (mathematical operations involving integers run faster than those involving floating points). The loss of resolution in using integers to represent the heading and velocity information compromises the effectiveness of the higher order control loops somewhat. However as will be seen in the experimental results and supplementary video the robot still functions satisfactory. The final coefficients for all the control loops are shown in Table 1 (some tuning is done on the test robot to optimise performance).

Our robot contains a wireless transceiver, which we transmit the robot dynamical parameters at 18 samples/second to a computer where they are plotted. Figures 10 and 11 show the plot of the linear motion test, where we instruct the robot to move forward, stop and then reverse. The curves in Figures 10 and 11 contrast the actual linear velocity versus the required velocity. Note the figures should be interpreted from right to left, i.e. event that occurs earlier will be further out in time. In the second test, we instruct the robot to turn left 45°, then turn right 45° twice. This is shown in Figure 12. Apart from the data presented here, we also prepared a short video accompanying this paper, showing the robot in action and further construction details:
https://www.youtube.com/watch?v=TI-Y9vrfFL0
The video provides demonstration of the robot standing still, regaining stability when pushed, moving at constant velocity and turning at specific angle.

One of the notable characteristics while the robot is moving forward or backward is the non-minimum phase behavior [19]. For example, in Figure 10, when the robot is set to move forward, it actually reverses for a short period before moving forward. This same behavior is observed when it is instructed to move backwards. Another observation is the small vibration when the robot is standing still which is apparent in Figures 10 and 11. This is due to the discrete nature of the stepper motor and this effect is more apparent when the motors are turning at low revolution speed.

Table 1. Actual coefficients used in test robot

Balance Controller		Velocity Controller		Steer Controller	
K_p	-30.0	K_p	4	K_p	1
K_d	-0.375	K_d	0	K_d	5
K_i	-315.0	K_i	1	K_i	0

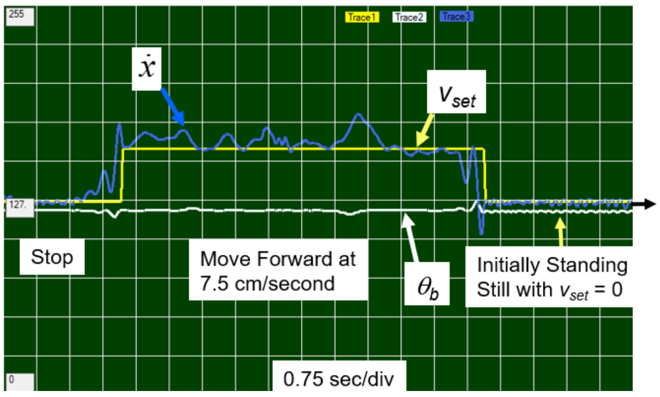


Figure 10. Test 1, robot moves forward at constant velocity with δ_{set} = 0°

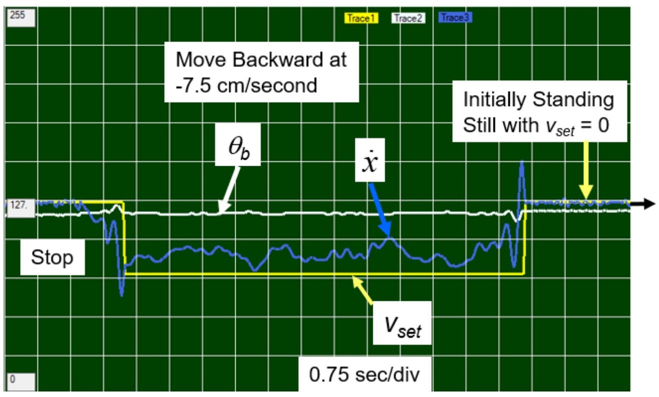


Figure 11. Test 2, robot moves backward at constant velocity with δ_{set} = 0°

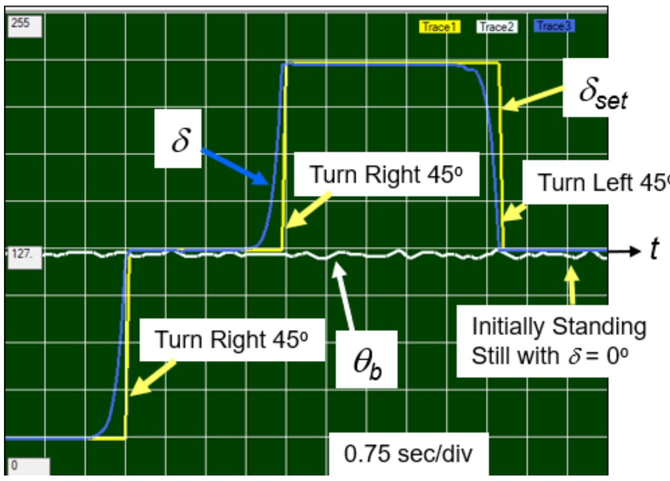


Figure 12. Test 3, turning the robot on-the-spot, v_{set} = 0.

5. CONCLUSION

In this paper the author has described the linear mathematical model and control of a two-wheeled self-balancing robot using stepper motors. Here a two-tier feedback control approach is used, with the lower tier controller keeping the robot balance upright, and the higher tier controller modulates the lower tier controller to achieve higher order movements such as turning and moving at constant linear velocity. An experimental robot is built to substantiate the idea presented. It is hope that this work will provide the basis for future investigations such as extending the control using state-space approach to eliminate multiple feedback control loops, experimentation with variable step angle θ_{step} drive on the stepper motors and using micro-stepping techniques for smoother operation (i.e. lower vibration while standing still), non-linear control etc.

REFERENCES

[1] F. Grasser, A. D'Arrigo, S. Colombi and A. C. Rufer, JOE: A mobile, inverted pendulum, *IEEE Transactions on Industrial Electronics*, 49(1), 107-113, 2002.

[2] R. C. Ooi, *Balancing a two-wheeled autonomous robot*, Final Year Thesis, University of Western Australia, 2003.

[3] D. P. Anderson, nBot balancing robot, http://www.geology.smu.edu/dpa-www/robo/nbot,_ 2013 (accessed 06.03.2019)

[4] R. Chan, K. A. Stol and R. C. Halkyard, Review of modelling and control of two-wheeled robots, *Annual Reviews in Control*, 37, 89-103, 2013.

[5] H. A. Kadir, *Modelling and control of a balancing robot using digital state space approach*, Master Thesis, Universiti Teknologi Malaysia, 2005.
[6] R. L. Glinski, Eddie-Plus self-balancing robot, http://makezine.com/projects/self-balancing-eddie-robot,_2017 (accessed 06.03.2019)
[7] Wowwee Group, MiP robot toy, product description, http://www.wowwee.com/mip,_2014 (accessed 22.05.2019)
[8] JJRobot, The B-Robot Evo self-balancing robot, project webpage, http://www.jjrobots.com/projects-2/b-robot,_2016 (accessed 06.03.2019)
[9] J. Brokking, Your own Arduino balancing robot, personal project webpage, http://www.brokking.net/yabr_main.html,_ 2017 (accessed 30.10.2017)
[10] Pololu Robotics, Balboa robot and accessories, product description, https://www.pololu.com/product/3575,_ 2017 (accessed 30.10.2017)
[11] Microchip Technology, Digital signal controller (DSC) with 70 MIPS core, product description, https://www.microchip.com/wwwproducts/en/dsPIC33EP256MU806,_ 2012 (accessed 06.03.19)
[12] Oriental Motors, *Stepper motor – an overview*, application note, http://orientalmotor.com/stepper-motors/technology/stepper-motor-overview.html (accessed 06.03.2019).
[13] W. Yeadon and A. Yeadon, *Handbook of Small Electric Motors*. New York, NY: McGraw-Hill, 2001.
[14] Texas Instruments, 2.5A Bipolar stepper motor driver with on-chip 1/32 micro-stepping indexer, product description, http://www.ti.com/product/DRV8825,_ 2014 (accessed 06.03.2019)
[15] Allegro MicroSystems, DMOS micro stepping driver with overcurrent protection, product description, http://www.allegromicro.com/en/Products/Motor-Driver-And-Interface-ICs (accessed 06.03.2019)
[16] Homepage of Mathworks, www.mathworks.com.
[17] Homepage of Scilab, www.scilab.org.
[18] K. Ogata, *Modern Control Engineering*. Upper Saddle River, NJ: Prentice-Hall, 2002.
[19] N. Nise, *Control Systems Engineering* (6th edition). Hoboken, NJ: John Wiley & Sons, 2011.
[20] G. F. Franklin, J. D. Powell and M. Workman, *Digital Control of Dynamic Systems*. Boston, MA: Addison-Wesley, 1997.