

Studies of a Two-Wheeled Self-Balancing Mobile Robot Based on a Simulation Viewpoint

Shariman Abdullah, Fairul Azni Jafar, Nur Aidawaty Rafan

Fakulti Teknologi dan Kejuruteraan Industri dan Pembuatan (FTKIP), Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, 76100 Durian Tunggal, Melaka, Malaysia

DOI: <https://doi.org/10.47772/IJRISS.2026.100800706>

Received: 27 August 2026; Accepted: 01 September 2026; Published: 16 September 2026

ABSTRACT

A two-wheeled mobile robot is not able to keep itself upright without the aid of an active controller, because its structure works in much the same way as an inverted pendulum mounted on a wheel base. This paper presents the design of a Proportional-Integral-Derivative (PID) controller for a two-wheeled self-balancing mobile robot, which was designed and tested through co-simulation instead of deriving a full mathematical dynamic model. A three-dimensional model of the robot, made up of a chassis, coaxial wheels and drive motors, and an elevated battery load that produces the moving centroid needed for the robot to behave like an inverted pendulum, was built in Solid Works and imported into MSC Adams as a virtual prototype. The Adams model was then exported as an adams_sub block into Matlab/Simulink, where a discrete PID controller was designed and manually tuned to control the tilt angle of the robot by adjusting the speed of both wheels. Co-simulation was carried out for three cases: self-balancing while at rest, moving forward at a 1-degree tilt angle on a flat platform, and moving across a 10-degree bump. The results show that a proportional-only controller was already enough to balance the robot at rest, while adding the integral and derivative terms during forward motion reduced the steady-state error to a negligible value and kept the overshoot to about 10.8%. The tuned controller ($K_p = 40$, $K_i = 450$, $K_d = 0.11$) also managed to keep the robot from falling while crossing the bump, although the disturbance caused by the bump was not fully rejected. A weakness noticed during forward motion, where the robot drifted sideways because of an uncontrolled yaw angle, was fixed by adding a second PID loop that drives the two wheels at different speeds, although this introduced some cross-coupled noise between the tilt and yaw loops. These results show that co-simulation between Adams and Matlab is an effective alternative to deriving a full mathematical model, reducing the need for a physical prototype and supporting efficient preliminary validation of controllers for two-wheeled self-balancing robots.

Keywords: two-wheeled self-balancing robot, PID controller, co-simulation, Adams-Matlab, inverted pendulum, mobile robot

INTRODUCTION

A two-wheeled mobile robot is the combination of a wheeled mobile robot and an inverted pendulum system. Because the centroid of an inverted pendulum sits above its pivot, the system is naturally unstable, and the pendulum is not self-actuated — it is actuated by the movement of the robot itself. For example, when the robot moves forward, the pendulum body leans backward, and the robot has to respond quickly to avoid toppling over. To regain its stability, the robot is fitted with sensors such as gyroscopes and accelerometers that detect how far the body has inclined from the vertical axis. This inclination is sent back to a controller, which sends a torque command to each wheel motor to stop the system from falling. As a result, a two-wheeled mobile robot shows a back-and-forth movement to overcome its unstable nature (An & Li, 2013).

Two-wheeled self-balancing robots have become increasingly popular among researchers, not only because of their nonlinear behaviour but also because of some advantages they have over other mobile robot designs. Compared with a statically stable robot such as a four-wheeled platform, a two-wheeled robot is harder to control, but it is still much simpler to control than a legged robot (Chan et al., 2013). Its small footprint also

allows it to be fitted with bigger wheels, giving it the potential to travel across rougher terrain than a normal wheeled robot of similar size.

In the conventional way of designing such a robot, three-dimensional modelling is carried out first, usually in CAD software such as SolidWorks, followed by controller design. The controller is derived from a mathematical model that is built from the free-body diagram of the 3D model, and this model usually has to be simplified with a number of assumptions so that the controller design stays manageable. The controller can then be tested in simulation software such as Matlab, but success in this simulation only shows that the controller can stabilize the robot under the assumptions used to build the mathematical model, not that it will work correctly on the real robot. Co-simulation, where a multibody dynamics package such as MSC Adams exchanges data in real time with a control-design package such as Matlab, allows the controller to be tested against a virtual prototype that already captures most of the geometry, mass distribution and contact behaviour of the real robot. This removes the need to derive and simplify a full mathematical model, reduces the design time, and avoids the cost of building a physical prototype just to validate the controller (Ángel et al., 2012; Brezina et al., 2011).

In this study, co-simulation between SolidWorks/Adams and Matlab is used to design, tune and validate a PID controller for a two-wheeled self-balancing mobile robot with a moving centroid. The objectives of this study are to: (1) design a two-wheeled self-balancing mobile robot using 3D-modelling software and design a controller that allows it to self-balance while at rest, move forward at a controlled tilt angle, and move across a bump in the simulation environment; (2) simplify the controller design process through co-simulation instead of deriving a complex mathematical model; and (3) analyse how effective the designed controller is, based on the results obtained from co-simulation. This study is limited to the simulation environment: the three-dimensional model is built in SolidWorks without fabricating a physical prototype, only a PID controller is considered, the bump slope is fixed at about 10° , the maximum tilt angle used is 3° , and the controller gains are tuned manually.

The specific contribution of this study is not the PID algorithm itself, which is well established in the literature, but the integration of a full design-and-validation workflow around it. Unlike studies that validate a controller purely in Matlab against a derived analytical model, this study couples a SolidWorks-based virtual prototype with Adams and Matlab/Simulink so that the same controller is designed, tuned, and tested against a single consistent multibody model across three distinct operating conditions, including an explicit disturbance-rejection scenario. This workflow also surfaced a coupling problem, uncontrolled yaw drift during forward motion, that is not visible from a tilt-angle-only analysis and is not commonly reported in single-scenario PID-tuning studies; the study documents both the fix (a second PID loop) and its cost (cross-coupled noise between the two loops), rather than reporting only a successful outcome. The combination of an integrated co-simulation workflow, multi-scenario evaluation, and transparent reporting of a controller limitation and its trade-off is what distinguishes this study from the broader PID-tuning literature.

LITERATURE REVIEW

Mobile Robot Configurations

Mobile robots can be grouped by their locomotion method into legged and wheeled types. Legged robots have high adaptability on rough terrain but are mechanically more complex, while wheeled robots are simpler and faster, although they are generally less able to adapt to uneven ground (Siegwart & Nourbakhsh, 2004). Among wheeled robots, three- and four-wheeled designs are statically stable, whereas two-wheeled robots depend on active control to stay balanced. Even with this extra control burden, a two-wheeled robot has a smaller footprint, can carry bigger wheels for the same chassis size, and is mechanically simpler than a legged robot. These reasons explain why this configuration continues to attract research interest (Chan et al., 2013).

Controllers for Two-Wheeled Self-Balancing Robots

A number of control strategies have been applied to two-wheeled self-balancing robots. Table 1 summarizes the main approaches reported in the literature, together with the co-simulation-based design approach used in

the present study.

Approach	Principle	Reported Strength	Reported Limitation	Reference
PID	Proportional, integral and derivative correction of the error signal	Simple structure; no plant model required; easy to tune manually	Performance degrades under strong disturbance or nonlinearity	Nasir et al. (2011); Moshayedi et al. (2022)
LQR	Optimal state feedback minimizing a quadratic cost function	Good balance between response speed and control effort	Requires an accurate linearized state-space model	Chan et al. (2013)
ADRC	Estimates and cancels the lumped disturbance in real time	Robust to model uncertainty and external disturbance	More complex to tune than PID	Wang et al. (2016); Ruan et al. (2014)
Backstepping	Lyapunov-based recursive nonlinear design	Guarantees stability for the nonlinear model	Depends on accurate nonlinear model derivation	Nomura et al. (2009)
Sliding Mode Control	Switches the control law to force the system onto a sliding surface	Strong robustness to disturbance and parameter variation	Chattering in the control signal	Meng et al. (2012)
Fuzzy Logic	Rule-based control using linguistic variables	Handles nonlinearity without an exact plant model	Rule base tuning is largely heuristic	Wu & Zhang (2011)
Co-Simulation-Based Design	Couples a multibody dynamics model (Adams, Simscape Multibody) with Matlab/Simulink instead of deriving an analytical model	Removes need to derive/simplify a mathematical model; validated against a realistic virtual prototype	Depends on access to and correct set-up of multibody simulation software	Tang et al. (2021); Emara et al. (2022); Khadr (2024); Yu et al. (2025)

Table 1: Comparison of Controller Design and Validation Approaches for Two-Wheeled Self-Balancing Robots

Among the controller types, PID is still the most commonly implemented for two-wheeled self-balancing robots because it does not need a full plant model, and it continues to be paired with co-simulation in recent work, for example to tune a PID controller automatically through an optimisation algorithm rather than by hand (Moshayedi et al., 2022). As the last row of Table 1 shows, co-simulation is best understood as a design and validation approach that can sit alongside any of these controller types, rather than a controller type on its own. The present study combines it with a PID controller, for the same reason PID is common in this field: it can be designed and tuned without deriving a full mathematical model of the robot.

Co-simulation of Adams and Matlab

Most simulations of two-wheeled self-balancing robots reported in the literature rely only on Matlab, using the dynamic and kinematic equations derived for the robot. Because this kind of simulation does not capture real-world effects such as inertia distribution, friction and contact, a number of studies have combined Adams, a multibody dynamics package, with Matlab/Simulink to build a more realistic simulation environment for designing and validating controllers, for example in vehicle suspension and vibration-isolation systems (Ángel et al., 2012; Brezina et al., 2011; Zhu et al., 2010) and in the control of two-wheeled vehicles (Qian et al., 2011).

More recently, the same co-simulation approach has continued to be used for a wider range of mobile and legged robots. Tang et al. (2021) used Adams and Matlab co-simulation to design and tune a controller for a two-wheel differential spherical robot, which is close to the robot studied here. A similar combination of

Adams or Simscape Multibody with Simulink has also been used for trajectory and gait planning of a quadruped robot (Morais et al., 2021), a gecko-like wall-climbing robot (Chen, 2022), a hydraulic hexapod robot (Liu et al., 2022), and a hexapod robot walking on complex terrain, where the co-simulation result was also checked against a physical prototype (Yu et al., 2025). Outside legged and spherical robots, Emara et al. (2022) built a digital-twin co-simulation for the closed-loop control of a three-wheeled omnidirectional mobile robot, Khadr (2024) used a similar multibody approach to model a three-wheeled mobile robot, and Boschetti and Sinico (2024) used Simscape Multibody to build digital twins of robot manipulators. Moshayedi et al. (2022) went a step further and combined co-simulation with an optimisation algorithm to tune a PID controller for an automated guided vehicle automatically, instead of by hand, which points to a possible improvement over the manual tuning used in the present study. On a broader scale, Ribas-Xirgo (2026) shows that the same simulator-based approach is now being extended to build digital twins of whole robotics laboratories, not just a single robot. Taken together, these recent studies show that co-simulation is still a practical and increasingly common way to design and validate a controller without deriving a full mathematical model, which supports the approach used in this study.

METHODOLOGY

Robot Design and Three-Dimensional Modelling

The two-wheeled self-balancing mobile robot was modelled in SolidWorks with a three-tier chassis. The bottom tier holds the two coaxial wheels and drive motors, the middle tier carries the battery pack, which acts as the load, and the top tier is a cover platform. The batteries were placed on an elevated tier above the wheel axis on purpose, because they are heavy enough to produce the moving centroid needed for the robot to behave like an inverted pendulum. The finished SolidWorks model is shown in Figure 1.

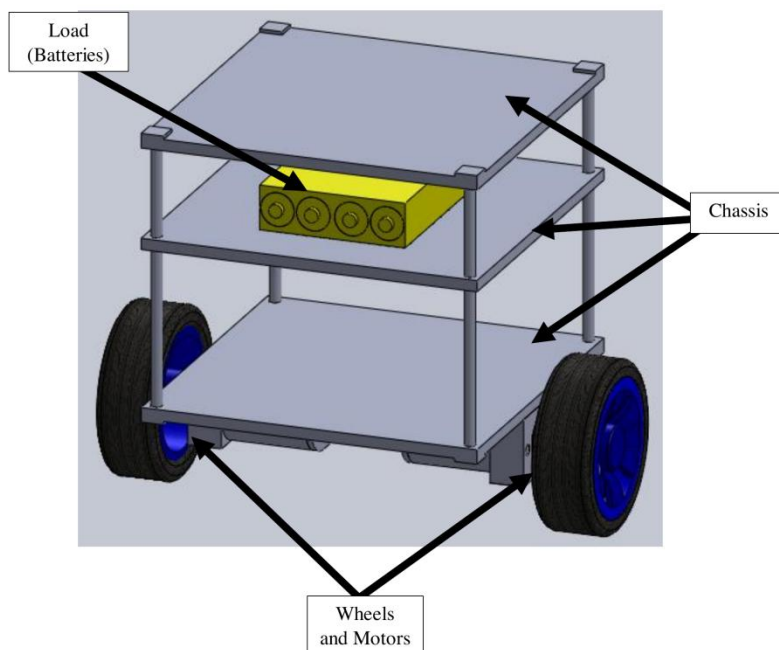


Figure 1: SolidWorks Model of the Two-Wheeled Self-Balancing Mobile Robot

Transfer to Adams and Co-Simulation Setup

The SolidWorks model was imported into MSC Adams, where fixed joints were assigned between the chassis plates and revolute joints between the wheels and the chassis, and a contact was defined between each wheel and the ground to capture friction and the normal force between them. The Adams Controls module was then set up to define the input variables (left- and right-wheel speed) received from Matlab and the output variable (tilt angle of the robot body) sent back to Matlab, and the configured model was exported as an “adams_sub” block that could be used directly inside a Matlab/Simulink block diagram. This removed the need to derive a

separate mathematical transfer function for the robot dynamics. Two platforms were built in Adams: a flat platform used for Case 1 and Case 2, and a platform with a series of bumps of about 10° slope, used for Case 3 to act as a disturbance to the controller. Figure 2 shows the co-simulation of the robot moving across the bump platform.

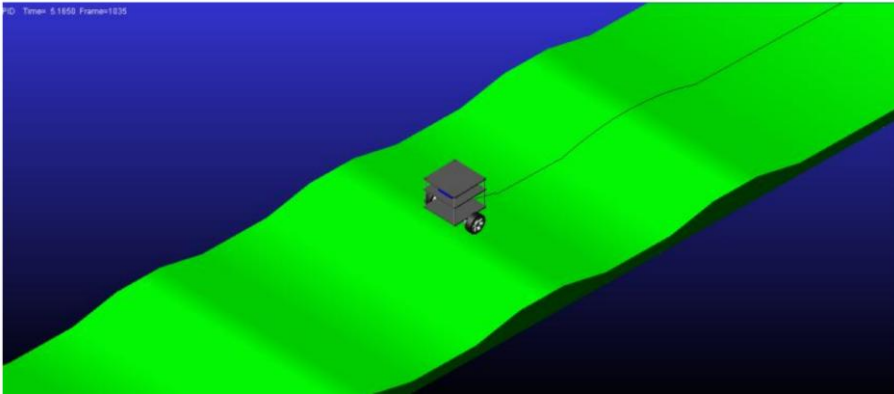


Figure 2: Co-Simulation of the Two-Wheeled Mobile Robot on a Platform with Bumps in Adams

PID Controller Design

A discrete PID controller was designed in Simulink to control the tilt angle of the robot by sending the same speed command to both wheels, so that the robot only moves forward or backward. The controller takes a step input that represents the desired tilt angle (step time of 0.5 s; final value of 0° for Case 1, 1° for Case 2 and 3° for Case 3) and compares it with the tilt angle fed back from the `adams_sub` block through a unit delay. This unit delay was added because, without it, the feedback was sent too quickly for the discrete solver to register the change correctly. A fixed step size of 0.005 s was used, which is small enough for the discrete controller to behave in the same way as a continuous one. A discrete controller was chosen because it matches how the controller would eventually be built using digital electronics on the physical robot. Figure 3 shows the block diagram of the designed PID controller.

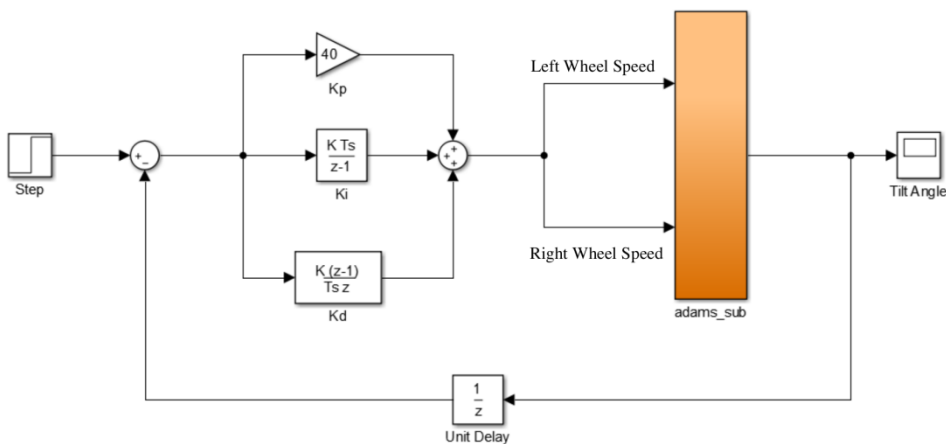


Figure 3: Block Diagram of the Designed Discrete PID Controller

The controller gains were obtained through manual, sequential tuning rather than an automated method such as Ziegler–Nichols or an optimisation algorithm; this choice keeps the tuning procedure transparent and easy to reproduce by hand, at the cost of the gains not being formally optimal. Tuning was carried out at the same fixed sampling time used for co-simulation, 0.005 s, so that the tuned gains did not need to be re-validated at a different step size. No explicit actuator saturation limit was imposed in the controller model; the wheel-speed command was left unconstrained, and the applied gains were instead bounded implicitly by the stopping criteria described below. The procedure followed three stages, each with its own explicit performance criterion. First, the proportional gain, K_p , was increased from zero until the robot could hold the desired 0° tilt angle for the full 10 s duration of Case 1 without falling; this was the stopping criterion for K_p . Second, the

integral gain, K_i , was introduced and increased until the steady-state tilt-angle error in Case 2 was reduced to a negligible level (below 0.01°); this was the stopping criterion for K_i , and was accepted even though it introduced a percent overshoot of about 12.5%. Third, the derivative gain, K_d , was increased until the percent overshoot in Case 2 was brought down toward the target of 10% or below, while re-checking Case 1 and Case 3 after every adjustment to confirm the robot still balanced at rest and did not fall while crossing the bump; this was the stopping criterion for K_d . The final gains obtained through this procedure, and used for all three cases, were $K_p = 40$, $K_i = 450$ and $K_d = 0.11$.

RESULTS

Case 1: Self-Balancing at Rest

With a constant 0° input, the robot had to hold its balance for 10 s. Without a controller, the tilt angle kept increasing until the robot fell, reaching about 45° before settling near 40° , which is the point where the robot touched the ground (Figure 4). With only a proportional controller, the robot was already able to hold 0° for the full 10 s. Adding the integral and derivative terms did not bring any visible improvement over the P-only response, and all four controller settings (P, PI, PD, PID) overlapped on the graph. A steady-state error of about 0.005° could be seen on close inspection but was small enough to be ignored.

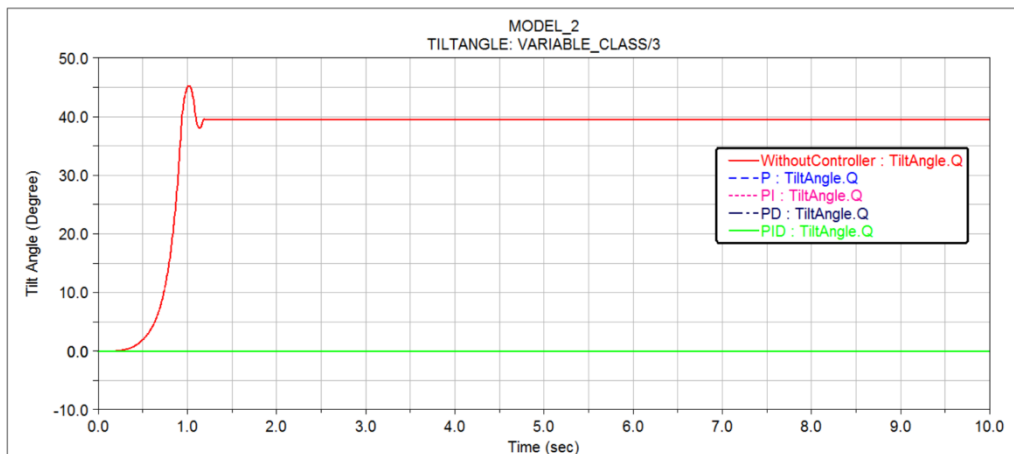


Figure 4: Tilt Angle Response for Case 1 (Self-Balance at Rest), With and Without the Controller

Figure 5 shows a direct visual comparison of the robot's body angle at the same simulation instant ($t \approx 7.93$ s) with and without the controller active, taken from the co-simulation animation. Without the controller, the chassis has tilted to about 40° from vertical, consistent with the tilt angle plotted in Figure 4, and the robot is visibly leaning toward the ground. With the PID controller active, the chassis remains upright at approximately 0° , illustrating directly what the flat line in Figure 4 corresponds to physically rather than only as a numerical trace.

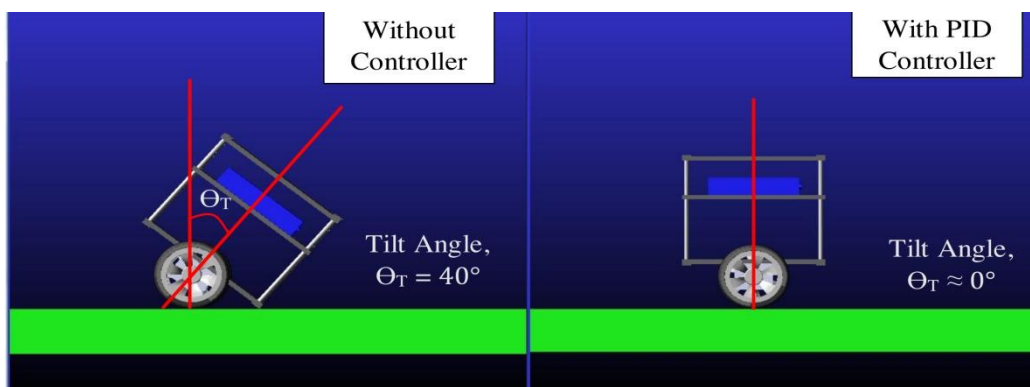


Figure 5: Comparison of the Robot's Tilt Angle With and Without the Controller Active

Case 2: Forward Motion at 1° Tilt Angle

For a 1° step input on a flat platform, a proportional-only controller caused a continuous drift, with the tilt angle rising from the desired 1° to about 1.5° after 10 s. Adding the integral gain fixed this drift and kept the tilt angle at the desired 1°, but it also caused an overshoot of about 12.5%. Adding the derivative gain brought this overshoot down to about 10.8% (Figure 6), without making the rise time (0.1 s) or settling time (0.25 s) any worse than with the PI controller, and the steady-state error stayed negligible at about 0.005°. Since increasing the gains further only made the overshoot worse, the 10.8% result, slightly above the 10% target, was accepted, as it did not stop the robot from balancing even at the higher 3° tilt angle used in Case 3.

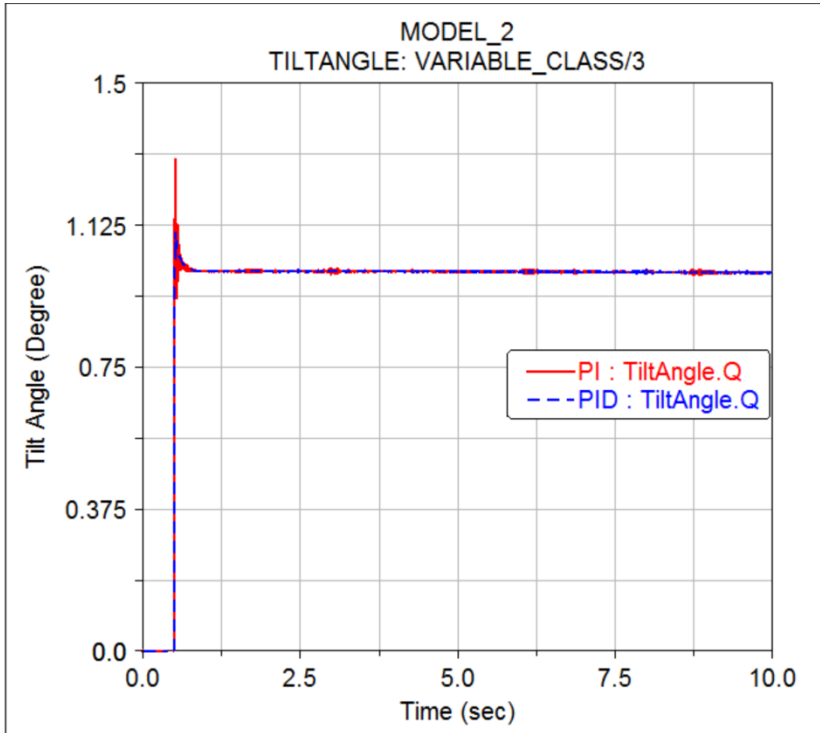


Figure 6: Effect of Adding Derivative Action (PID) Compared with PI Control in Case 2

Case 3: Crossing a Bump

For the robot to cross a 10° bump without stalling, the commanded tilt angle was raised to 3° to give it enough forward speed. Figure 7 shows that the tuned PID controller kept the tilt angle close to 3° and stopped the robot from falling while crossing the bump, although some low-amplitude oscillation can still be seen once the robot reaches the bump, caused by the disturbance from the terrain. This shows that although the PID controller can keep the robot balanced under this disturbance, it does not reject it completely.

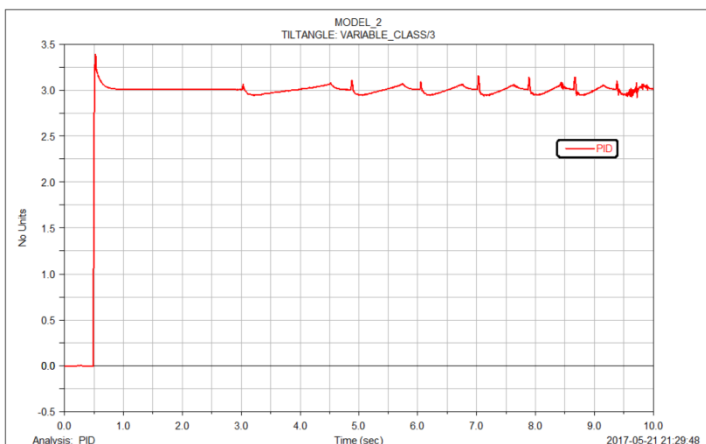


Figure 7: Tilt Angle Response for Case 3 (Crossing a 10° Bump)

Figure 8 shows six snapshots of the co-simulation animation as the robot moves across the bump platform between $t = 0$ s and $t = 9.27$ s. The bump platform consists of a series of raised sections with a slope of about 10° , modelled directly as part of the Adams terrain rather than as a mathematical disturbance signal; the robot's actual centre-of-mass trajectory across this terrain is traced by the curved line visible in panels 2 to 6. The sequence shows the robot climbing onto the first bump (panels 2 and 3), crossing the raised sections while maintaining its upright posture (panels 4 and 5), and reaching a flatter section of the platform by the end of the run (panel 6), which corresponds to the tilt angle staying close to the 3° setpoint throughout Figure 7 despite the disturbance from the terrain.

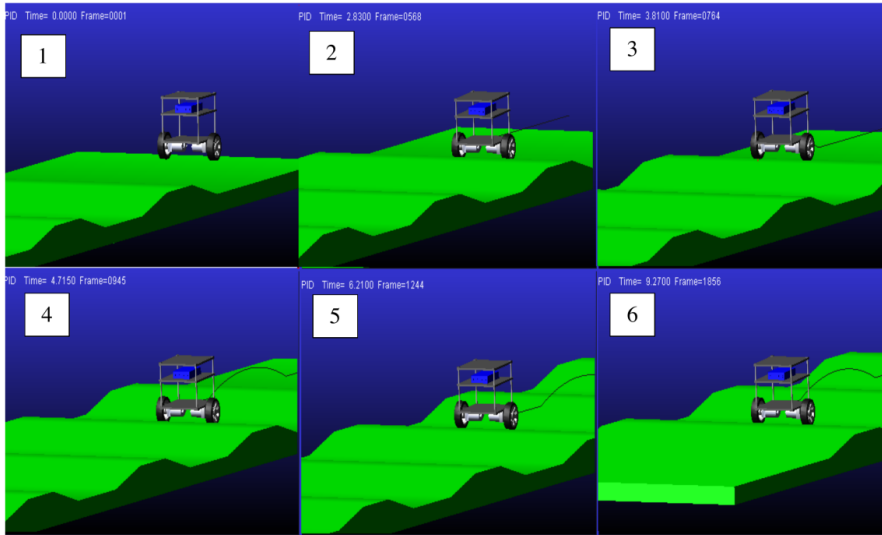


Figure 8: Snapshots of the Robot Crossing the Bump Platform Between 0 and 10 Seconds

Summary of Performance Metrics

Table 2 collects the quantitative performance indicators for all three cases, read from the co-simulation output shown in Figures 4, 6 and 7. Rise time, settling time and percent overshoot are only defined for a step response to a non-zero target; Case 1 uses a constant 0° input and is reported separately as a stability outcome rather than a step-response metric. The bump-crossing disturbance in Case 3 is a repeated, continuous disturbance rather than a single discrete event, so it is reported as a peak deviation and an oscillation band around the 3° setpoint rather than a single settling time. Control effort (e.g., commanded wheel torque or current) was not logged as a separate output in the original co-simulation, which recorded only the tilt-angle (and, for the modified controller, yaw-angle) response; this is a limitation of the underlying dataset rather than of the co-simulation method itself, and is noted here rather than substituted with an estimated figure.

Case	Rise Time	Settling Time	Overshoot	Steady-State Error / Deviation Band	Peak Tilt Angle
Case 1 – No Controller	—	—	Diverges (robot falls)	—	Peak $\approx 45^\circ$; settles $\approx 40^\circ$
Case 1 – PID	0 s	0 s	0%	$\approx 0.005^\circ$	$\approx 0^\circ$ throughout
Case 2 – PI	≈ 0.1 s	≈ 0.25 s	12.50%	$\approx 0.005^\circ$	—
Case 2 – PID	0.1 s	0.25 s	10.80%	$\approx 0.005^\circ$	—
Case 3 – PID (bump)	Varies (continuous bump disturbance)	Varies (continuous bump disturbance)	$\approx 13\%$ (initial step, peak 3.4° vs 3° target)	± 0.15 – 0.2° about 3° during bump traversal	Peak $\approx 3.4^\circ$ (initial); ≈ 3.15 – 3.18° (bump-induced)

Table 2: Summary of Controller Performance Metrics Across the Three Test Cases

DISCUSSION

Weakness of the Tilt-Only Controller

Although the single-loop PID controller met the tilt-angle target in all three cases, watching the animation of the co-simulation showed a weakness that cannot be seen from the tilt-angle graphs alone: while moving forward, the robot drifted sideways away from its intended straight path, and if left uncorrected, it would eventually run off the platform even though it was still holding its commanded tilt angle (Figure 9). This happens because the robot's yaw rotation about its vertical axis is not controlled — the single-loop controller sends the same speed command to both wheels and has no way to correct any yaw disturbance that builds up during the motion.

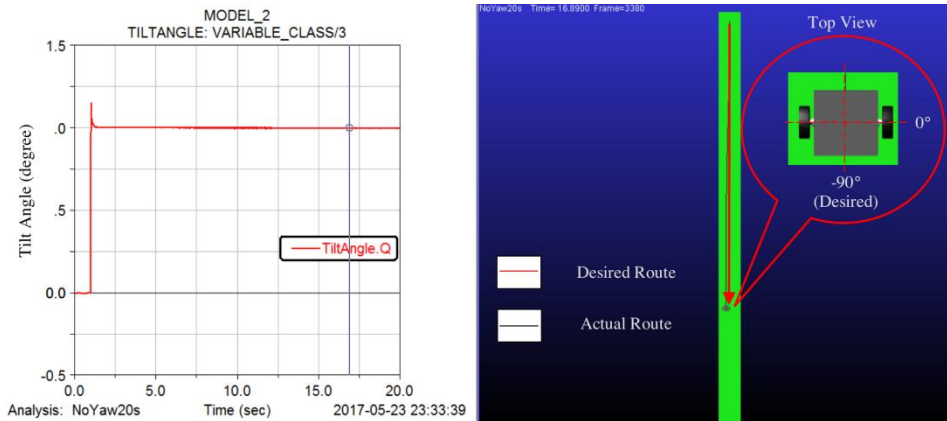


Figure 9: Deviation of the Robot from Its Desired Straight-Line Route Due to Uncontrolled Yaw

Modified Dual-Loop Controller

To fix this, a second discrete PID loop was added to control the yaw angle of the robot, working alongside the existing tilt-angle loop. The yaw loop uses switch logic to create a speed difference between the left and right wheels whenever the robot moves away from its desired heading of -90° , letting it turn back toward the intended route while the tilt loop keeps it upright. With this change, the robot managed to keep its tilt angle close to 1° and its heading close to -90° , and its actual path closely followed the desired straight route (Figure 10). However, because both loops act on the same pair of wheel-speed outputs, they interact with each other: compared with the tilt-only case, the modified response shows extra high-frequency noise in both the tilt-angle and yaw-angle signals once the yaw loop is switched on. This noise did not cause the robot to fall in any of the runs carried out, but it remains a limitation of using two separately tuned PID loops that share the same actuators, and it could possibly be reduced by using a controller that handles tilt and yaw together, such as a multivariable state-feedback or ADRC-based design.

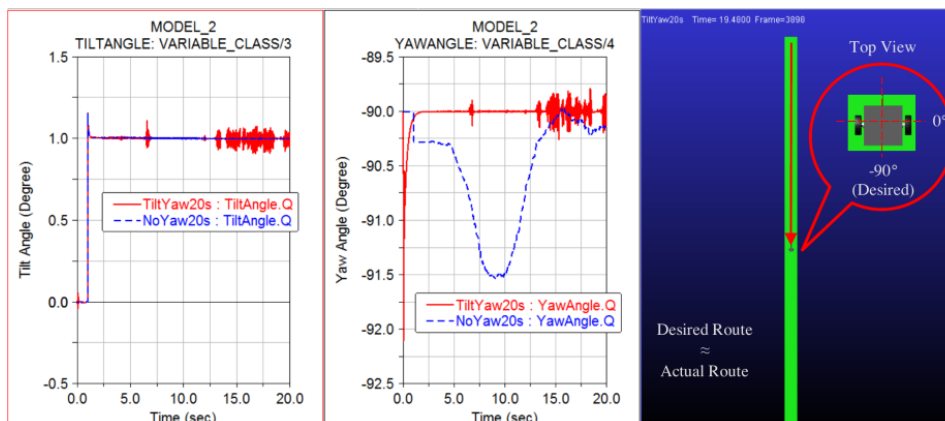


Figure 10: Tilt Angle, Yaw Angle and Route Tracking of the Robot with the Modified Dual-PID Controller

The cross-coupled noise observed once the yaw loop is active can be explained mechanistically: both the tilt loop and the yaw loop write into the same pair of wheel-speed commands, since the tilt loop sets the common (average) speed of the two wheels while the yaw loop sets their speed difference. Because these two loops were tuned independently as single-input single-output PID controllers rather than designed together, each loop's correction is applied without any explicit model of how it disturbs the state the other loop is trying to hold, so a correction from one loop is seen by the other as an unmodelled disturbance. A coordinated design, for example a multivariable state-feedback controller (such as LQR) or an ADRC-based design that treats tilt and yaw as a single coupled system with two outputs, could in principle account for this interaction directly in the control law rather than relying on the two loops to reject each other's disturbance after the fact. A decoupled or cascade structure is a lower-effort alternative: the yaw correction could be rate-limited or low-pass filtered so that it only adjusts the wheel-speed differential slowly relative to the tilt loop's bandwidth, which would reduce the high-frequency interaction between the two loops without requiring a full multivariable redesign. Distinguishing between these two options, coordinated multivariable control versus a decoupled/cascade PID structure, is left as a direction for future work, since it requires redesigning the controller rather than adjusting the gains of the existing one.

Implications for Co-simulation-Based Controller Design

These results support co-simulation as a practical way to design a controller for a two-wheeled self-balancing robot. Because Adams provides a virtual prototype directly to Matlab as an `adams_sub` block, the controller could be designed and improved without first deriving and simplifying a nonlinear mathematical model, which removed a common source of modelling error and shortened the design process. Being able to check both the tilt-angle graph and the real-time animation was also important in this study, since the yaw-drift weakness discussed above was only found through the animation and could not be seen from the tilt-angle graph alone. These findings support the wider argument that co-simulation is a faster alternative to a fully analytical design-then-prototype approach for balancing robots, since it reduces reliance on a physical prototype and supports efficient preliminary controller validation before committing to fabrication.

CONCLUSION

A PID controller for a two-wheeled self-balancing mobile robot was designed and tested using co-simulation between SolidWorks, Adams and Matlab/Simulink, without deriving a full mathematical dynamic model of the robot. The tuned controller ($K_p = 40$, $K_i = 450$, $K_d = 0.11$) balanced the robot while at rest, held a commanded 1° tilt angle during forward motion with about 10.8% overshoot and a negligible steady-state error, and kept the robot from falling while crossing a 10° bump, although it did not fully reject the disturbance caused by the bump. A yaw-drift weakness in the single-loop controller, which caused the robot to leave its intended straight route even though it was holding its tilt angle, was fixed by adding a second PID loop for yaw control, although this brought some cross-coupled noise between the two loops. These results show that co-simulation can simplify and speed up controller design for two-wheeled self-balancing robots, and that it lets a controller's effectiveness be checked both from graphs and from direct animation of the virtual prototype. Future work should look into replacing the PID controller with one that is better able to reject the bump disturbance and to handle the tilt and yaw dynamics together, such as LQR, ADRC or H_∞ control, and should test this approach against a physical prototype of the robot.

ACKNOWLEDGEMENTS

The authors extend their sincere appreciation to Universiti Teknikal Malaysia Melaka (UTeM) and the Fakulti Teknologi dan Kejuruteraan Industri dan Pembuatan (FTKIP) for the valuable support provided, including access to essential resources, which facilitated the successful execution of this study.

REFERENCES

1. An, W., & Li, Y. (2013). Simulation and control of a two-wheeled self-balancing robot. 2013 IEEE International Conference on Robotics and Biomimetics (ROBIO), 456–461.

- https://doi.org/10.1109/ROBIO.2013.6739501
2. Ángel, L., Pérez, M. P., Díaz-Quintero, C., & Mendoza, C. (2012). ADAMS/MATLAB co-simulation: Dynamic systems analysis and control tool. *Applied Mechanics and Materials*, 232, 527–531. <https://doi.org/10.4028/www.scientific.net/AMM.232.527>
3. Boschetti, G., & Sinico, T. (2024). Designing digital twins of robots using Simscape Multibody. *Robotics*, 13(4), 62. <https://doi.org/10.3390/robotics13040062>
4. Brezina, T., Hadas, Z., & Vetiska, J. (2011). Using of co-simulation ADAMS-SIMULINK for development of mechatronic systems. *MECHATRONIKA*, 2011 14th International Symposium, 59–64. <https://doi.org/10.1109/MECHATRON.2011.5961080>
5. Chan, R. P. M., Stol, K. A., & Halkyard, C. R. (2013). Review of modelling and control of two-wheeled robots. *Annual Reviews in Control*, 37(1), 89–103. <https://doi.org/10.1016/j.arcontrol.2013.03.004>
6. Chen, Z. (2022). Joint simulation of the kinematics of a gecko-like quadrupedal wall-climbing robot based on SIMULINK and ADAMS. *Proceedings of SPIE 12261, International Conference on Mechanical Design and Simulation (MDS 2022)*, 122610R. <https://doi.org/10.1117/12.2638737>
7. Emara, M. B., Youssef, A. W., Mashaly, M., Kiefer, J., Shihata, L. A., & Azab, E. (2022). Digital twinning for closed-loop control of a three-wheeled omnidirectional mobile robot. *Procedia CIRP*, 107, 1245–1250. <https://doi.org/10.1016/j.procir.2022.05.139>
8. Khadr, A. (2024). Multibody dynamics modeling and simulation of a three-wheeled mobile robot using a robotic approach. *International Journal of Intelligent Unmanned Systems*, 12(3), 270–289. <https://doi.org/10.1108/IJUS-12-2023-0192>
9. Liu, Z., Jin, B., Dong, J., Zhai, S., & Tang, X. (2022). Design and control of a hydraulic hexapod robot with a two-stage supply pressure hydraulic system. *Machines*, 10(5), 305. <https://doi.org/10.3390/machines10050305>
10. Meng, X., Liu, X., He, Y., Li, J., & Zhang, Y. (2012). The design of VSC with disturbance compensation based on two-wheeled unstable vehicle. 2012 24th Chinese Control and Decision Conference (CCDC), 2211–2215. <https://doi.org/10.1109/CCDC.2012.6243035>
11. Morais, S., Singh, H., & Acharya, A. (2021). Trajectory and gait planning of a quadruped using Adams and Simulink. 2021 4th Biennial International Conference on Nascent Technologies in Engineering (ICNTE), 1–6. <https://doi.org/10.1109/ICNTE51185.2021.9487660>
12. Moshayedi, A. J., Li, J., Sina, N., Chen, X., Liao, L., Gheisari, M., & Xie, X. (2022). Simulation and validation of optimized PID controller in AGV (automated guided vehicles) model using PSO and BAS algorithms. *Computational Intelligence and Neuroscience*, 2022, Article 7799654. <https://doi.org/10.1155/2022/7799654>
13. Nasir, A. N. K., Ahmad, M. A., Ghazali, R., & Pakheri, N. S. (2011). Performance comparison between fuzzy logic controller (FLC) and PID controller for a highly nonlinear two-wheels balancing robot. *Informatics and Computational Intelligence (ICI)*, 2011 First International Conference on, 176–181. <https://doi.org/10.1109/ICI.2011.37>
14. Nomura, T., Kitsuka, Y., Suemitsu, H., & Matsuo, T. (2009). Adaptive backstepping control for a two-wheeled autonomous robot. *Proceedings of ICCAS-SICE 2009*, 4687–4692.
15. Qian, H., Chen, L., Qiao, W., Li, P., Yang, S., & Liu, Q. (2011). Controlling simulation study on two-wheeled self-balancing electrical motorcycle based on ADAMS and MATLAB. *Cross Strait Quad-Regional Radio Science and Wireless Technology Conference (CSQRWC)*, 2011, 1704–1707. <https://doi.org/10.1109/CSQRWC.2011.6037305>
16. Ribas-Xirgo, L. (2026). Simulator-based digital twin of a robotics laboratory. *Machines*, 14(3), 273. <https://doi.org/10.3390/machines14030273>
17. Ruan, X., Wang, X., Zhu, X., Chen, Z., & Sun, R. (2014). Active disturbance rejection control of single wheel robot. *Proceeding of the 11th World Congress on Intelligent Control and Automation (WCICA)*, 4105–4110. <https://doi.org/10.1109/WCICA.2014.7053403>
18. Siegwart, R., & Nourbakhsh, I. R. (2004). *Introduction to autonomous mobile robots*. Cambridge, MA: MIT Press.
19. Tang, Y., Liu, J., Liang, J., Yang, T., Xu, K., He, J., & Gao, B. (2021). Co-simulation of two-wheel differential spherical robot based on ADAMS and MATLAB. 2021 IEEE 11th Annual International

- Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER), 533–538. <https://doi.org/10.1109/CYBER53097.2021.9588231>
20. Wang, X., Chen, S., Chen, T., & Yang, B. (2016). Study on control design of a two-wheeled self-balancing robot based on ADRC. 2016 35th Chinese Control Conference (CCC), 6227–6232. <https://doi.org/10.1109/ChiCC.2016.7554335>
21. Wu, J., & Zhang, W. (2011). Design of fuzzy logic controller for two-wheeled self-balancing robot. Strategic Technology (IFOST), 2011 6th International Forum on, 1266–1270. <https://doi.org/10.1109/IFOST.2011.6021250>
22. Yu, C., Fan, D., & Liu, X. (2025). Collaborative optimization control of gravity center and pose of hexapod robot in complex terrains. *Machines*, 13(9), 871. <https://doi.org/10.3390/machines13090871>
23. Zhu, D., Qin, J., Zhang, Y., Zhang, H., & Xia, M. (2010). Research on co-simulation using ADAMS and MATLAB for active vibration isolation system. Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on, 1126–1129. <https://doi.org/10.1109/ICICTA.2010.711>