Application of PID on Self Balancing Robot:

Extended Essay
Mathematics
3,297 words

Table of Contents:

Application of PID on Self Balancing Robot

Introduction:

Through my participation in the FIRST robotics competition for several seasons, I always came across the problem of the robot not being able to move in a straight trajectory due to asymmetry in design or motor voltage reception. This problem was solved using the PID algorithm, which is what I want to explore in this extended essay by building and employing the algorithm on a self-balancing robot. The PID algorithm stands for proportional integral derivative. A PID controller is used in industrial control applications to regulate process variables that could range from the speed of the vehicle to the temperature of room. In my example, the PID algorithm will be regulating the angle of inclination for the robot, to ensure that it will remain perpendicular to the ground. Since the robot will be standing only on two motorized wheels, the design relied on creating structural symmetry and balance of mass on both sides of the robot to ease the process of stabilizing (Refer to figure 1). The battery was centered in the center of the top section, while the electronics (lightweight) were mounted on the first level.

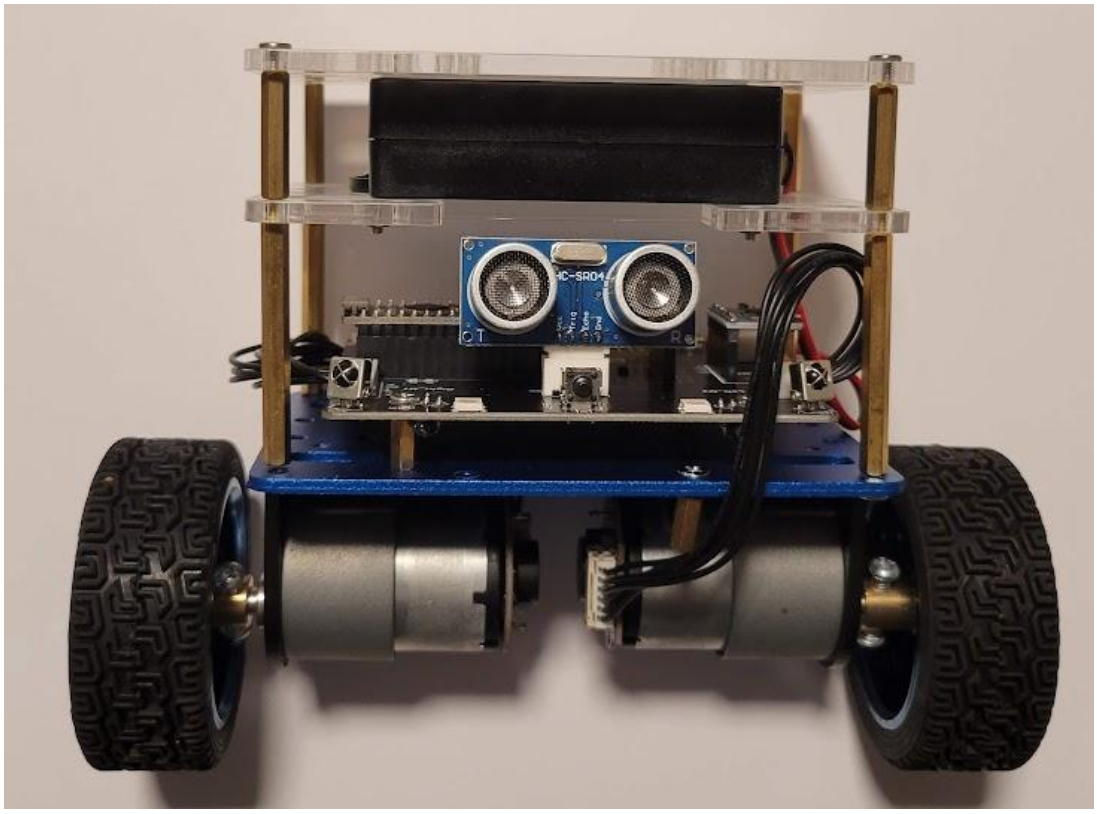


Figure 1

The research question for this paper is to find how to calculate and apply the PID controller algorithm on a two-wheel self balancing robot that constantly adjusts itself to reach the desired target (being balanced).

Function of PID Algorithm:

As stated before PID is a closed loop algorithm that stands for Proportional, Integral, and Derivative. Each of these components is a term used to calculate the appropriate amount of power to apply to the system in the next timestep. PID is a highly flexible type of Closed Loop control to make a system reach a desired target value in a smooth and immediate manner. PID is used to set a continuous output value that takes time to change. The application of the PID algorithm today is beyond essential for regulating activities in areas such as manufacturing, autonomous driving, and other places that have process variables. The algorithm function is to calculate and return the desired set output of the next timestep. In order to understand how the PID controller works, it is important to know basic open and closed loop theory. Open loop systems use an algorithm to determine how much power to feed into the motor to get to the desired target (refer to figure 2). This system fails to account for unexpected disturbances.

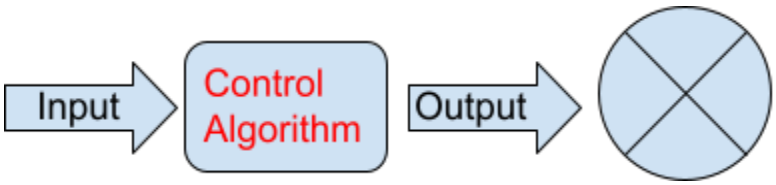


Figure 2

A closed loop system is capable of taking more complex disturbances and the output is fed back as input in order to reduce errors and increase stability (Refer to Figure 3).

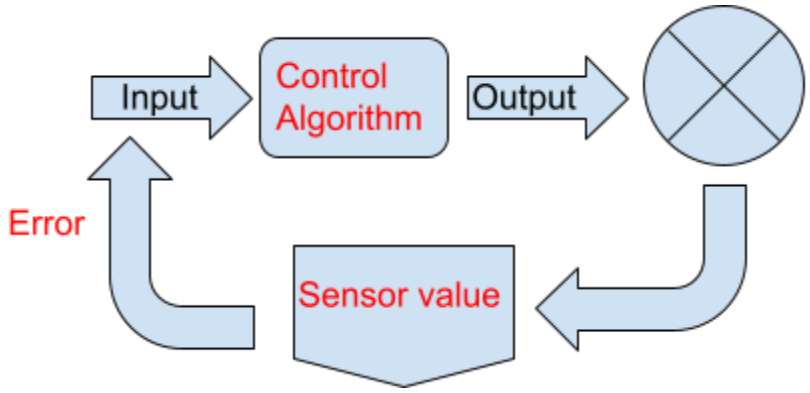


Figure 3

This then results in a reference signal (target position) that would be compared with the current measured value in order to provide the error term which the algorithm would use to act accordingly (reduce it). The way feedback is provided to the control system is through sensors measuring the process variable. This variable is the "parameter" that needs to be "controlled" (NI, 2022). For example, in a car a gas pedal is the process variable. Combining the three terms of the algorithm, PID serves the purpose of reducing the error between the process variable and reference signal to control the system. The algorithm uses past information (from integral), and present information (from proportional gain) to deduce the future action (from derivative). The proportional response is the examination of the present value of the process variable evaluated at a given interval of time. If the error is high there is a high correction value and vice versa. The integral response sums up past errors no matter how big or small, because small errors over time can change the output. Errors will be taken constantly in the control system and eventually the summation of these errors will lead to adjustment of output so that the error can go down to zero. The derivative response predicts the error rate by examining the rate of change over a given interval of time. The derivative response is "proportional to the rate of the change" (Teow, 2018) of the process variable.

PID and Dynamic Stability:

The final goal of the robot is to reach dynamic stability. Dynamic stability is the "ability of a powered system to return to its steady state after facing significant disturbances." (Nenchev , 2016) In order to know the robot's current position, the MPU6050 (9 axis gyroscope) will be used. The information from the sensor will be

used by the PID algorithm which will adjust the motors to reach the balanced effect (see figure 4). A disturbance to this system could be slippery surface, human push of the robot, physical obstacles, etc.

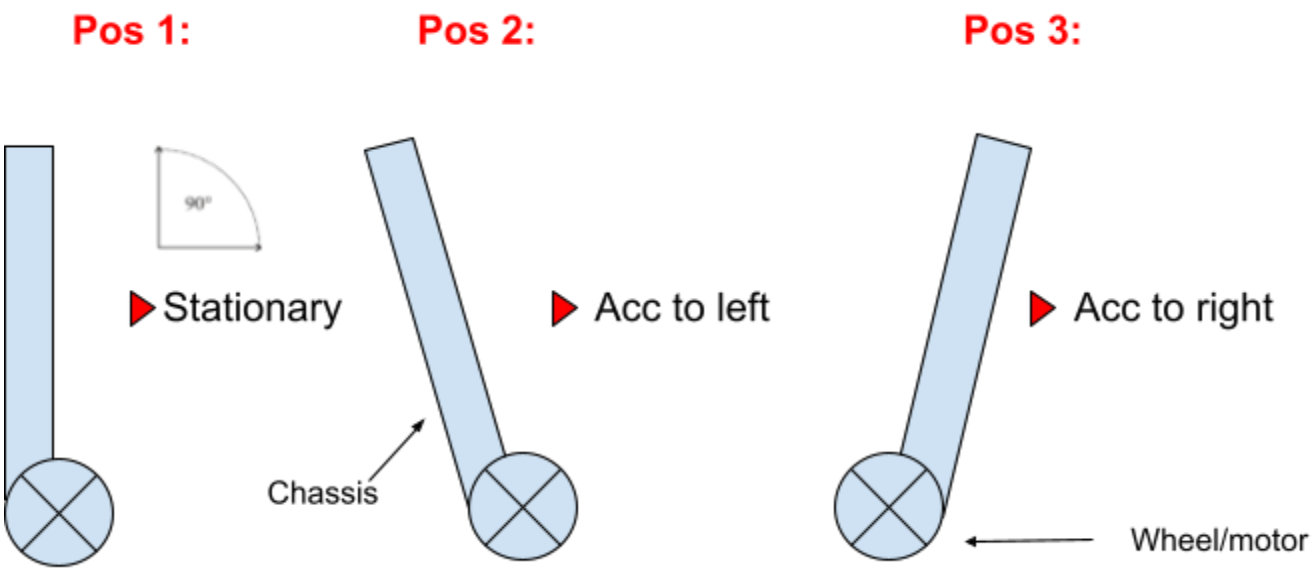


Figure 4

When the chassis is vertical to the wheal, the motors will not move (Pos 1). When the chassis tilts to the right or left (Pos 2/3), the motors will accelerate to the same direction the chassis is moving in until the wheels have a greater speed than that of the tilting chassis which will return it to the state being perpendicular to the wheel. So, there are three variables that need to be controlled here.

1. **Balance Control (vertical position):** Controlling the forward/backwards movement in order to maintain the upright balanced position.
2. **Speed Control:** Adjusting the inclination of the robot by controlling speed of movement.
3. **Direction Control:** Controlling the direction change of the robot by controlling the **"**Differential rotation" between the two motors.

Mathematical equation of the PID Algorithm:

In order to apply the PID algorithm for regulating the above three variables. A deeper understanding of the mathematical equation of the algorithm is necessary.

- The mathematical expression of the PID algorithm is the following:

$$u(t) = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(t')\,dt' + K_\mathrm{d} \frac{de(t)}{dt}$$

(Honeywell, 2000)

Term k = k gain coefficient that tunes how sensitive the system is to each of the paths.

- **Kp** = Proportional gain;
- **Ki** = Integral gain;
- **Kd** = Derivative gain;
- **dt** = difference in time between control cycles;
- **e(t)** = Error;
- **de(t)** = previous error;

Graphical Relationship of the PID Components:

Proportional Control Component: $K_\mathrm{p} e(t)$

The mathematical representation of proportional control in the algorithm is Kp * e(t). Kp is the scale factor, and the scale factor is the magnification K of the straight line passing through the Coordinate point (0,0). The larger the K factor is, the larger the slope of the straight line. The equation of the line is y = k * x, where k is the scale factor of kp, so in fact its y = kp * X, where X is the difference between the measured value of the current sensor and the target value. For short, the error is e(t), derived by this equation e(t) = currentValue - totalValue. The result of the equation y, is the output value u(t) of the control algorithm. If only the P component of the controller is to be used, u(t) = Kp * (currentValue - totalValue), it will lead to as shown below in figure 5 to severe oscillation and on the physical robot, severe vibration of motors (constant back and forth). A graphical interface was used to print the values of the sensor and motor encoder output of the robot on a graph shown below. The following graphs were all live updates from the robot.

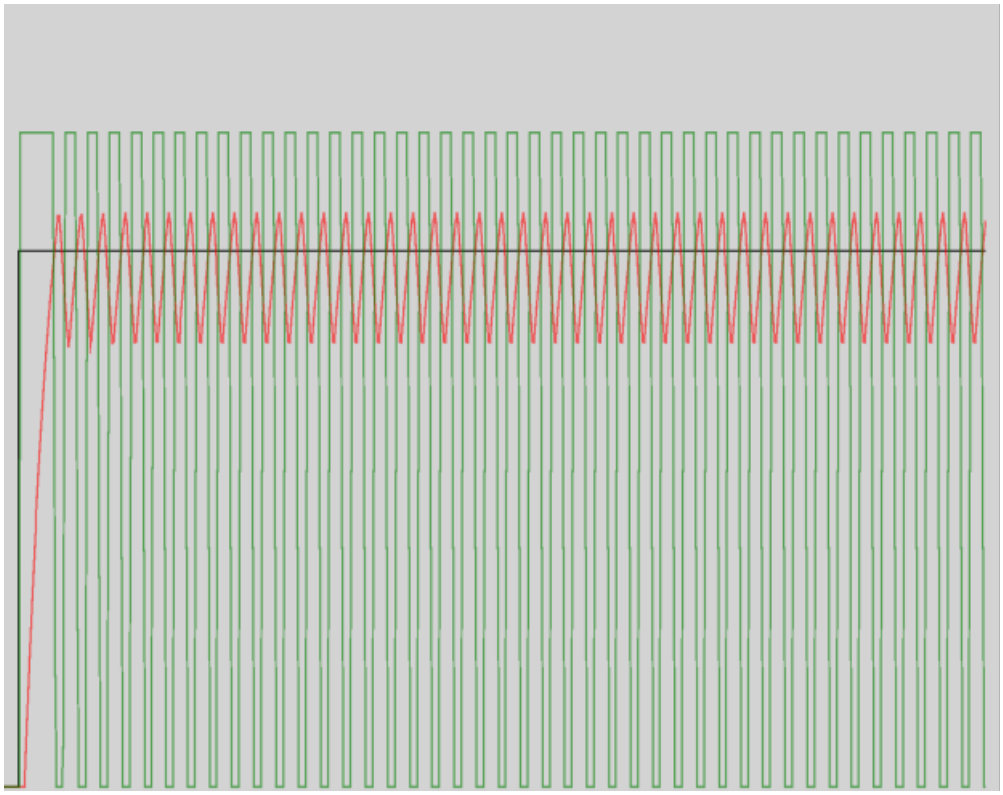


Figure 5: High P gain creates oscillation around target value

Black line is set point (90 degrees from ground)

Green line is the output % of both motors

Red line is the actual inclination value

Proportional control is accelerating the robot forward or backward through motors moving in positive or negative directions. Since vertical stability is needed, assume the balanced position (being 90 degrees to horizontal ground) is A, and the robot is currently in vertical position B (other than A), then Error = A - B, and the amount we need to move motors by at this time is Kp * Error.

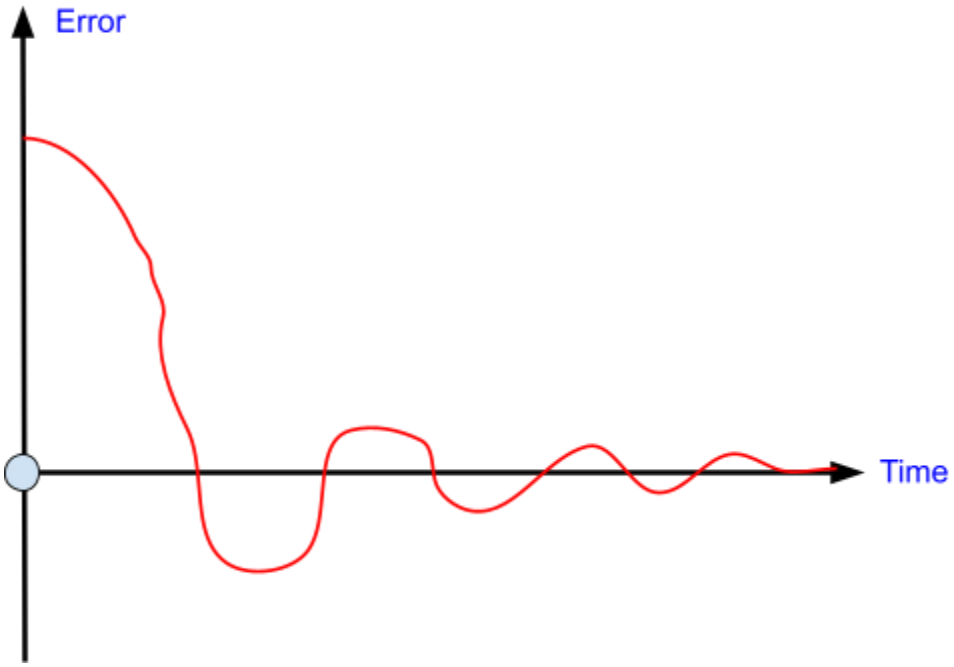
If the error is positive, meaning A > B, then the robot needs to move motors positively in forward direction and vice versa. The system will always be adjusted by moving motors forwards and backwards, as long as there is an error. Having Kp alone control the robot is problematic. If kP is large, the robot will move rapidly and will most likely overshoot position A, and it will repeat the same movement forever (thus having oscillation in the simulation). This is why additional components are required to further help the robot understand its proximity from the balanced position.

Differential Control Component:

$$K_{\mathrm{d}} \frac{de(t)}{dt}$$

The derivative component in the equation is the difference between "the current and the previous errors, divided by the time between the two" (DawsonMU, 2016). The best way to explain D gain, is by thinking of it as a spring. If a spring is pulled and let go of, it will oscillate for a long time until it stops in the balance position again. If, however, the string was to be under water, the time for the spring to reach the balanced position again would be significantly smaller. Because of the Dampening effect in water, the rate of change of the controlled physical quantity tends to zero. The D gain (kd) reduces any oscillation from the P gain by slowing down the rate of change of the current value. When the current value is changing rapidly, the D term moves in the opposite direction just as quickly, acting like a dampener or the breaking force (see figure VIII).

Thinking about it in a more mathematical sense, as the P gain reduces the error to zero (assuming here that Error is positive), the curve of the rate of change of the error over time will be less than zero. The greater the error is the greater the absolute value of the d gain. Eventually, the "reduction of error will be inhibited", and the process would eventually stop as the slope of the line reaches zero. The Diagram below shows the graphical relationship:



Adding the D term to the robot code, it could be seen how the D gain stabilizes the robot.

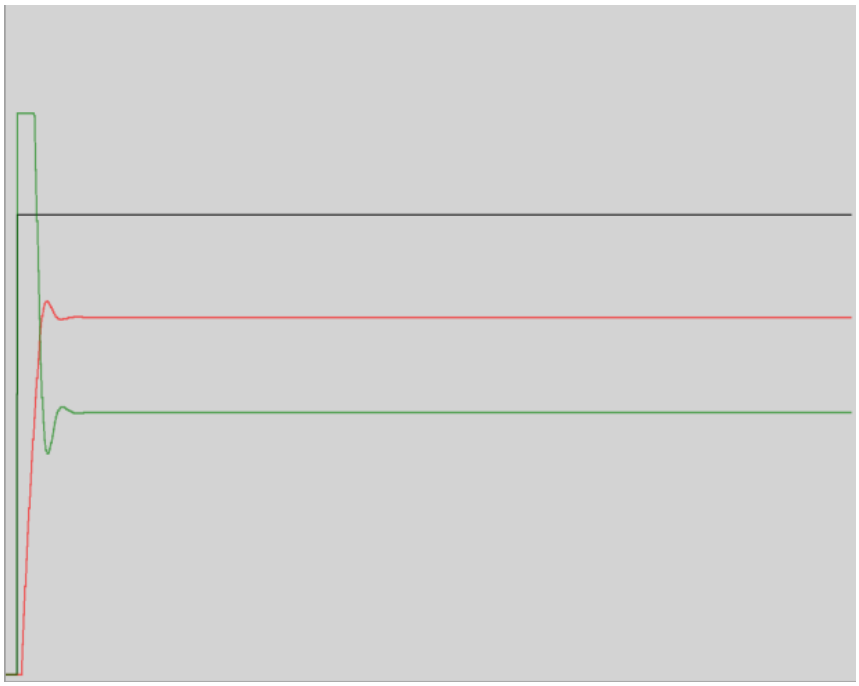


Figure 6: Dampening of error after addition of D term

Integral Control:

$$K_{\mathrm{i}} \int_0^t e(t')\,dt'$$

When any constant, external forces exist in the system, such as gravity or tension, the PID controller will bring the current value towards a point that is not the target value. This constant difference between the current and target values is called the Steady State Error. The Steady State Error can be reduced by ‘accumulating’ or summing up the error over time into the I Term (kI) until it cancels out the external force.

Since the proportional control is proportional to the size of the error, the smaller the output is, the closer the Error is to zero. Because the output is zero when there is no error, it is near impossible to eliminate the error completely and make the controlled U (t) reach the given value. There are disruptincies that could drastically change the outcome such as a rough surface, a low battery, an inclined surface, and many other variables. In order to count for their discrepancies, there must be a stable error to maintain a stable output in order to keep the output stable. Strengthening the proportional function can only reduce the static error, but never eliminate it. Therefore, the introduction of integral function can reduce the error under static conditions, as close as possible to the target value, but when integrating, attention should be paid to setting the integral limit to prevent the amount of integration too large to control.

As discussed above, through proportional control, the Error gets adjusted to 0 as much as the system allows, and differential control makes the slope of the Error curve reach 0. However, if the slope of the error curve is 0, it does not mean that the Error is also zero. In order to reach zero error, mathematically, the area enclosed under the curve and the X-axis should be equivalent to zero as shown in the figure below.

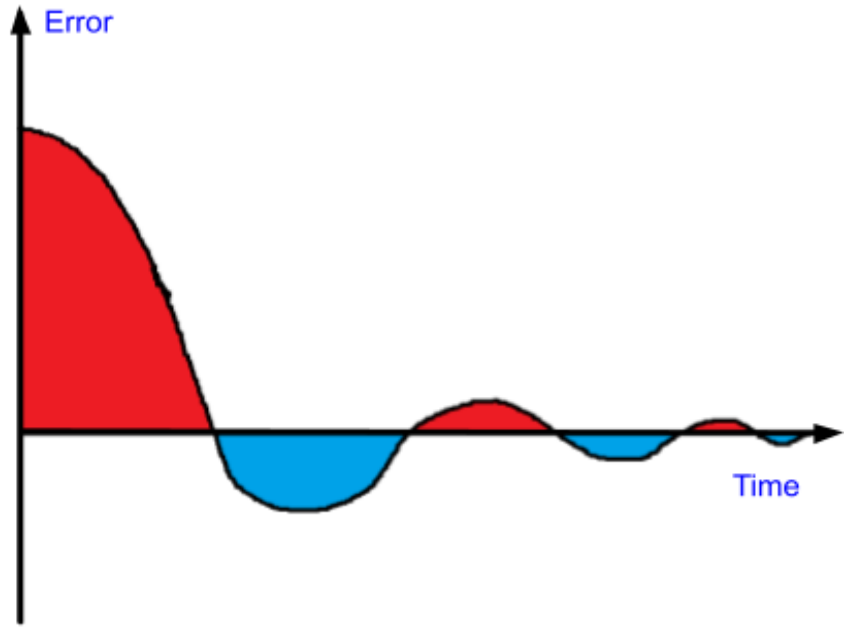


Figure 7

The sum of area under the curve of whatever is above the X-Axis (red) and area under the curve of whatever is under (Blue) should be equal to zero. That is why the Integral of e(t)dt is taken from 0 to t. Until the error value reaches zero, the Integral will always keep the system moving. It is also important to note that high I Terms tend to lead towards instability, especially when there's sensor lag as shown in the figure below.

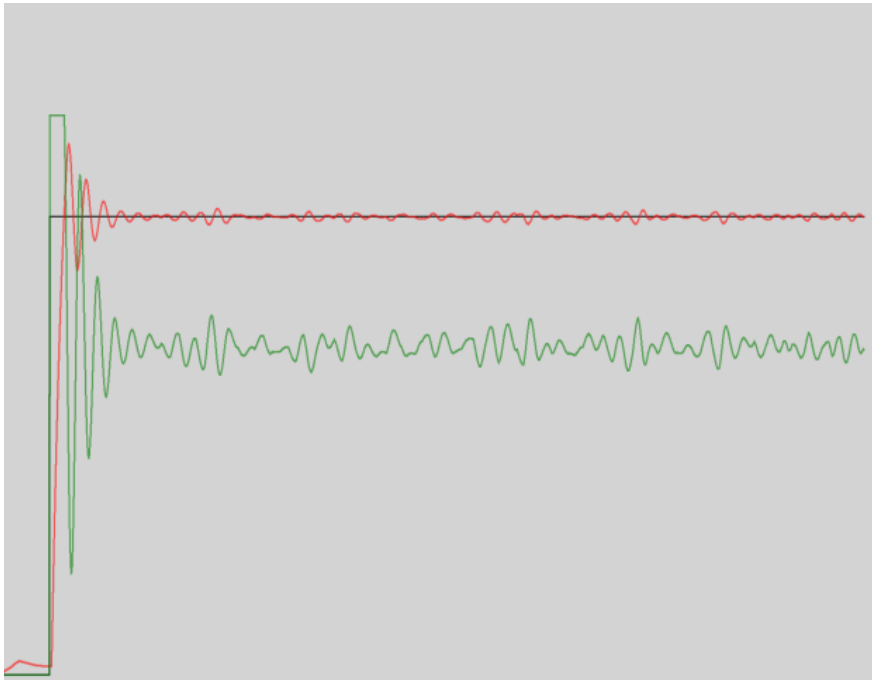


Figure 8: Addition of High I gain to PD controller destabilizes the output

Summing up the 3 components, Proportional, Integral and derivative, would give us an output u(t) that reaches zero error. Each portion of the algorithm compensates for the other's deficiencies (See figure below).

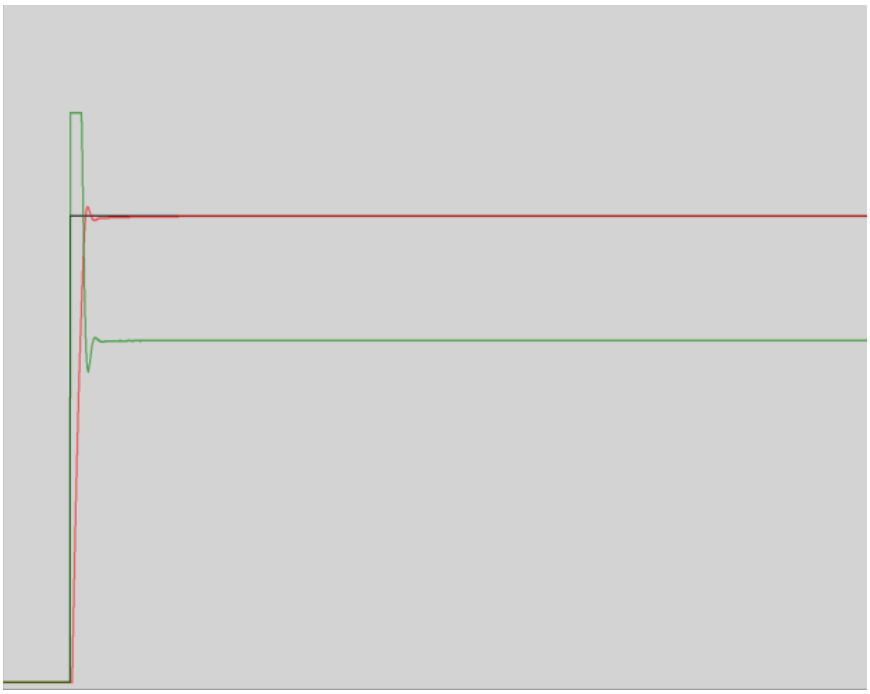
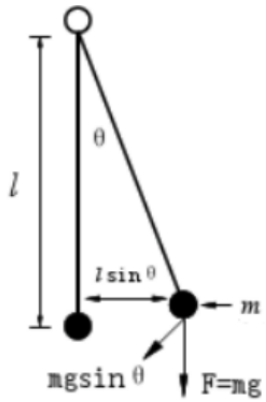


Figure 9: Sum of PID components results with a perfect stability

Mathematics Behind vertical Stability:

Generally, PID of robot related devices is recommended to omit integral link, which is mainly due to the existence of integral saturation. When the static error is too large, even though the device reaches the limit stroke (such as the servo motor turning to the maximum angle position or the DC motor running to the maximum speed), the error cannot be eliminated. In this example of the self balancing robot, assuming that the robot has become stable at a certain time, and suddenly there is a problem with the sensor, which detects too large or too small data, so that the control system mistakenly thinks that the robot has a great deviation from the target, resulting in a sudden increase in static error, an increase in the integral term and an acceleration of motor speed. Then the robot loses balance. So in order to simplify the problem, using PD control alone is most efficient.

In order to understand how to achieve vertical balance, the vertical control model needs to be compared with a simple pendulum model.



$$F = \ -mgSin(\boldsymbol{\theta}) \approx \ -mg(\boldsymbol{\theta})$$

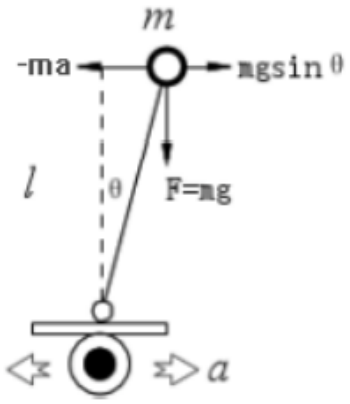
F: Drive eight back to balance position

$-\ mgSin(\boldsymbol{\theta})$ = net force on the bob tangent to the arc

The weight mg has components mgcosθ along the length and mgsinθ tangent to the arc.) Tension in the robot exactly cancels the component mgcosθ parallel to the string. This leaves a net restoring force back toward the equilibrium position at θ = 0.

Under this force, the simple pendulum will move periodically, but it will be damped by the air when it moves, so it will eventually balance. Sin θ ≈ θ. Mathematically, when θ is very small, it can be approximately equal to sin θ, which is convenient for calculation. sinθ and θ differ by about 1% or less at smaller angles). Thus, for angles less than about 15 degrees, the restoring force F is $-\ mg(\boldsymbol{\theta})$.

Taking this concept a step further and applying it on the robot model, we get the following.



$$F = mg\,Sin(\boldsymbol{\theta}) - ma \approx mg\boldsymbol{\theta} - mk_1\boldsymbol{\theta}$$

Suppose it is a negative feedback regulation, and the wheel acceleration a is positively proportional to the inclination angle, and the proportion is K1. In order to stabilize faster, the damping force also needs to be increased, so the formula can be changed as follows:

$$F = mg(\boldsymbol{\theta}) - mk_1(\boldsymbol{\theta}) - mk_2(\boldsymbol{\theta}')$$

the wheel acceleration control algorithm can also be obtained through this following equation:

$$a = k_1(\boldsymbol{\theta}) + k_2(\boldsymbol{\theta}')$$

Therefore, in order to achieve vertical balance of the vehicle, it is necessary to measure the inclination angle θ and angular velocity θ of the vehicle model. The limitation of the MPU6050 sensor is that it has no "accumulated error in the acceleration value of the three axes" (Oftek, 2022) so the way to calculate the inclination angle would be through calculating the tan(). This however will contain great noise since the acceleration will be generated when the robot moves, and the vibration will be generated when the motor is running. This won't allow it to be used directly. The gyroscope has little impact on the external vibration, with high accuracy, and it can not be used directly by calculating tan(). The angle of inclination can be obtained by the integration of angular velocity, but there will be accumulated error. Therefore, in order to obtain the accurate obliquity and angular velocity values, we need to use the Kalman filter. Kalman filters are used to optimally estimate the variables of interest when they can't be measured directly, but an indirect measurement is available.Kalman filter has a good performance in filtering noise, which could be produced from sensor

inaccuracy or sudden physical change of the state of the robot. The Kalman filter parameter setting are as following:

{

dt = 0.005, Q_angle = 0.001, Q_gyro = 0.005, R_angle = 0.5, C_0 = 1, K1 = 0.05;

}

MPU6050 obtains the angle of inclination and angular velocity of the car through the Kalman filter. Kalman filtering is an algorithm that "provides estimates of some unknown variables given the measurements observed over time." (Oftek, 2022) The algorithm is primarily used with systems that produce noise or have built in inaccuracies. Motorized vehicles produce a lot of noise, so Kalman filtering is a relatively popular method used in navigational vehicles such as cars and possibly aircrafts. Although it has been able to maintain balance under the control of the above vertical ring, due to the accuracy problem, the actual measured angle of the sensor always deviates from the angle of the vehicle model, so the vehicle model is not actually vertical to the ground, but there is an inclination angle, under the effect of gravity, the vehicle model will accelerate towards the inclined direction. So we need to introduce a speed loop to maintain the speed stability, so that the balanced car can stay still and move steadily. To achieve this, a new KI controller was created. The code snippet is pasted below:

```
void Balanced::PI_SpeedRing()
{
  double car_speed=(encoder_left_pulse_num_speed + encoder_right_pulse_num_speed) * 0.5;
  encoder_left_pulse_num_speed = 0;
  encoder_right_pulse_num_speed = 0;
  speed_filter = speed_filter_old * 0.7 + car_speed * 0.3;
  speed_filter_old = speed_filter;
  car_speed_integeral += speed_filter;
  car_speed_integeral += -setting_car_speed;
  car_speed_integeral = constrain(car_speed_integeral, -3000, 3000);

  speed_control_output = -kp_speed * speed_filter - ki_speed * car_speed_integeral;
}
```

- **speed_control_output = -kp_speed * speed_filter - ki_speed * car_speed_integeral**:

KP * speed: Since it is speed control, it is of course the speed is multiplied as a proportional parameter to maintain the speed in a stable state.

Ki * displacement: Ki is the integral link, and the velocity integral is the displacement, so the displacement control parameters are selected.

- **speed_filter = speed_filter_old * 0.7 + car_speed * 0.3**

Carries out low-pass filtering to slow down the speed difference and disturb the upright.

- **speed_filter_old = speed_filter;**
- **car_speed_integeral += speed_filter;**
- **car_speed_integeral += -setting_car_speed;**

Integrating speed

- **car_speed_integeral = constrain(car_speed_integeral, -3000, 3000);**

Limit the integral value to determine the maximum upper limit of the speed.

The final piece for achieving balanding would be controlling the steering speed. For this a PD control will be used. The realization of the steering ring is mainly to obtain the data of the z-axis gyroscope as the steering speed deviation for P control. The goal is to keep the steering speed as the set value.

The only thing that is left in this is combining the 3 separate algorithms together and defining the relative P, I, and D values for each algorithm. Upon logical testing of values, the robot achieved stability with:

kp_balance = 55, kd_balance = 0.75;

kp_speed = 10, ki_speed = 0.26;

kp_turn = 2.5, kd_turn = 0.5;

(note the robot can actually move faster but this is slower and more accurate)

Conclusion:

To conclude, PID is a closed loop control algorithm that serves the purpose of calculating the desired set output in a given timestep. In order to apply the algorithm onto self-balancing robots, it is needed to create three separate PID controllers (not necessarily including all 3 components). A PD control for vertical bance, PI

controller for speed control, and PD controller for steering control. The 3 algorithms work together to ensure the robot is vertically balanced at all times. Determining the P, I, and D values was the hardest of the process. A future improvement could be using the ziegler method. Ziegler method relies on using predetermined PID values, instead of tuning them the way they performed throughout this paper.

Works Cited:

Dawson, Zachary, et al. *Lab 4: Self-Balancing Robot - Carnegie Mellon University*. 2016, https://www.cs.cmu.edu/~16311/current/labs/lab04/previous_versions/handout2016.pdf.

NA, N. "Understanding and Realization of PID Algorithm." *Understanding and Realization of PID Algorithm - Fear Cat*, 2018, https://blog.fearcat.in/a?ID=00650-b36fde60-6d85-47c9-bd02-3fa27a25307f.

c, Dragomir. "Dynamic Stability." *Dynamic Stability - an Overview | ScienceDirect Topics*, 2016, https://www.sciencedirect.com/topics/engineering/dynamic-stability#:

Oftek, Karthis. "MPU6050 Principle Detailed Explanation and Example Application." *MPU6050 Principle Detailed Explanation and Example Application - Fear Cat*, 2016, https://blog.fearcat.in/a?ID=00600-51151d10-df4-41a1-bb17-7d4116f1fed6.

Selgado, Mario. "PID Theory Explained." *NI*, 2022, https://www.ni.com/en-us/innovations/white-papers/06/pid-theory-explained.html.

Honeywel, Desborough. "CDs 101: Principles of Feedback and Control." *CDS 101, Principles of Feedback Control*, Caltech , https://www.cds.caltech.edu/~murray/courses/cds101/fa02/.

Teow, James. "Understanding Robot Motion: PID Control." *Medium*, Medium, 1 June 2018, https://medium.com/@jaems33/understanding-robot-motion-pid-control-8931899c31df.