# Robust Controller Design and Implementation for a Step Climbing, Self-Balancing Robot

Kwadwo Akompong Xu Mo Zach Nobles Shivani Saboo Yuan Qian

Carnegie Mellon University
24774: Advanced Control Systems Integration

***Abstract*—There is an ever growing need of automated workers. Two wheeled self balancing robots are a favorable choice with the most common application to deliver items. There is considerable research in balance and path planning of a two wheeled robot, however most research assumes a level surface. This limits the field of application and this is the gap we want to address. The aim of this project is to design a controller robust enough to climb a step treating the impact as a disturbance. In this report we demonstrate the implementation of a PID and LQR controller to enable the robot to successfully climb up steps upto 24% of wheel diameter. We further test the capability of the robot to climb steps with off center weights.**

***Index Terms*—Self-Balancing, LQR, PID, Two Wheeled Robots, Matlab, Simulink, Arduino**

## I. Problem Description

To develop control algorithms that enable a self-balancing robot to climb a step while balancing an external weight.

We look to have our two-wheeled robot achieve the following with our control algorithms:
(i) Ensure smooth motion while the robot is balancing and climbing the step.
(ii) Produce consistent, reliable results while climbing steps.
(iii) Capable of balancing climbing steps with off-axis weights.

### A. *Motivation*

With the pandemic and labor shortage there is a increased demand for automated workers. There is also an growing demand for automated package delivery, Two wheeled self balancing robots are uniquely suited to fulfill this demand. They have the advantage of occupying low floor space and simple mechanical design while maintaining high maneuverability.

Most research into two wheeled robots concerns itself with motions planning and weight balancing while assuming level floor. This highly limits the field of application. Steps are common obstacles that are probably present in field of application. It is important to extend capability of a self-balancing robot to enable it to climb a step.

The challenges associated with stair climbing must be handled by using a controller capable of quick real-time adjustments to the changing dynamics involved with travelling up steps. We seek to design and implement robust controllers capable of allowing a mobile robot to ascend up a flight of stairs while carrying a payload of variable mass. We will use the Elegoo Tumbller as our mobile robot to test our controllers. This mobile robot has IMU, gyroscope, accelerometer, and ultrasonic sensors from which we can gather information about the environment and our mobile robot's state.

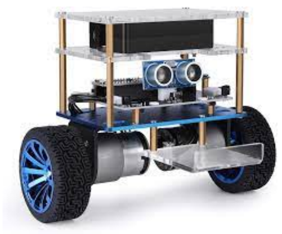

Fig. 1: Elegoo Tumbller.

### B. *Existing Literature*

A significant amount of research has been performed on the dynamics and navigation of a two-wheeled mobile robot. Only some research is available in the case of a two-wheeled mobile robots travelling up a flight of stairs.

Romlay et al discuss the various control methods of balancing two-wheeled mobile robot by employing linear and non-linear controller and adapting and self-learning algorithm [1]. They also explore existing methods of navigating or obstacle avoiding algorithms used to complete tasks such as transporting and monitoring.

The simplest way to balance a two-wheeled robot while carrying weight is to use PID control as demonstrated by Ihrfelt and Marin, who implemented a PID controller over WiFi, with inputs from a gyroscope and accelerometer, with a response time of 10-20 milliseconds [2].

To develop navigation capabilities T. Takei et al created an automatic baggage transportation system with two cooperative subsystems: a balancing-and-traveling control subsystem and a navigation subsystem. Change in center of gravity by a loaded baggage is considered when applying the linear state feedback control method for balancing and traveling. [3]

Bannworth et al worked on making a viable control for enabling a two-wheeled self balancing robot climb a step. They use an auxiliary reaction wheel actuator to do so. A mathematical model of step ascent was derived to create feedback liberalization controller [4]. They start the step from static position and modeled a reference trajectory for the robot. Their success rate was approximately 85%.

Chen et al Derived equations of motion for the system and a controller designed taking into account the initial conditions and possible reference trajectories [5]. They also found out, through simulation, the limiting conditions of the developed controller including factors such as wheel slip and motor torque limit. They managed to climb a step of height up to 5.7cm over 4.5s which is approximately 14% of wheel diameter.

Most step ascension robots use a change in structure of a simple cart pole mechanism or at least a assume knowledge of step height. With this project we wished to keep the structure of the Tumbller as a basic cart-pole system and implement robust control to measure the highest step that can be climbed without tumbling. We assume that the robot approaches the step with some velocity and treat the impact of the step as a disturbance. By employing PID and LQR control we maintain the angle of the robot and the desired speed.

## II. System Modeling

The modeling of a Tumbller is similar to that of an inverted pendulum. The classic simplified force diagram of an inverted pendulum is shown below in Fig. 1 [6].

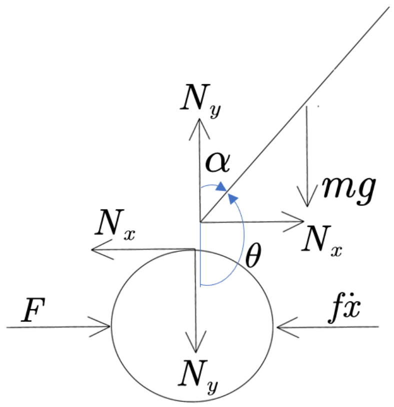

Fig. 2: Inverted pendulum force diagram

### A. Nonlinear Model

The equation for the cart at the bottom (corresponding to the base wheels of Tumbller) is [6]

$$M\ddot{x} = F - f\dot{x} - N_x \tag{1}$$

where $M$ is the mass of the cart and $x$ is the horizontal displacement of the Tumbller; also, the $f$ is the ratio of friction and the $N_x$ is the x-axis force between the pendulum and the cart.

Furthermore, $F$ is the input force generated by the motors in two wheels. It can be calculated from the voltage in motors [8].

$$F = \frac{2k_T}{Rr}V \tag{2}$$

where $k_T$ is the motor torque constant, $R$ is the motor resistance, $r$ is the wheel radius and $V$ is the motor voltage.

The equation for the above pendulum (corresponding to the body part of tumbller) is [6]

$$I\ddot{\theta} = -N_x l\cos\theta - N_y l\sin\theta \tag{3}$$

where $I$ is the moment of inertia, $\theta$ is the angle between the pendulum and the vertical axis, the $N_y$ is the y-axis force between the pendulum and the cart and $l$ is the length to pendulum center of mass, which is half of the length of pendulum.

To get what $N_x$ and $N_y$ represent, the following x and y component equations are obtained [7].

$$m\ddot{x} = N_x - ml\ddot{\theta}\cos\theta + ml\dot{\theta}^2\sin\theta \tag{4}$$

$$N_y = ml\dot{\theta}^2\cos\theta + ml\ddot{\theta}\sin\theta + mg \tag{5}$$

Combine equations (1) and (3), we get

$$(M+m)\,\ddot{x} = F - f\dot{x} - ml\ddot{\theta}\cos\theta + ml\dot{\theta}^2\sin\theta \tag{6}$$

Combine equations (2), (3) and (4), we get

$$\left(I + ml^2\right)\ddot{\theta} = -ml\ddot{x}\cos\theta - mgl\sin\theta \tag{7}$$

Therefore, we get the nonlinear dynamic model (5) and (6) for the Tumbller.

The parameters are referenced by previous labs [8], shown in Table I.

TABLE I: Parameters in modeling

| Parameter | Value |
|---|---|
| $M$ | 0.493 kg |
| $m$ | 0.312 kg |
| $I$ | 0.00024 kg·m$^2$ |
| $l$ | 0.04 m |
| $f$ | 0.01 N·s/m |
| $k_T$ | 0.11 N·m/A |
| $R$ | 10 ohm |
| $r$ | 0.0335 m |

### B. Linear Model

To linearize the model, we first find the equilibrium point at $\theta = \pi$. Also, assume small derivative for $\theta$. To display more simplified equations, set $\alpha = \theta - \pi$. Then,

$$\cos\theta = \cos(\pi + \alpha) \approx -1 \tag{8}$$

$$\sin\theta = \sin(\pi + \alpha) \approx -\alpha \tag{9}$$

$$\dot{\theta}^2 = \dot{\alpha}^2 \approx 0 \tag{10}$$

The linear model is obtained as follows.

$$(M+m)\,\ddot{x} + f\dot{x} - ml\ddot{\alpha} = F \tag{11}$$

$$\left(I + ml^2\right)\ddot{\alpha} - mgl\alpha = ml\ddot{x} \tag{12}$$

Express the model in state space, we get

$$\begin{bmatrix}\dot{x}\\ \ddot{x}\\ \dot{\alpha}\\ \ddot{\alpha}\end{bmatrix} = A\begin{bmatrix}x\\ \dot{x}\\ \alpha\\ \dot{\alpha}\end{bmatrix} + Bu \tag{13}$$

$$y = C \begin{bmatrix} x \\ \dot{x} \\ \alpha \\ \dot{\alpha} \end{bmatrix} + Du \tag{14}$$

where

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-(I+ml^2)f}{I(M+m)+Mml^2} & \frac{m^2gl^2}{I(M+m)+Mml^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-mlf}{I(M+m)+Mml^2} & \frac{mgl(M+m)}{I(M+m)+Mml^2} & 0 \end{bmatrix} \tag{15}$$

$$B = \begin{bmatrix} 0 \\ \frac{I+ml^2}{I(M+m)+Mml^2} \\ 0 \\ \frac{ml}{I(M+m)+Mml^2} \end{bmatrix} \tag{16}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{17}$$

$$D = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{18}$$

$$u = F \tag{19}$$

## III. Controller Design

### A. PID Controller

We first design a PID controller to stabilize the system designed above. A PID controller is a feedback loop controller which compares the collected data with a reference value, and then uses this difference to calculate a new input value. The purpose of this new input value is to allow the system data to reach or maintain the reference value. The PID controller can adjust the input value according to the historical data and the occurrence rate of the difference to make the system more accurate and stable. The proportional unit (P), integral unit (I) and differential unit (D) of the PID controller correspond to the current error, the past cumulative error and the future error respectively. By adjusting the three parameters of the PID controller, the control system can be tuned to try to meet the design requirements. The response of the controller can be expressed by the speed of the controller's response to errors, the degree of controller overshoot, and the degree of system oscillation.

In the stepping situation of the Tumbller, we try to independently control the speed and the upright balance of the Tumbller.

The upright balancing control of Tumbller requires PD control. When the angle is close to mechanical zero, the P controller does not work and the controller thinks that the task completed then. However, the controller ignores a prediction for the next moment, that is, when the angle is at mechanical zero, the motor still has angular velocity. In simple terms, the Tumbller still has inertia. At this time, the D controller needs to be added to predict the deviation at the next moment. The reason why we consider not using integral (I) is that the Tumbller requires rapidity in the balancing control, that is, P is very large in comparison, and when the proportional coefficient is large, the system has almost no static error. The function of integral is mainly to eliminate the static error. The static difference has little effect on the vertical control, so integrals are not needed. In short, the Tumbller does not have to be completely at the mechanical median value. It doesn't matter if it is a little bit off, as long as it is stable.

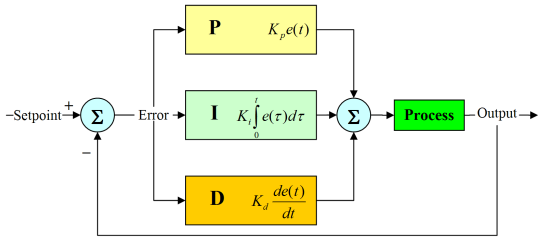

Fig. 3: A block diagram of a PID controller in a feedback loop. $r(t)$ is the desired process value or setpoint $SP$, and $y(t)$ is the measured process value $PV$.

The reasons why we also use a speed control in the mean time is that there is an error in the mechanical median of the Tumbller. If there is only balancing control, the Tumbller may keep going forward or backward to maintain this error mechanical median. The Tumbller can't stand still and balance for a long time mainly because the center of gravity is unstable, which causes the Tumbller to maintain balance only through unidirectional movement. Adding the speed loop is to correct the angle of the Tumbller. When the Tumbller is driving forward, the speed loop makes the Tumbller move at a faster speed to make the "pendulum" part angle backward. At this time, the balancing control makes the Tumbller back to maintain balance. The speed deviation of the speed control is caused by the movement of the vertical control motor, that is, the speed noise is very large, so the differential (D) control is not added. For high accuracy requirements, PI control can be used when the system is noisy.

We combined features of the two PID controllers, to enforce high reactivity to pendulum angle and low reactivity to the angular velocity of the pendulum, the following gains are selected:

$$\begin{aligned} kp_{balancing} &= 55 \\ kd_{balancing} &= 0.75 \end{aligned} \tag{20}$$

$$\begin{aligned} kp_{speed} &= -6 \\ ki_{speed} &= -0.7 \end{aligned} \tag{21}$$

### B. LQR Controller

In addition to the PID controller, an LQR controller has also been designed to stabilize the system. LQR is a form of linear state feedback which can be used to convert unstable open loop poles into stable closed loop poles via pole placement [9]. These pole locations are created by applying a gain to the state vector, or error in the state vector in the case of trajectory tracking. The general form of a state feedback controller for trajectory tracking is shown in Figure 4 where $K$ represents the gain matrix and $G$ represents the system.

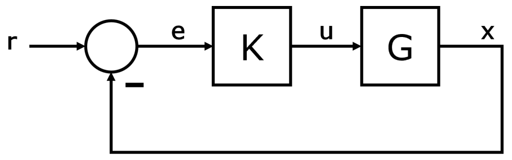

Fig. 4: Block diagram of linear state feedback controller, $K$, applied to plant, $G$, with reference signal, $r$.

In LQR, the closed loop poles are placed in optimal locations derived from a cost function. The LQR cost function have used has the following form, where $J$ is the total cost, $u$ is a vector consisting of each actuator, $R$ is a matrix representing a cost multiplier for actuation, $u$, $Q$ is a matrix representing the a cost multiplier for error in state [10].

$$J = \int_0^\infty (x^T Q x + u^T R u) dt \tag{22}$$

Given the dynamics of the Tumbller and the operating conditions of stair climbing, high emphasis must be placed on reducing error in the pendulum angle. A high emphasis on reducing linear speed will also prove useful in providing the Tumbller with sufficient agility for climbing stairs. The system does not need to react highly to error in position or pendulum angle velocity. For these reasons, the cost matrix $Q$ has been defined as follows:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0.001 \end{bmatrix} \tag{23}$$

The actuator cost matrix, $R$, which happens to be single scalar in the case of the Tumbller, can be defined through simulation trail and error. A high value for $R$ will result in very low actuation, while a low value for $R$ will result in highly reactive, often jittery, actuation. Simulation provides this environment where the trade off between reactivity and oscillation can be weighted. While the actual simulation environment is discussed later in Simulations, the final value for the actuator cost matrix is presented below:

$$R = 10 \tag{24}$$

Now that the cost function for the LQR algorithm has been defined, a gain matrix can be computed to place poles in the desired locations. Matlab's lqr() function is used to compute these gains. The resulting gain matrix, K, is shown below, completing the LQR controller design process.

$$K = \begin{bmatrix} -0.2659 & -2.7780 & 18.7783 & 1.2844 \end{bmatrix} \tag{25}$$

### C. Limitations and Justification for Linear Control Techniques

Given that both controllers boil down to forms of linear state feedback, the shortcomings of linear state feedback should be brought to light. As shown in System Modeling, conversion from a nonlinear model to a linear model required localizing the system. In the case of the Tumbller, the system is linearized about the stationary upright position. While this linear model accurately represents the dynamics at the upright position, it begins to predict less accurate dynamics as the system moves away from that position. Because the controllers are created using local dynamics, the stability of the resulting closed loop system is also a local property. This local stability is called the region of attraction which defines a subspace of the total state space in which the closed loop system is stable. This notion of local stability associated with the controllers will restrict the extent to which our operable state space can span [11]. In the case of the Tumbller, this results in higher limitations on the speed and tilt under which stability is maintained when compared to the ranges in which a nonlinear control design can span. However, for the operating ranges in which we expect the Tumbller to operate under given actuator strength limitations, linear state feedback control should be entirely sufficient.

## IV. Simulations

### A. System

For the purpose of simulation we developed Linear and Nonlinear models in Simulink. The linear models were used to tune the controllers, while the nonlinear model was used to see real system response.

The linear system Was modelled by using a State-Space block with the linear system written in system modeling, the block is shown in Fig. 5.

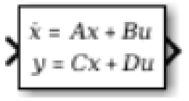

Fig. 5: State-Space Block used to model linear system in Simulink

The nonlinear system was modeled by developing a subsystem that maps the relations of the input, states and constants according to the nonlinear equations. Guidance was taken from [12] to build the state space model. The system is shown in Fig. 6.

It is important to note that the angle reference in the linear system is considered as 0 for upright position. While in the nonlinear system $\pi$ is considered upright. You can see the $\pi$ reference being passed to the angle in control simulations.

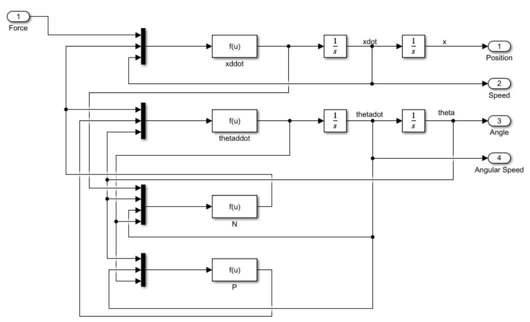

Fig. 6: State-Space Block used to model nonlinear system in Simulink

### B. Reference

With the objective of stair stepping, the Tumbller must change vertical height; however vertical height is not included in the state space. Therefore, we must represent the trajectory in clever form. Visualizing what stair climbing looks like, we notice that as Tumbller impacts the step, the horizontal velocity is brought to a stop momentarily. This is the map we will use to translate the objective into a trajectory. The resulting linear speed trajectory will then consist of a constant speed reference (implemented as a step block), and a saturated impulse signal at the time of impact. While saturation is not required for a fully continuous closed loop system, we must saturate the impulse in the discrete case to avoid sending an impulsive control response over an entire sample period. The resulting reference block diagram is shown in Figure 7 below.

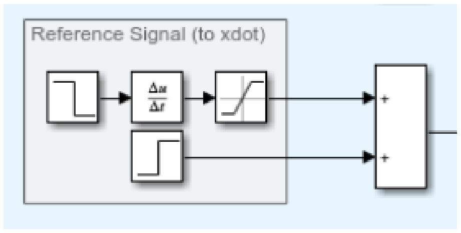

Fig. 7: Reference for $\dot{x}$ to simulate step ascension

### C. PID Controller

We refer to the PID structure shown in Fig. 3 to design two PID feedback structures respectively, and then connect them in parallel. Input the sum of the two controllers into our defined nonlinear system model and use the two variables angle and speed in the output as feedback to complete our loop. This is a continuous time plant, as the hardware reacts as a continuous time system. While the plant is continuous, the controller will operate in discrete time. To emulate the hardware controller, we have implemented a refresh rate to our controller and apply a constant control signal between samples. To accomplish this, we have applied a zero order hold block after the plant.

We set different sampling times for the two controllers. The main reason is that we think that although we need two controllers to achieve a balanced state, the angle controller is undoubtedly the one with higher priority. We decide that the frequency of sampling and updating for the angle controller is higher, so as to prevent the two controllers from conflicting and to better complete the main task. The resulting simulation environment is shown in Figure 9 below. The controller design is shown in Fig. 8.

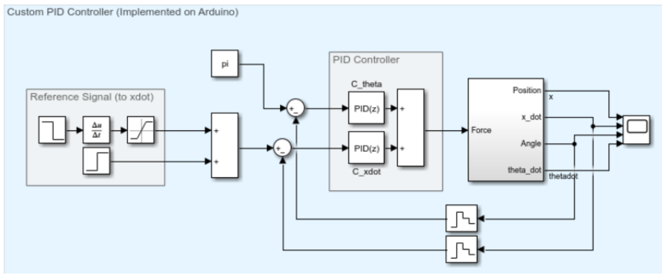

Fig. 8: Simulation environment used to develop and evaluatethe PID controller. The angle control updates every 5ms, whiles the speed control updates every 40ms

Simulation results, displayed below in Figure 9 show that our PID model is capable of stabilize the Tumbller pendulum angle and speed in relatively short time. At the beginning, the angular speed oscillates violently and with a large amplitude, which implies that after the impulse signal simulating the impact, there may be significant jitters for a short period of time to quickly reach a relatively stable state.

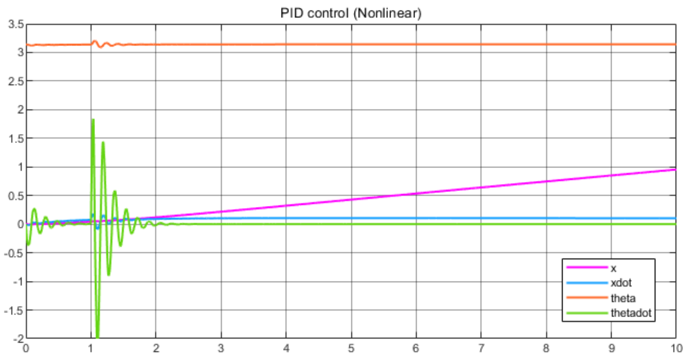

Fig. 9: Simulation response to the simulation environment and reference signals described in Figure 8. The states are defined as: $x$ is the linear position, $\dot{x}$ is the linear velocity, $\theta$ is the pendulum angle, and $\dot{\theta}$ is the pendulum angular velocity. A pendulum angle of $\pi$ represents the upright position.

### D. LQR Controller

To simulate the LQR controller, we have used a similar architecture to that of the linear feedback block diagram shown in Figure 4. The plant is defined as the nonlinear system model. This requires a reference signal of $\pi$ to hold the upright position. As we explained above in PID subsection, to discretize the continuous input for the discrete controller, we apply zero-order holder right before the gain matrix. The resulting control system is shown in Figure 7.

Simulation results, displayed below in Figure 11 show that the simulation shows reliable tracking of the pendulum angle and relatively strong tracking of speed. By nature of the system, a pendulum angle of zero cannot be sustained with a nonzero speed; therefore, we accept the drift in speed, as tracking pendulum angle is more significant for the objective. This response is accomplished with very low mean actuation and a maximum actuation of 2V upon impact with the step.

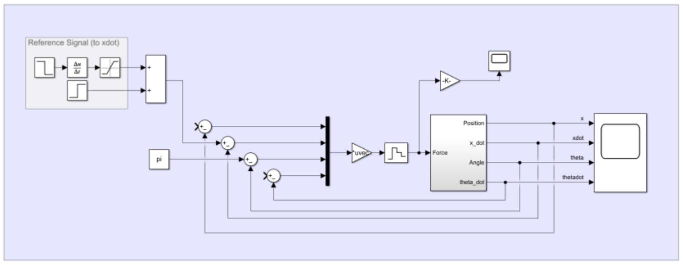

Fig. 10: Simulation environment used to develop and evaluate the LQR controller. The controller is run at a sampling rate of 5ms.

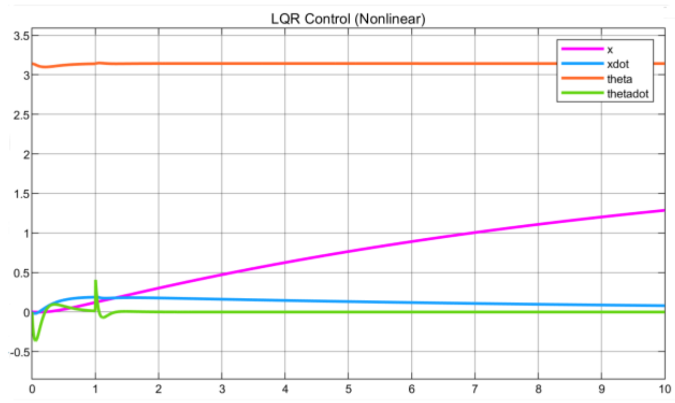

Fig. 11: Simulation response to the simulation environment and reference signals described in Figure 10. The states are defined as: $x$ is the linear position, $\dot{x}$ is the linear velocity, $\theta$ is the pendulum angle, and $\dot{\theta}$ is the pendulum angular velocity. A pendulum angle of $\pi$ represents the upright position.

## V. Hardware Implementation and Testing

### A. Hardware Setup

The Tumbller is controlled and operated using a Elegoo NANO microcontroller. It controls two Hyatech 37-520 motors and calculates it's position and acceleration using a GY-521 gyroscope and accelerometer.

### B. Sensors

The position of the Tumbller was calculated using the encoders attached the Hyatech Motors. Due to the limited capacity of the nano-pins, the precision of the encoder count was halved and the direction of the Tumbller had to be determined using software.

We calculated position using a low-pass derivative filter. The cutoff frequency was determined experimentally by running the Tumbller at constant speeds and then using the filter. The criteria was a balance between clearing noise while minimizing the delay. We found that a LPDF with a cutoff frequency of 30Hz worked well.

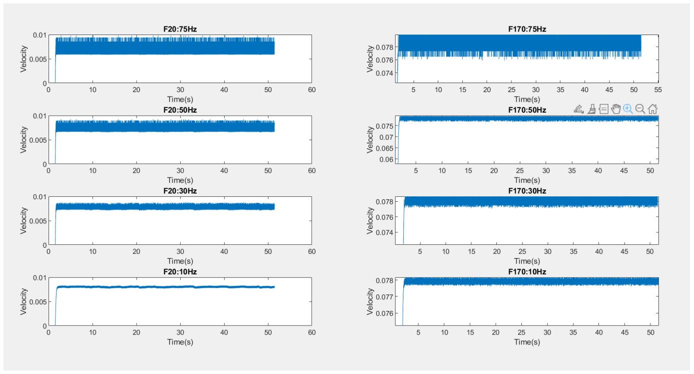

Fig. 12: Experimental Data from various LPDF based on encoder data when the voltage sent to the motor is 0.7V and voltage = 6.0V

The angle of the Tumbller is calculated using the readings provided by the gyroscope-accelerometer. We used the Kalaman filter designed by the manufacturer to convert the angular acceleration and linear acceleration to angular position data. Due to manufacturing issues, there is an inherent error in the data received and processed by the NANO, thus a correction term is add to our reference values to compensate for this.

As with the velocity we calculated angular velocity using a LPDF. We chose our cutoff frequency based on the performance of the control algorithms and ultimately landed on a cutoff of frequency of 50Hz.

### C. Software Framework

The framework of the software is fairly straight forward. Upon being turned on the respective sensors are tuned to their initial values, and the Bluetooth or Serial channel is opened depending on whether it is being run manually with Bluetooth control or autonomously.

After the electronics are initialized, an interrupt occurs every 5ms. Inside the interrupt, the states are updated and the control algorithm sends the new control signal to the motors.

While being used manually control, every 40 ms, input from the user is incorporated into the signal sent to the motors.

### D. PID Controller

For the PID we follow the structure as seen figure 7, where the theta is constantly tuned every 5ms while the speed control is updated every 40ms.

### E. LQR Controller

Upon testing the hardware, we found that the values from our simulations produced slightly different results than what was used in hardware. We altered these values slightly based on experimental results.

We found it useful to have the reference $\theta$ of the LQR slightly off-axis to compensate for the weight of the plate and any weights we placed on-top on the Tumbller.

Finally to allow for better control when the Tumbller is met with a disturbance, we implemented a gain scheduler. We look to increase $\theta$ gain when the $\theta$ is heavily off reference and then increase the velocity gains when the $\theta$ is near the reference but the velocity of the Tumbller is still high.

## VI. Final Demo Results

### A. Balance

As a general performance measurement, we have tested the balancing performance of the PID and LQR controllers designed in the sections above. A baseline for these controllers will be presented using the stock PID controller developed by the Tumbller manufacturers. The results for the baseline controller, our PID controller, and our LQR controller are shown in Figures 13, 14, and 15 respectively.

Comparison of the baseline controller and our PID controller suggests the our PID oscillates less frequently but at a similar speed. While our PID controller shows a small offset of approximately 2 degrees, the Tumbller was able to maintain a zero mean velocity. This leads us to infer that sensor drift has caused the offset and that the angle is in fact zero mean. From the data and this assumption of sensor noise, we conclude that our PID controller outperforms the baseline controller when balancing.

Comparison of the baseline controller and our LQR controller show very similar oscillation frequency and velocity magnitude; however, we see less variation in pendulum angle for he LQR controller. Based on the physical location of the motor encoders and the sensors for angular position, the data suggests that while the two controllers provide equal movement at the wheels, the LQR results in less movement at the center of mass. The leads to the conclusion that the LQR controller outperforms the baseline controller in balancing.

Comparison of the LQR controller and PID controller is less conclusive. The PID controller shows slower oscillation, creating a less jittery balance, while the LQR controller yields less movement at the center of mass.

### B. Stair Stepping

In this part, the stair stepping performance of our controllers is displayed. The results for the baseline controller, our PID controller, and our LQR controller are shown in Figures 16, 17, and 18 respectively.

It can be seen from the angle plots that both custom PID and LQR design achieved the objective of climbing the stair without falling. The angle changes very little, only 2 to 3 degrees in the process of stepping, going through a procedure of reclining, followed by a quick recovery. After recovering, the Tumbller started to moving forward upright, similar to how it behaves in balancing.

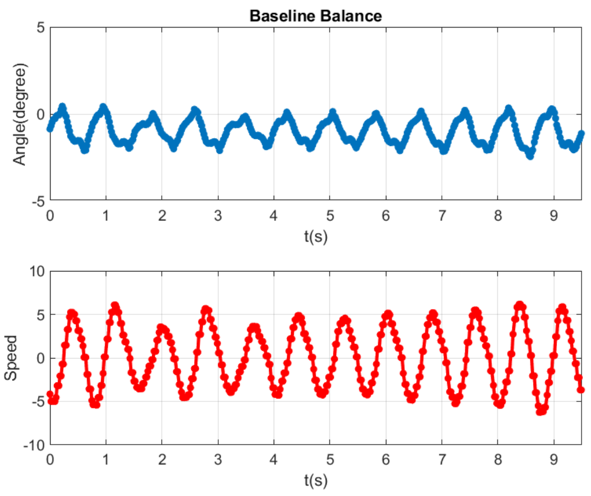

Fig. 13: Undisturbed balancing performance of stock PID controller.

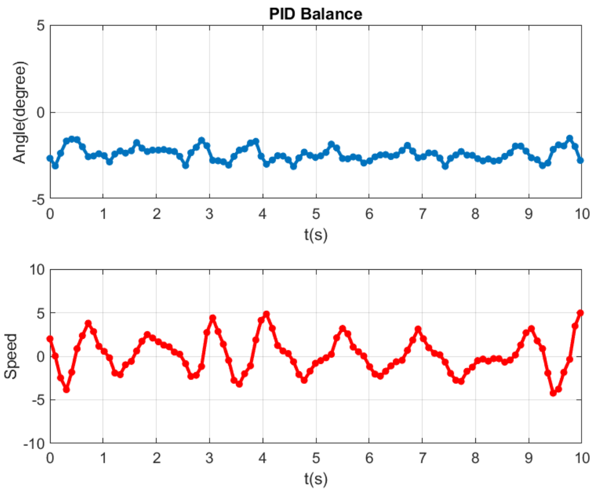

Fig. 14: Undisturbed balancing performance of our PID controller.

Compared the baseline controller, our PID controller reacts a bit slowly. This may not be a conclusive due to variations in the experiment conditions. The behaviors are quite similar in both angle and speed. Both angle and speed trend toward equilibrium, leading to the conclusion that our PID controller successfully dealt with the stepping problem.

Compared the baseline controller, the stair impact on the angle of LQR-based Tumbller is smaller. The decent performance is consistent with the balancing case, which means that the ability of LQR controller to resist external disturbances is strong. Also, the balance position that is slightly tilted back is

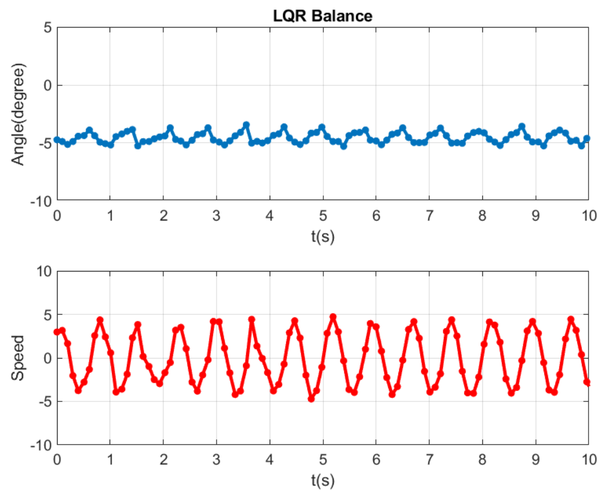

Fig. 15: Undisturbed balancing performance of our LQR controller.

preferable for going upstairs.

As for the two controllers we designed, the speed change seems very similar during the process and the angle performance is also very satisfactory. In the existing scenario, choosing either controller can achieve the desired goal.

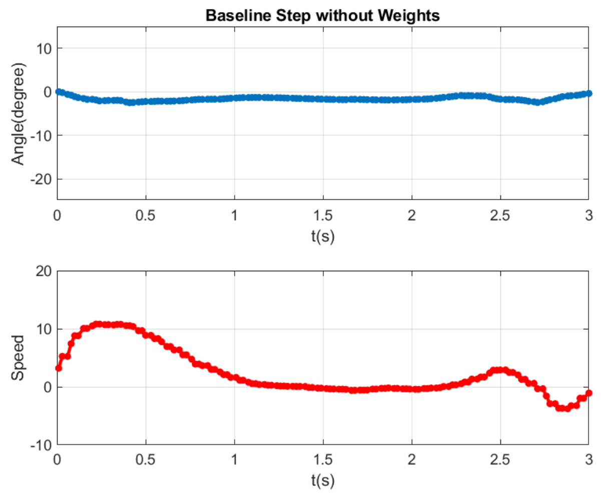

Fig. 16: Undisturbed stepping performance of stock PID controller.

## C. Stair Stepping with Weights

We want to explore the best performance of the controller we designed, so we add weights to the Tumbller in the stepping testing. The weight used in the experiment is an uneven object with a mass of about 100g, as shown in the figure 19. It is placed on a tray in front of the Tumbller in the experiment.

Same as the previous performance testings, we use the built-in PID controller as the baseline controller to evaluate the performance improvement of the custom controllers. The

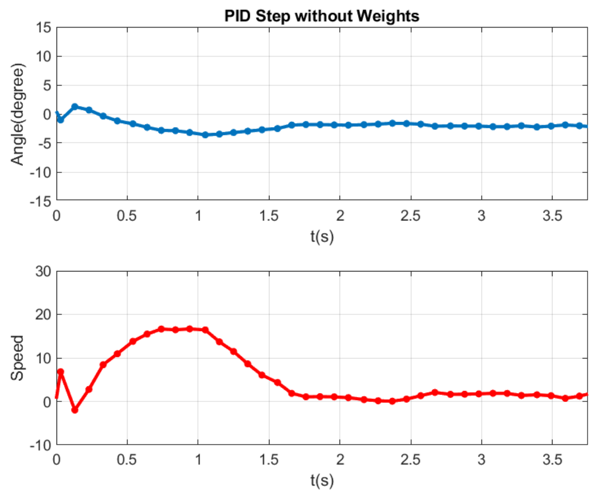

Fig. 17: Undisturbed stepping performance of our PID controller.

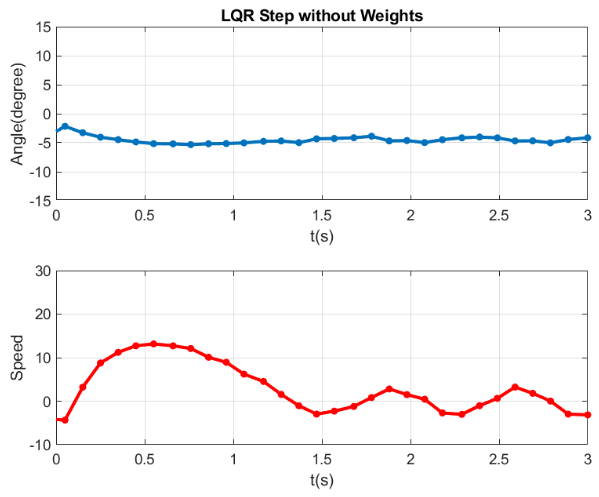

Fig. 18: Undisturbed stepping performance of our LQR controller.

results of the baseline controller, our PID controller, and our LQR controller are shown in the figure 20, 21 and 22.

We first discuss the performance of angle controllers. For the custom PID controller, both during and after stepping shows an offset of about 2 degrees, which implies that Tumbller has a slight backward tilt in order to maintain balance. Considering that there is a load in front of the Tumbller, the mass center of the Tumbller is shifted forward, such a backward condition is acceptable. The entire Tumbller still completes stepping stably, and the average offset of the baseline controller is as high as 12 degrees. This offset is even visible to the naked eye. Our custom PID controller is beter than the baseline controller. As for the custom LQR controller, its average backward offset is about 10 degrees during the entire stepping process, so our custom LQR control is still

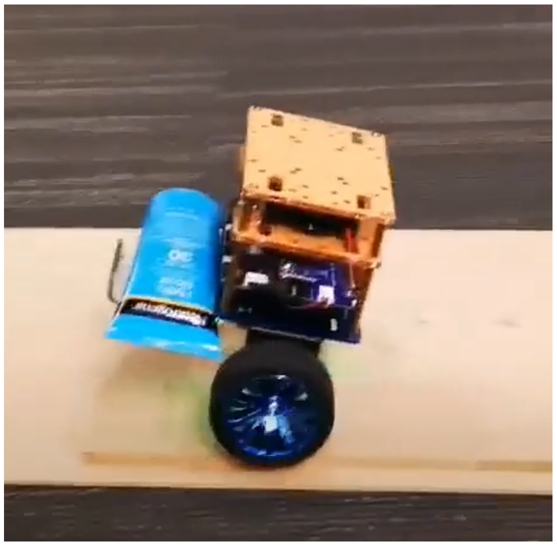
Fig. 19: Weighted Tumbller

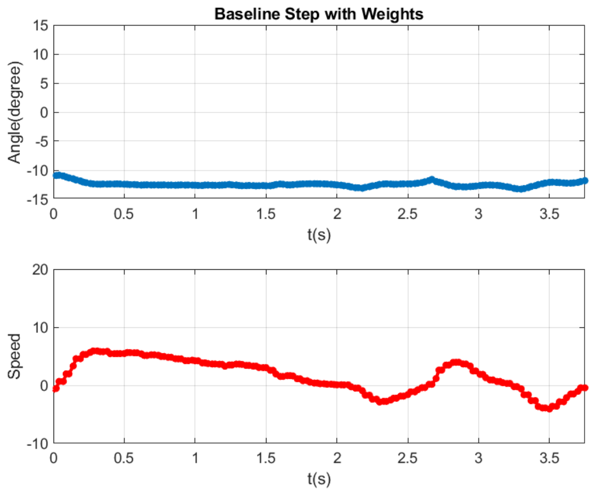

Fig. 20: Undisturbed stepping with weights performance of stock PID controller.

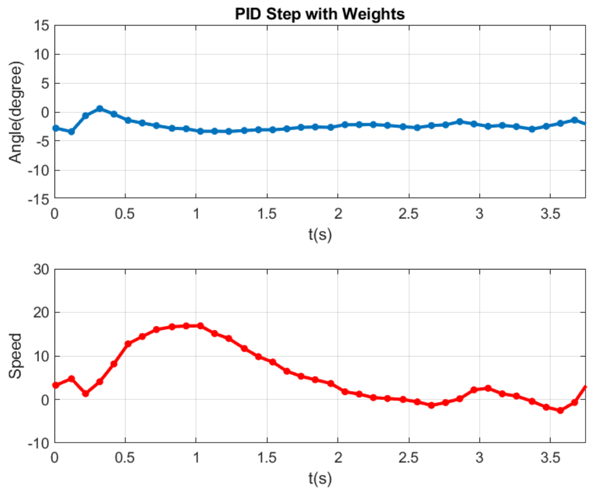

Fig. 21: Undisturbed stepping with weights performance of our PID controller.

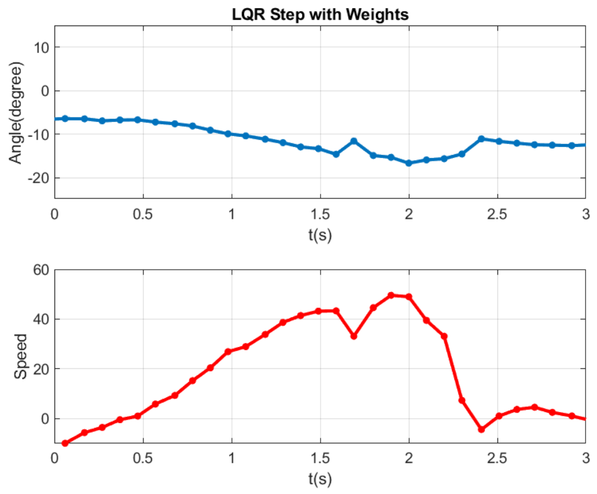

Fig. 22: Undisturbed stepping with weights performance of our LQR controller.

better than the baseline controller.

As for the performance of the speed controller, the performance of the baseline controller and the custom PID control are very similar, which can be seen from the shape of the two curves. Both PID controllers have the ability to allow a Tumbller with weights complete stepping and then stand still on the stair, but it is worth noting that the speed of custom PID is higher than the baseline controller in the first speed peak and lower than the baseline controller in the second speed peak. This shows that our PID controller has two advantages over the baseline controller. First, the speed is higher when hitting the edge, so it can step higher stairs. In the experiment, this data is 0.625 inch versus 0.5 inch. Second, after stepping is completed, the custom PID controller will move a smaller distance before approaching a standstill, that is, the adjustment amplitude will be smaller. This can be derived from the area of the intersection of the second half of the curve and the x-axis. As for the custom LQR controller, its maximum speed is far greater than any PID controller. In fact, it almost completed a stepping of 0.75 inch in the experiment. At the same time, it only takes less than 2.5 seconds to complete stepping, even if the time for adjustment is added, it only takes 3 seconds, and its performance is 25% better than the baseline controller. We also notice that a slight stall occurs at the moment stepping is completed. According to observations, it is speculated that the speed may be too fast and the tires slipped.

A video showcasing the balancing, step, and step with weight tests performed on the PID and LQR controllers has been posted to YouTube and can be accessed via the following link [13].

## VII. Conclusions

To conclude, we met the objective of making a two wheeled self balancing robot climb a step with external weight using robust control. We successfully implemented both PID and LQR controls. The maximum step height climbed for Tumbller is 24% of wheel diameter which is a good accomplishment, given that literature available so far claims climbing 14% of wheel diameter. We also improved upon the base PID controller by 25%. The Tumbller also successfully carried a weight of 100g over the step which is 12.5% of its total weight.

As a future prospect we can introduce path planning control to the system to make the robot capable of detecting steps and maintaining required speed to ascend it. This will also enable to robot to ensure that is approached the step front-on and not at an angle. We also need to work on reliability of the step ascension as there are some cases in which the the Tumbller fails. The controller should also be applied to a larger scale robot to test in life size conditions.

## References

[1] Romlay Muhammad Rabani, Mohd Ibrahim Azhar, Toha Siti,Rashid Mahbub, ”Two-wheel Balancing Robot; Review on Control Methods and Experiments”, International Journal of Recent Technology and Engineering, vol. 7, 2019

[2] F. Ihrfelt and W. Marin, ”Self-balancing robot: WiFi steerable self-balancing robot”, Dissertation, KTH Royal Institute of Technology, 2020.

[3] T. Takei, R. Imamura and S. Yuta, ”Baggage Transportation and Navigation by a Wheeled Inverted Pendulum Mobile Robot,” in IEEE Transactions on Industrial Electronics, vol. 56, no. 10, pp. 3985-3994, Oct. 2009

[4] J. X. J. Bannwarth, C. Munster and K. A. Stol, ”Step ascension of a two-wheeled robot using feedback linearisation,” 2015 6th International Conference on Automation, Robotics and Applications (ICARA), pp. 161-166, 2015

[5] T. Chen, P. Hazelwood and K. Stol, ”Step ascent modelling of a two-wheeled robot,” 2012 19th International Conference on Mechatronics and Machine Vision in Practice (M2VIP), , pp. 310-315, 2012

[6] H. Hanna, S. henrik, ”Two-Wheeled Self-Balancing Robot”, Bachelor’s thesis, KTH Royal Institute of Technology, Aug. 2015. https://kth.diva-portal.org/smash/get/diva2:916184/FULLTEXT01.pdf

[7] Control Tutorials for Matlab and Simulink, Inverted Pendulum: System Modeling, https://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=SystemModeling, 2021

[8] M. Bedillion, ”Laboratory 2: Balancing Robot Control”, Lab handbook in 24-774: Special Topics in ACSI, Carnegie Mellon University, Sep. 2017.

[9] Control Tutorials for Matlab and Simulink, Inverted Pendulum: Root Locus Controller Design, https://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=ControlRootLocus, 2021

[10] Control Tutorials for Matlab and Simulink, Inverted Pendulum: State-Space Methods for Controller Design, https://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=ControlStateSpace, 2021

[11] Khalil, Hassan K., ”Nonlinear Systems”, Macmillan Publ. Co, 1992.

[12] Control Tutorials for Matlab and Simulink, Inverted Pendulum: Simulink Modeling, https://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=SimulinkModeling, 2021

[13] ”ACSI Group 1: Stair Climbing Tumbller.” *YouTube*, https://youtu.be/fxm-dpQ6Rgs, 2021