ROB 550 BalanceBot Report - Team 06

Akhilesh Mohan, Shih-Chi Liao, Oscar De Lima

***Abstract*—In this report, the PID controllers employed for balancing our BalanceBot and for tracking position are elaborated. Odometry based on wheel encoders was calibrated to obtain state-estimates of the robot position, while gyrodometry was employed for heading estimation. Although the simple setpoint march techniques used for trajectory following performed acceptably, there was still room for improvement.**

I. Introduction & Objectives

The objective of this project was to design feedback controllers to stabilize and move a BalanceBot, that was modelled like a mobile inverted pendulum. A state-estimation and navigation system was to be implemented based on readings from an Inertial Measurement Unit and wheel odometry to allow the robot to follow a trajectory. The teams were provided with a Mobile Robotics cape & a Beaglebone Green, and the robot was assembled and system identification was carried out. Manual control of the BalanceBot through a DX6e transmitter was to be enabled before a competition, that included tasks such as a straight line drag race, square driving and manual control over circuits with obstacles. Successful completion of these tasks demonstrate the robot's ability to balance and track trajectories while also testing the robot's speed and responsiveness to manual control. This report elaborates the different components worked on to enable the robot to achieve the aforementioned objectives.

II. Methodology

A. BalanceBot Dynamics

Before we begin to develop controllers for the system, it is important to understand the dynamics of the system. For simplification, the dynamics of the BalanceBot are approximated as a mobile inverted pendulum with a wheel attached to the bottom of the body that is free to rotate about the wheel axis. The derivations of the equations of motion (EOMs) based on the free-body diagram shown in Fig. 7 (in Appendix) were treated in the lecture handouts [1], and hence only the EOMs are presented here.

The motion of the BalanceBot is governed by Eq. (1) and (2), which are derived assuming a no-slip condition between the wheel and ground. Table III provides detailed information on the symbols used and the measured values of the parameters required for this model.

$$\begin{aligned}(m_b R_w L\cos\theta)\,\ddot{\phi} + \left(I_b + m_b L^2\right)\ddot{\theta} \\ = m_b g L\sin\theta - \tau\end{aligned} \tag{1}$$

$$\begin{aligned}\left(I_w + (m_b + m_w)\,R_w^2\right)\ddot{\phi} + (m_b R_w L\cos\theta)\,\ddot{\theta} \\ = m_b R_w L\dot{\theta}^2\sin\theta + \tau\end{aligned} \tag{2}$$

The only input to the system is the motor torque on the wheels which result in a reaction torque on the body, providing a stabilizing effect and preventing the BalanceBot from tipping over. We also include the motor dynamics in our model in order to account for the motor inertia effects and any delays in the transmission of the torque input to the system. The output torque of the motor is governed by Eq. (3), where u is the PWM duty cycle of the motor which is the output of the feedback controllers used and ω is the motor rotation speed measured by the encoders.

$$\tau = \tau_s u - \frac{\tau_s}{\omega_{NL}}\omega \tag{3}$$

However, in practice, since the BalanceBot has two motors, the total motor torque τ used in the EOMs is in fact $\tau = \tau_L + \tau_R$, where τ_L and τ_R represent the torques generated by the left and right motors.

In order to simplify the complex non-linear dynamics and to develop easily implementable linear controllers for the BalanceBot, the equations of motion are linearized around $\theta = 0$. Along with the assumptions that we are dealing with small angles and by setting $\dot{\theta} = 0$, and by further including the motor model shown in Eq. (3), we can obtain the transfer functions $G_1(s)$, from the PWM duty cycle u to the body angle θ and $G_2(s)$, from θ to ϕ, the wheel angle as given in Eq. (4) and (5) where parameters a_1, b_1, etc. are as defined in the Appendix.

$$G_1 = \frac{\Theta(s)}{U(s)} = \frac{-b_1\,(a_1 + a_2)\,s^2}{(-a_2^2 + a_1 a_3)\,s^4 + b_2\,(a_1 + a_3 + 2a_2)\,s^3 - a_1 a_4 s^2 - a_4 b_2 s} \tag{4}$$

$$G_2 = \frac{\Phi(s)}{\Theta(s)} = \frac{-\,(a_2 + a_3)\,s^2 + a_4}{(a_1 + a_2)\,s^2} \tag{5}$$

B. BalanceBot System Identification

1) Motor parameters: In order to incorporate the motor dynamics into the controller several motor parameters had to be calculated.

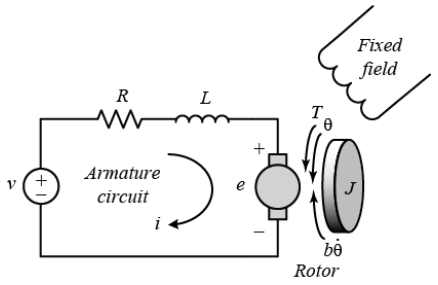


Fig. 1: Schematic of DC motor. R is the coil resistance of the motor, L is the motor inductance, V the applied voltage, e is the back emf generated by the rotor, τ is the torque generated, ϕ is the angular position of the rotor, ω is the angular velocity and b is the coefficient of viscous friction.

The first parameter was the motor coil resistance R which was measured with a multi-meter across the motor positive and negative terminals of the motor. The circuit shown in Fig. 1 can be expressed as

$$V = Ri + L\dot{i} + GK\omega \tag{6}$$

$$\tau = GKi = I_{gb}\dot{\omega} + b\omega \tag{7}$$

where G is the gear ratio and K is the motor constant. If the system reaches steady state the derivative term in Eq. (6) and the $I_{gb}\dot{\omega}$ term in Eq. (7) will equal 0. In the system is stalled, then the angular velocity becomes 0 and

$$i_{stall} = \frac{V}{R};\ \tau_{stall} = GKi_{stall} \tag{8}$$

To calculate the motor constant K, the current at no load, i_{NL}, is measured with the current sense pin from the motor driver. K is then calculated from

$$V = Ri_{NL} + K\omega_{NL}G \tag{9}$$

Knowing K, b is calculated as

$$b = \frac{GKi_{NL}}{\omega_{NL}} \tag{10}$$

To calculate the inertia of the motor shaft I_{gb}, the time constant T of the system needed to be measured. The system is started at the nominal voltage of 12 V and it is allowed to reach steady state. The angular velocity is recorded from this exercise. The value of T is the time that it takes the system to reach an angular velocity of 63% of ω_{NL}.

The Laplace transform of Eq. (6) and (7) gives

$$V(s) = RI(s) + LsI(s) + GK\Omega(s) \tag{11}$$

$$GKI(s) = I_{gb}s\Omega(s) + b\Omega(s) \tag{12}$$

Because the dynamics of the mechanical part of the system are much slower than the electrical dynamics, it can be assumed that T is approximately the time constant of the mechanical system. The transfer function for the mechanical system is

$$\frac{\Omega(s)}{I(s)} = \frac{GK/b}{I_{gb}s/b + 1} \tag{13}$$

First order transfer functions have the form

$$G(s) = \frac{A}{1 + Ts} \tag{14}$$

Since the values of T and b are known, the inertia of the motor shaft can be obtained from $T = I_{gb}/b$.

2) Moments of Inertia: The moment of inertia of the BalanceBot was estimated using a bifilar pendulum. By suspending the BalanceBot along the different axes of rotation and by providing a small initial amplitude of rotation about that axis, a pendulum oscillation period was calculated using the onboard rate gyro. Software based on the `rc_test_dmp` routine from the RC Library was incorporated along with measurement timestamps to evaluate the oscillation period.

The experiment was repeated a minimum of 3 times along each axis until repeatable data was obtained. Post data collection, an FFT was done on the angular velocity data from each run to obtain the time period corresponding to the frequency ($< 1Hz$) with highest amplitude from the FFT.

Plugging the average time-period of oscillations from the runs, T_i about the i^{th} axis into the follow equation, the moment of inertia about each axis was estimated.

$$J_{ii} = \frac{mgd^2}{16\pi^2 L}T_i^2 \tag{15}$$

where m is the mass of the BalanceBot, d and the L are the width and length of the rectangle formed by suspension wires and the line passing through the CG of the BalanceBot, and g is the acceleration due to gravity - $9.81m/s^2$.

C. BalanceBot Control

The three states of the BalanceBot that needed to be actively controlled using motor torque inputs, are its body angle θ, wheel angle ϕ (which determines the position in space) and its heading angle, ψ. The heading angle of the BalanceBot is measured with respect to the initial heading of the BalanceBot when the onboard software is first initialized, where a positive heading

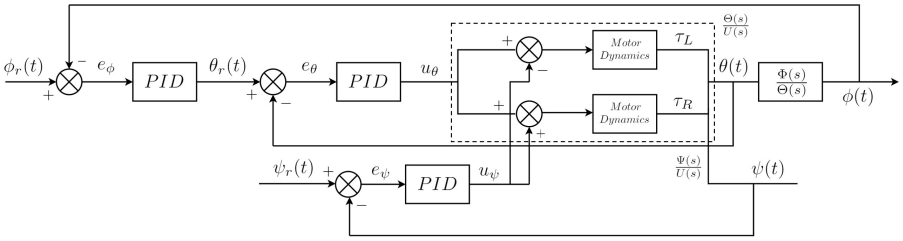


Fig. 2: Full System Block Diagram

follows the right hand thumb rule convention with the Z-axis of the BalanceBot pointing vertically upward. The θ and ϕ angles are measured using a frame that sits on the BalanceBot and moves with it in space (but does not rotate).

The simple, linear and widely used Proportional, Integral, Derivative (PID) controller is used to control all the states of the BalanceBot where the controller acts upon the error or the difference between the reference and actual state of the robot. The output of the PID control block is evaluated as follows, where e represents the error fed into the controller,

$$d(t) = K_p e + K_i \int_0^t e\, dt + K_d \frac{de}{dt} \tag{16}$$

and in the Laplace Domain, the transfer function of the PID controller is given as,

$$D(s) = K_p + \frac{K_i}{s} + K_d s \tag{17}$$

where $K_p, K_i \,\&\, K_d$ are the proportional, integral and derivative gains respectively. It must be noted that a PID controller introduces two additional zeros and one additional pole in the overall transfer function of the closed loop system. An appropriately tuned PID controller can place all the poles of the closed loop system in the left half of the complex plane accordingly in order to achieve desired stable system response characteristics.

For the control of our BalanceBot, a successive loop closure feedback mechanism is required as the equations of motion for θ and ϕ are coupled with torque as the input. We first identify a desired position and a corresponding desired wheel angle ϕ and a PID controller acting upon the error in the wheel angle outputs a reference body angle, θ_{ref} to be tracked. Subsequently, another PID controller acts upon the error w.r.t. this reference θ and outputs a PWM duty cycle u_θ input to the motors that will provide the necessary reaction torques to track θ_{ref}.

However, as mentioned, we also need to control the heading of the BalanceBot ψ. Given a reference heading, ψ_{ref}, another PID controller shall be used to output a PWM duty cycle, u_ψ based on the error between the reference and the current heading obtained from state-estimation. A simple differential motor drive is used to track heading, where a positive change in heading implies a left turn according to the BalanceBot frame. Hence, the duty cycles output from the body angle controller, u_θ and the heading controller, u_ψ are then applied to the left and right motors as follows to enable the BalanceBot to track both the body angle and heading simultaneously (u_L and u_R refers to the total commanded duty cycle of the left and right motors respectively).

$$\begin{aligned} u_L &= u_\theta - u_\psi \\ u_R &= u_\theta + u_\psi \end{aligned} \tag{18}$$

Fig. 2 depicts the overall architecture of the lower level control system implemented to allow the BalanceBot to track a certain position and heading.

D. State Estimation

In order to apply the described control strategies, the system needs accurate state estimation, including body angle, wheel angle, and pose of robot. We utilize the Tait-Bryan measurement from the 9 DOF inertia measurement unit (IMU), MPU9250, as the estimation of body angle. For the wheels angle, a magnetic quadrature encoder is mounted on the shaft of each motor to count wheel rotation ticks and to determine the rotation direction. By applying a low-pass filter with a cut-off frequency of 10Hz on both the wheel encoder readings, we reduced the noise in the data for further odometry calculations to obtain pose of the robot.

The travel distance of the wheels can be related to the encoder value under the assumption of the no-slip condition. The relationship is shown as Eq. (19), where $\Delta s_{R/L}$ are the travel distance of right and left wheel, $N_{Encoder}$ is the resolution of encoder, and N_{Gear} is the gear ratio of motors.

$$\Delta s_{R/L} = \frac{2\pi * (\Delta Encoder_{R/L}) * D_{R/L}}{N_{Encoder} * N_{Gear}} \tag{19}$$

Knowing the angle of wheel rotation, the diameter of wheels, and the wheelbase length; we could generate

the 2D pose (x, y, θ) of the robot by odometry. We approximate traveled path Δs as a straight line Δd in each odometry update, while assuming the angular velocity of each wheel is constant during the period. By Eq. (20), where b is wheelbase length, we could estimate the pose of robot after each odometry update.

$$\begin{aligned} \Delta\theta &= \frac{\Delta s_R - \Delta s_L}{b} \\ \Delta d &= \frac{\Delta s_R - \Delta s_L}{2} \\ \Delta x &= \Delta d \cos(\theta + \Delta/2) \\ \Delta y &= \Delta d \sin(\theta + \Delta/2) \end{aligned} \tag{20}$$

To compensate the odometry error, such as wheel slippage, we implement Gyrodometry [2] that combines data from gyroscope and odometry. While odometry suffers from unbounded accumulating error, gyroscope has bias causing the orientation drift. Gyrodometry use odometry data for most of the time and only takes gyroscope data when the difference between the two is greater than a predefined threshold. Eq. (21) shows the pseudo-code of gyrodometry implementation, where dt is the update frequency of the Gyrodometry, $\Delta\dot{\theta}_{odo}$ is the rate of angle change in odometry, $\Delta\dot{\theta}_{gyro}$ is the measurement of gyroscope, $|\Delta_{G-O}|$ is difference between the two measurements, and $\Delta\theta_{th}$ is the designated threshold. The threshold we applied is $0.125/dt$.

$$\Delta\theta = \begin{cases} \Delta\dot{\theta}_{odo} dt & \text{, if } |\Delta_{G-O}| \leq \Delta\theta_{th} \\ \Delta\dot{\theta}_{gyro} dt & \text{, if } |\Delta_{G-O}| > \Delta\theta_{th} \end{cases} \tag{21}$$

E. Odometry Calibration

Errors in odometry could be due to uncertainties in measurement of system parameters, such as diameter of wheels, length of wheelbase, and resolution of the encoders; and could also caused by non-systematic errors, such as uneven floor or unexpected object on the terrain. While non-systematic errors occur when on rough floors, systematic errors would accumulate constantly.

To deal with the most significant systematic errors, unequal wheel diameter and uncertainty in length of wheelbase, we first match the real-world translation distance of the robot to the distance calculated from encoders by scaling the nominal wheel diameter. By Eq. (22), we could calculate the nominal wheel diameter D_{nomial}, where $\Delta Encoder$ is the average of right and left encoder reading.

$$D_{nomial} = \frac{11 * N_{Encoder} * N_{Gear}}{2\pi * \Delta Encoder} \tag{22}$$

Second, we perform experiments similar to UMBmark [3] to calibrate the measurement error in wheelbase and unequal wheel diameter. In our experiment, we move the robot by hand along a square in both clockwise and counterclockwise directions to decompose the two different types of error. The respective errors could be computed by Eq. (23), where x_{CCW} and x_{CW} are the x position of odometry in counterclockwise and clockwise direction at the end, $alpha$ is the error caused by non-match wheelbase length, E_b is the correction term of wheelbase, $beta$ is the error caused by unequal wheel diameter, and E_d is the correction term of diameter. We are using the x position of odometry because the magnitude of y position would be the same as x position in theory, if only unequal wheel diameters and inaccurate wheelbase length contribute to systematic errors.

$$\begin{aligned} \alpha &= \frac{x_{CCW} + x_{CW}}{4L} \\ b_{actual} &= \frac{\pi/2 + \alpha}{\pi/2} * b_{nominal} \\ \beta &= \frac{x_{CCW} - x_{CW}}{4L} \\ R &= \frac{L/2}{\sin(\beta/2)} \\ E_d &= \frac{D_r}{D_l} = \frac{R + b/2}{R - b/2} \\ D_L &= \frac{2}{E_d + 1} D_a \\ D_R &= \frac{2}{(1/E_d) + 1} D_a \end{aligned} \tag{23}$$

F. Trajectory Following

1) Set-point March for Competition Tasks: Since the competition tasks did not demand a complex path planning algorithm/strategy like a state-feedback based control that is explained in the Appendix, the wheel angle & heading setpoints of the trajectory to be followed were constantly updated in the `setpoint_control` loop for both the drag race and the 4×4 square tasks. Based on the task to be executed (which was read from the configuration file), the setpoint control loop would set specified forward and turn velocities of the BalanceBot, based on the current state of the BalanceBot estimated from odometry, once the BalanceBot was switched from the manual to autonomous mode using a switch channel input from the transmitter. Furthermore, separate rate sensitivity factors were specified for both the forward and turn velocities in the configuration file, and these were adjusted based on the tasks that the BalanceBot was to perform in manual/autonomous modes.

For the drag race, the total distance from the start to the finish line was known, $\approx 11m$. So upon switching from manual to autonomous mode for the drag race, a forward velocity setpoint was activated and could be adjusted using the specified rate sensitivity factor. The actual wheel angle setpoint was then updated in each loop according to, $\phi_{ref_k} = \phi_{ref_{k-1}} + \frac{V_{fwd} dt}{R_w}$, where V_{fwd} was set to be non-zero based on its corresponding rate sensitivity factor until a total distance of 11.2m was traversed by the BalanceBot. An additional 0.2m distance was included to account for any deviations from a straight line path due to odometry errors, and to ensure that the BalanceBot crosses the finish line before coming to a stop.

For the 4×4 square task, we adopted a strategy similar to the drag race. By applying a constant forward velocity setpoint, the robot was set to move forward continuously at a desired speed. We then added a turning speed and commanded a heading change of 90° at certain travel distances, which corresponded to the corners of a 1m square. The xy position from odometry was not used as our control setpoint since the odometry couldn't be calibrated well enough to be deployed during the competition.

III. Results

A. BalanceBot System Identification

1) Motor Parameters: Using the method outlined in section II-B1 and the provided executable `test_motor_params`, the motor parameters obtained are as listed in Table I.

TABLE I: Motor parameters

Parameter	Left motor	Right motor
Motor coil resistance, R (ohm)	5.9	7.2
No load speed, ω_{nl} (rad/s)	42.0278	36.5390
Stall torque, τ_{stall} $(N \cdot m)$	2.0339	1.6667
Motor constant, K $(N \cdot m$ / $\sqrt{W})$	0.0134	0.0148
Coefficient of friction, b $(N \cdot m \cdot s)$	0.0005	0.0011
Shaft inertia, I_{gb} $(Kg \cdot m^2)$	2.8665e-05	5.6287e-05

2) Mass and Moments of Inertia: The mass of the Balancebot was evaluated in two parts - the mass of the body along with all the electronics and motors mounted on it and the mass of the two wheels (including motor shaft mount). The mass of the Balancebot body without the wheels and the shaft attachments weighed $1093g$, while the average weight of the two wheels along with the motor shaft mount was about $103.5g$. The bifilar pendulum experiment was conducted without the two wheels and the motor shaft mounts, as the moments of inertia of the wheels will be estimated using its mass and diameter, assuming that it is a thin disk. Hence, the mass m used in Eq. (15) was $1093g$. Table Eq. (II) lists each axis's time-period and their corresponding moments of inertia estimated.

TABLE II: Estimated Moments of Inertia using Bifilar Pendulum Method

	Time Period (s)					J_{ii}
Axis, i	*1*	*2*	*3*	*Avg*	*St.Dev*	(kg/m^2)
X	3.80	3.70	3.88	3.76	0.10	0.00803
Y	4.43	3.90	4.42	4.12	0.35	0.01059
Z	2.68	2.70	2.79	2.72	0.06	0.00423

TABLE III: BalanceBot Parameters for Model

Mass of body, m_b	1093 g
Average mass of wheel, m_w (incl. motor shaft mount)	103.5 g
Average radius of wheel, R_w	0.0415 m
Distance of center of mass from wheel center, L	0.096 m
Moment of inertia of body about tilt axis, J_{xx}	0.00803 kg/m^2
Gearbox Ratio, G	20.4
Nominal Voltage to Motors, V_n	12.0 V
Average Motor Stall Torque @ V_n	1.8503 N.m
Average Motor No-Load Speed, ω_{nl}	39.2834 rad/s
Average Motor Shaft Inertia, I_{gb}	$4.2476 \times 10^{-5} kg/m^2$
Average inertia of wheels, $I_{gb} + mR_w^2$	$1.316 \times 10^{-4} kg/m^2$

B. BalanceBot Control

Following system identification and measurements of the various parameters of the BalanceBot, the dynamic equations outlined in section II-A can be converted into numerical form. The parameters used to convert the equations into numerical form are as listed in Table III.

Using the above parameters, the transfer function $G_1(s)$ from the PWM duty cycle of the motors to the body angle, θ of the inner loop is evaluated from Eq. (4) as below,

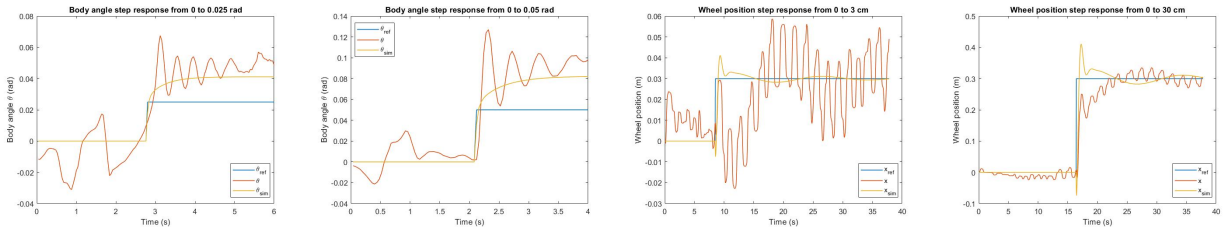
$$G_1(s) = \frac{\Theta(s)}{U(s)} = \frac{-1070s}{s^3 + 118.8s^2 - 103.5s - 4196} \quad (24)$$

The poles and zeros of this transfer function $G_1(s)$ are -119.3514, 6.228, -5.6501 and 0, respectively. Having a pole in the right hand complex plane indicates that the system is naturally unstable. Similarly, the transfer function from the body angle θ to the wheel angle ϕ is calculated using the parameters from Eq. (5) as below,

$$G_2(s) = \frac{\Phi(s)}{\Theta(s)} = \frac{-3.363s^2 + 154.1}{s^2} \quad (25)$$

The poles and zeros of the transfer function $G_2(s)$ are 0, 0 and -6.7702, 6.7702, respectively. Since both the poles of the system are at 0, it is only marginally stable and is prone to oscillations.

Next, the PID controllers, as discussed in section II-C, are implemented in software using `rc_filter` functions in the RC Library. More specifically, three separate PID filters for θ, ϕ and ψ are initialized using the `rc_filter_pid` function available in the RC Library. Although in section II-C, the PID controllers were discussed in continuous-time domain, these can be only be implemented in a discrete form in software. The `rc_filter` functions act upon filters whose transfer functions are specified in the discrete Z domain. The



(a) (b) (c) (d)

Fig. 3: Response of the BalanceBot to a step error of (a) 0.025 rad and (b) 0.05 rad in θ, and a step error of (c) 3 cm and (d) 30 cm in position (or ϕ). Orange lines represent actual system response while yellow lines show response from MATLAB simulations; blue lines are the reference values. Gains used are indicated in Table IV

transfer function of a PID controller (/filter) in discrete-time [4] [5] is as given below,

$$D(z) = K_p + K_i \frac{dt}{z-1} + \frac{K_d}{T_f + \frac{dt}{z-1}} \tag{26}$$

where dt is the discrete sampling time and T_f is the first-order derivative filter time-constant. The real time controller in software is called through an IMU interrupt, triggered periodically at a fixed frequency of $100Hz$, and hence $dt = 0.01s$. Further, a first-order derivative filter is required as a pure differentiator cannot be directly implemented in discrete-time. Hence, a non-zero T_f is needed for all discrete PID filter instances and typically $T_f > dt/2$ in order to ensure that the derivative filter results in a stable pole of the closed loop system [4]. Furthermore, having a higher time constant, T_f allows the use of a larger derivative gain, K_d as the high frequency noise due to larger derivative action is attenuated more.

With this knowledge of how PID filters are implemented in discrete-time, three PID controllers were tuned and the final gains used for each of the controllers are as listed in Table IV. The process of tuning the PID controllers was rather the most challenging part as many permutations for the three sets of PID gains exist. The numerical form of the BalanceBot dynamics was simulated in MATLAB to identify a set of gains that provided settling times of $\approx 1s$ with minimal overshoot and quick damping of any oscillations. The initial set of gains used for simulations were obtained from the root locus analysis MATLAB file provided and a gain for a lead-lag controller was picked to ensure stability, and the lead-lag controller was subsequently approximated as a PID controller using [6]. These set of gains were then simulated using the MATLAB PID file and tuned to achieve a desired response. Despite these efforts, the simulated responses were very different from the actual system response as seen in Figure 3, possibly due to the highly non-linear nature of the system and also a discrete implementation of the PID controllers that were simulated in continuous-time.

TABLE IV: PID Controller Parameters

Controller	PID Controller Gains			
	K_p	K_i	K_d	T_f
Body Angle, θ	3.0	10.0	0.015	3.141
Position, ϕ	0.002	0.0025	0.015	0.3141
Heading, ψ	1.0	0.0	0.1	0.3141

Hence, further fine tuning of the gains was required on the BalanceBot. The gains for each controller were tuned in a systematic manner starting with the inner loop controller. Knowledge about how varying PID gains affect a system was primarily used while tuning the gains. It was noted that the integral and derivative gains could not be as large as those obtained in simulations due to practical issues such as integral wind-up and oscillations due to amplification of high-frequency noise. Furthermore, for the position controller, it was observed that there was considerable oscillations about the setpoint and in many occassions the BalanceBot would overshoot the position setpoint and take a long time to get back. As such, an additional saturation on the integral term of the position controller was implemented such that the integral component was saturated as $\theta_{ref_I} \in [-0.01, 0.01] rad$. This allowed the BalanceBot to stay closer to the setpoint, but did not fully eliminate oscillations around the setpoint.

The transfer functions of the PID controllers for the inner & outer loop and heading controllers, in both continuous-time and discrete-time are as given in Table V. In software, once the discrete-time filters for each PID controller is initialized, they are then marched forward with the respective errors at that instant using the `rc_filter_march` function, which then provides the respective outputs of each of the PID filters at that instant. Using the outputs, u_L and u_R are set during each IMU interrupt as outlined in section II-C.

The actual system response for step errors in body

TABLE V: PID Controller Transfer Functions

Controller	PID Controller Transfer Functions	
	Continuous-time (s)	Discrete-time (z)
θ (D_1)	$\frac{0.015s^2+3s+10}{s}$	$\frac{3.0048(z-0.9968)(z-0.9667)}{(z-1)(z-0.9968)}$
ϕ (D_2)	$\frac{0.015s^2+0.002s+0.0025}{s}$	$\frac{0.049755(z^2-1.998z+0.9982)}{(z-1)(z-0.9682)}$
ψ (D_3)	$0.1s+1$	$\frac{1.3184(z-0.9759)}{(z-0.9682)}$

angle, position and heading controllers are given in Figure 3 and 4 respectively, along with the response obtained from simulations on MATLAB. It must be noted that when data for step response of the inner loop was recorded on the actual system, the outer loop position controller was turned off. From Fig. 3, there are sustained oscillations while tracking both body angle and position, though the body angle oscillations were not very noticeable on the robot. However from the simulated responses, it appears that the inner loop is critically damped, but the characteristics exhibited by the actual system varies significantly. Furthermore, the steady state settling value of θ matches that of the simulated responses shown in Fig. 3 (a) and (b), but it does not track the set change in θ_{ref} as the effect of the integrator in the PID control is lost, so to speak, because the additional pole at 0 introduced by the integral component of the PID controller is cancelled by the zero at 0 of the transfer function $G_1(s)$ as in Eq (24). Nonetheless, the steady state of the inner loop was sufficiently close to the setpoint to allow the BalanceBot to remain stable near the point of linearization.

With saturation applied to the integral term of the position controller, the BalanceBot was able to stay closer to the position setpoint but there were consistent oscillations with amplitude of about $\pm 3cm$ around the setpoint that could not be eliminated. This could also be attributed to the fact that the inner loop always resulted in a consistent offset (or overshoot) from the actual θ_{ref}. Nonetheless, the BalanceBot exhibited very stable movement when the position setpoints were changed in a continuous manner during the manual driving and autonomous segments of the competition and hence the gains used for the controllers were deemed to be acceptable with regards to the task objectives for the BalanceBot.

From Fig 4, it can be seen that the heading controller works the best amongst the three controllers because its dynamics is not coupled with that of θ or ϕ, and as a result the step response plot shows that a settling time of less than $1s$ is achieved with minimal oscillations and acceptable overshoot. The chosen set of gains for the heading controller allowed a good performance during the manual driving segment of the competition, which was reflected in the smooth turning profiles, with almost no jerks, exhibited by the BalanceBot.

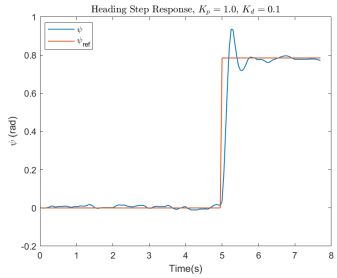


Fig. 4: Response of the BalanceBot to a step change of 45^o in heading angle, ψ. Gains used $K_p = 1.0$, $K_d = 0.1$

C. Odometry

In order to calibrate odometry of the robot, we conducted three sets of experiments, moving the robot in a straight line, and in a square clockwise and counterclockwise. Each set of experiments had three trials. Through these experiments, we could improve the accuracy of the odometry by adjusting the diameter of right and left wheels and the length of wheelbase.

First, we moved the robot in a 11-meter straight line and collect the data of encoder readings. By equation 22, we obtained the nominal wheel diameter of our robot as 0.08385m. Second, the robot was moved in a square of length 2.42 meter in both clockwise and counterclockwise directions. The average distance of odometry error is 0.28m for counterclockwise and 0.44m for clockwise before calibration. By equation 23, we computed the nominal wheelbase as 0.2118 m, diameter of right and left wheels as 0.0839 m and 0.0838 m respectively.

We used the same experiment log with the new parameters to verify the calibration result. Figures 5A and 5B show the comparison result of odometry before and after calibration, where the odometry after calibration is closer to the actual square. The scatter plot of ending position (figure 5C) clearly indicate that the odometry error is significantly reduced. The errors for both counterclockwise and clockwise is 0.06m after calibration, which is equal to an improvement of 78.6% and 86.4% respectively.

The residual error after calibration may be due to systematic errors, such as misalignment of wheels, and other non-systematic errors in the experiment. Further calibration could be perform to reduce the systematic

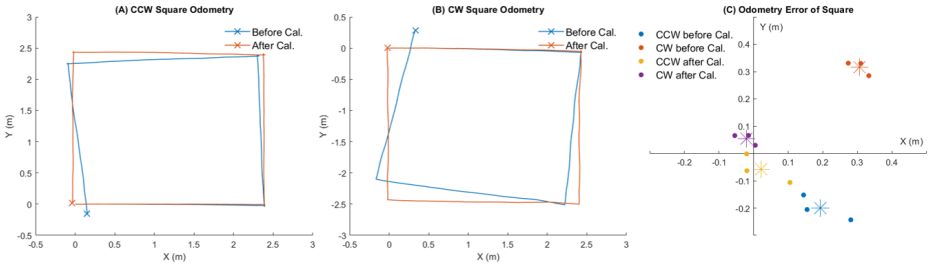


Fig. 5: Odometry before and after calibration. Figure A and B show the odometry of a real square path in counterclockwise and clockwise direction. The stars are the ending position of each experiment. Figure C shows the scatter plot of ending position error of each experiment trial before and after calibration. The stars are the mean position error of each experiment.

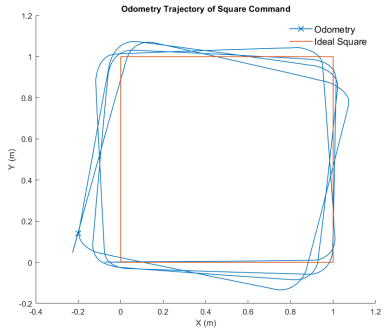


Fig. 6: Odometry of 4 laps of square path, where the star markers shows the ending position. The overshoot at the end happened due to the oscillating position control.

error. However, the magnitude of odometry error is now very close to the oscillation magnitude of the position controller of our robot, making it difficult and unnecessary to calibrate further before having a better tuned controller.

D. Competition Performance

The final competition consisted of 5 events. For event 2 the robot was able to complete a square trajectory of width 1 m, 4 times while balancing. The robot didn't cross the markings on the floor for 2 out of the 4 laps. One of the reasons why the robot wasn't able to avoid crossing the line for the 4 laps was that the odometry calibration wasn't accurate. Figure 6 shows the path of 4 laps of square. The drift occurs because the robot was commanded to follow total travel distance and turning at the certain distance, which is not feedback-based trajectory control and error accumulates as the robot tried to balance at the same time. The team got 150 points in this event.

In event 3, the robot finished the drag race in 13.63 seconds. This time could be improved with further tuning of control gains and the saturation limits. Events 4 and 5 involved an easy and a hard manual obstacle course race. The team took 43.26 seconds to finish the easy race and 40.68 seconds to finish the hard one. Improvements on the easy race could have been made by driving more aggressively. The scores for each events are recorded in Table VI in the Appendix.

E. Potential Improvements

1) *System Identification*: Errors in measurement of parameters such as mass or moments of inertia due to uncertainties like oscillations along multiple axes during bifilar pendulum experiments, etc. could be alleviated to improve how closely simulations match actual system responses.
2) *Controller Tuning*: Better tuning of the body angle controller is needed to account for the discrepancy between θ_{ref} and actual θ being tracked by the BalanceBot. Oscillations in the body angle had to be further damped as in many situations the amplitude of the oscillations were large enough to cause the BalanceBot to tip over and this was evident during the drag race. This prevented us from employing a body angle based control for the drag race, that could have potentially helped us achieve much better timings. Also, since a large integral gain was used for the body angle controller, adding a saturation to the integral component and/or resetting it once setpoint is reached could have helped reduce oscillations.
3) *Trajectory Following*: Although the trajectories for the competition were relatively simple, major drawbacks for our robot were use of uncalibrated odometry and an open loop path control, which required hard-coded tuning to accomplish the tasks. By applying the calibrated odometry and implementing state-feedback of x and y position, the robot's trajectory following performance could have been improved.

IV. Conclusion

To build a balancing robot the team had to implement functions in C to run the motors, stabilize the dynamics of the system, move the robot with a remote controller and command the robot to follow trajectories autonomously. The final implementation could balance reliably, even when disturbed by external forces or when following a trajectory. In the end, the system was able to complete the required checkoff tasks and placed 4^{th} in the competition.

References

[1] N. M. T. B. Peter Gaskell, Saam Ostovari, "Dynamics and control of a mobile inverted pendulum," 2019.

[2] J. Borenstein and L. Feng, "Gyrodometry: A new method for combining data from gyros and odometry in mobile robots," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 1. IEEE, 1996, pp. 423–428.

[3] ——, "Umbmark: A method for measuring, comparing, and correcting dead-reckoning errors in mobile robots," 1994.

[4] "Mathworks® MATLAB documentation: pid." [Online]. Available: https://www.mathworks.com/help/control/ref/pid.html

[5] J. Strawson, "Robot control library documentation." [Online]. Available: http://strawsondesign.com/docs/librobotcontrol/index.html

[6] E. Henriksson, "Interpretation of lead-lag controllers and their connection to pid controllers," 2012. [Online]. Available: https://www.kth.se/social/upload/5236f795f276542ea736d9a8/Understanding%20leadlag.pdf

[7] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Wiley, 2005. [Online]. Available: https://books.google.com/books?id=wGapQAAACAAJ

V. Appendix

A. Derivations of Equations of Motion

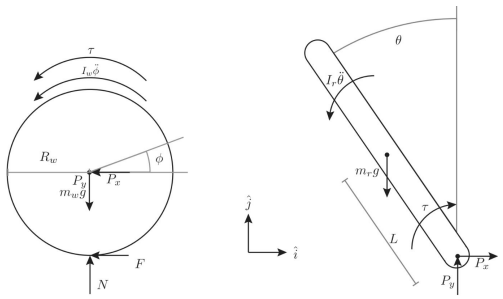


Fig. 7: Free-body diagram used for deriving the dynamic model equations for the BalanceBot

The coefficients in Eq. 4 and 5 are as listed below -

$$\begin{aligned} a_1 &= I_w + (m_w + m_b) R_w^2 \\ a_2 &= m_b R_w L \\ a_3 &= I_b + m_b L^2 \\ a_4 &= m_b g L \\ b_1 &= 2\tau_s \\ b_2 &= 2\tau_s/\omega_{NL} \end{aligned} \tag{27}$$

B. Gain Tuning

Fig. 8 shows an example of how the gain tuning was done for the PID controllers. Starting with a high proportional gain of 5.0, oscillations were observed and hence the P gain was reduced to 1.0 and to improve the settling time, a D gain was added. When both gains were increased, the response exhibited slightly weird behaviours with the D gain fighting the step change and hence the a P and D gain of 1.0 and 0.1 was finally used.

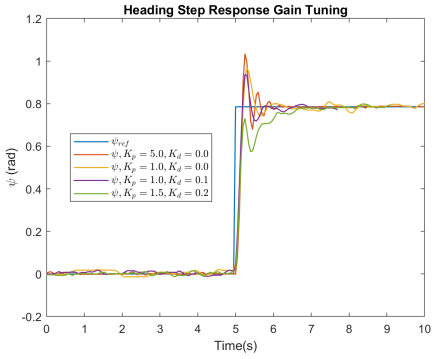


Fig. 8: Response of the BalanceBot to a step change of 45^o in heading, ψ for various PD gains

C. State Feedback Control

This section provides information about the state-feedback control method for trajectory following that was initially attempted. However, this was not eventually used due to the additional complexity involved and time taken to tune the parameters for this method to work effectively.

In order to command the robot to follow a trajectory two methods were implemented and only one of them was used in the final competition. The first method implemented involved state feedback control. For the competition, the team didn't use state feedback control to follow a trajectory because its complexity and several other pitfalls which will be explained in this section. The idea of this method is that the velocity and angular velocity of the robot will be updated according to the 3 error variables ρ, α and β shown in Figure 3.

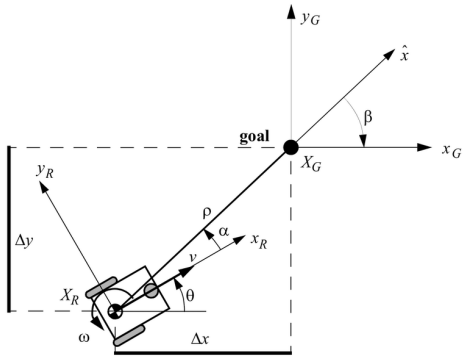


Fig. 9: Polar coordinate representation. ρ represents the distance to the goal, α is the angle error between the direction of the current velocity and the line crossing the current position to the goal and β is the pose error.

The 3 error variables are calculated as follows

$$\rho = \sqrt{\Delta x^2 + \Delta y^2} \tag{28}$$

$$\alpha = -\theta + atan2(\Delta y, \Delta x) \tag{29}$$

$$\beta = -\theta - \alpha \tag{30}$$

The control law

$$v = K_\rho \rho \tag{31}$$

$$w = K_\alpha \alpha + K_\beta \beta \tag{32}$$

$$\beta = -\theta - \alpha \tag{33}$$

The dynamics for the error are then

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -K_\rho \rho cos(\alpha) \\ K_\rho sin(\alpha) - K_\alpha \alpha - K_\beta \beta \\ -K_\rho sin(\alpha) \end{bmatrix} \tag{34}$$

Where the gains K_ρ, K_α and K_β are the constants that decide how important each error is. This control law will drive the robot to zero error. The system can be proven to be stable if

$$K_\rho > 0; K_\beta < 0; K_\alpha - K_\rho > 0 \tag{35}$$

If the goal is behind the robot, it can move backwards by changing the sign of the velocity in the implementation. The results from the simulation in Matlab can be shown in Figures 11, 12 and 13. One of the main pitfalls from this method is that it is hard to find a set of gains K_α, K_β and K_ρ that can make a square trajectory, take a reasonably amount of time and have reasonable velocities.

The plots from the MATLAB simulation of the above trajectory following algorithm is as shown below.

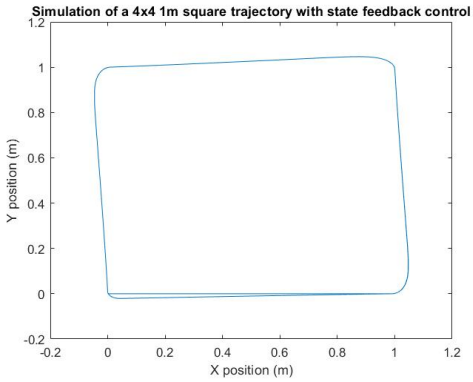


Fig. 10: Plot of the matlab simulation of the trajectory planning for a 4x4 square trajectory using state feedback.

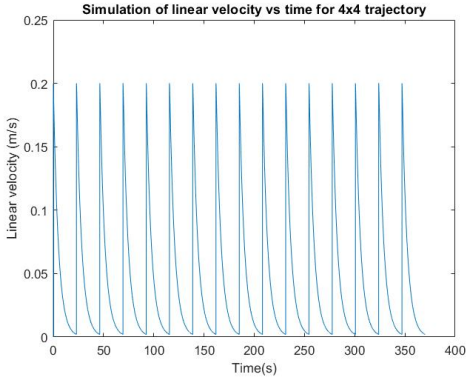


Fig. 11: Linear velocity profile of the robot for the 4x4 square trajectory shown in Figure 11.

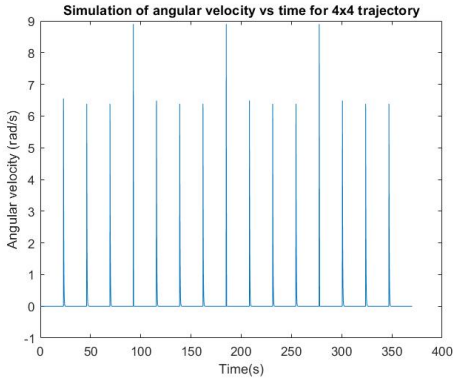


Fig. 12: Angular velocity profile of the robot for the 4x4 square trajectory shown in Figure 11.

D. Step Response Plots

In this section, enlarged plots of the step responses shown in Fig. 3 is included for reference for interested readers. Please refer to Fig. 13.

E. Competition Results

The points obtained by the team for each task in the competition is listed in the Table below.

TABLE VI: Competition results

	Event 1	Event 2	Event 3	Event 4	Event 5	Total
Points	200	150	120	150	175	795

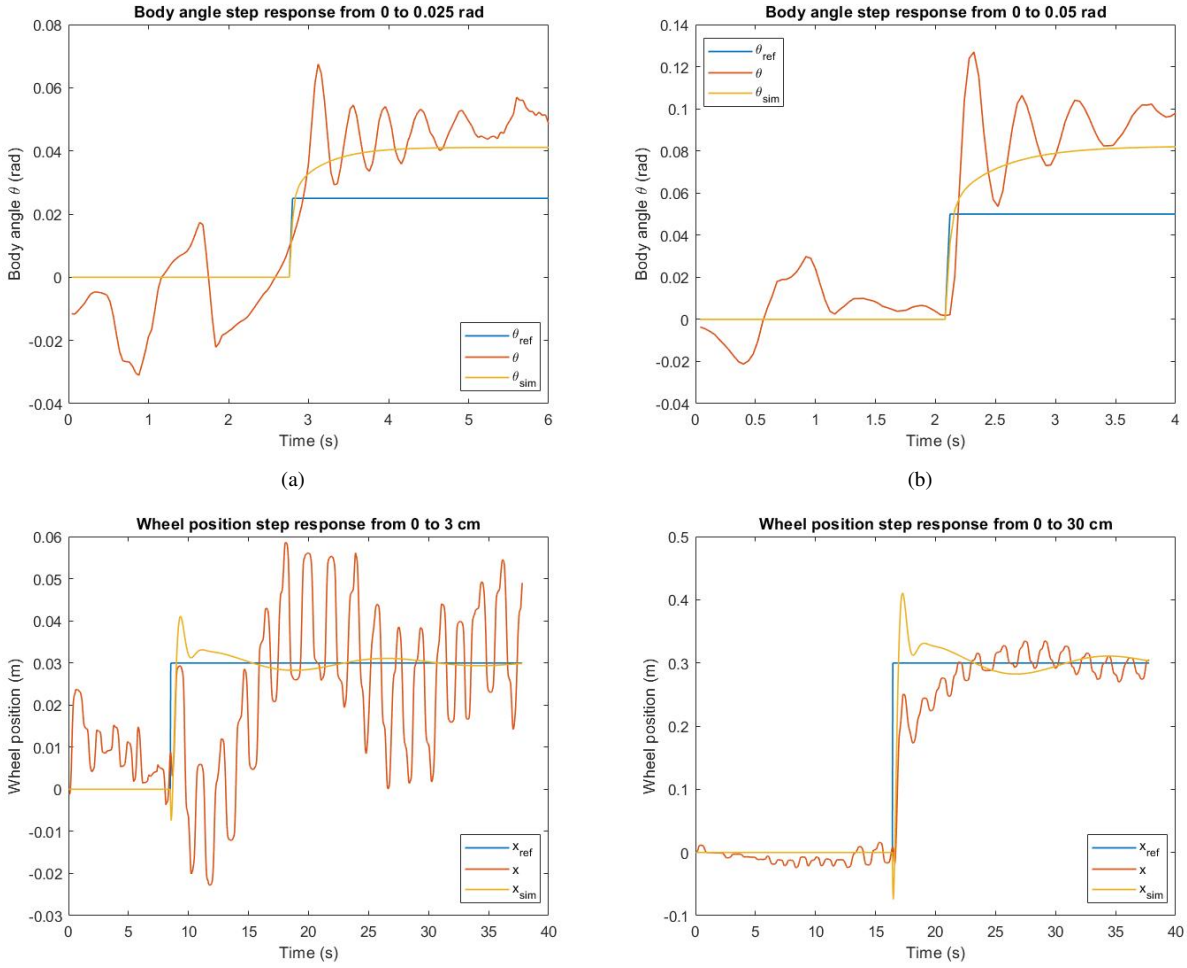


Fig. 13: Response of the BalanceBot to a step error of (a) 0.025 rad and (b) 0.05 rad in θ, and a step error of (c) 3 cm and (d) 30 cm in position (or ϕ). Orange lines represent actual system response while yellow lines show response from MATLAB simulations; blue lines are the reference values. Gains used are indicated in Table IV