ROB 550 BalanceBot: Team 5 Report

Shreyas Bhat, Saptadeep Debnath
Email: (shreyasb, saptadeb)@umich.edu

***Abstract*—This report describes our work on balancing a two wheeled Mobile Inverted Pendulum (MIP) and controlling it manually via an RC controller and also taking it through two pre-determined trajectories autonomously. A two-loop PID controller was set up to stabilize the unstable equilibrium position of the robot. Another PID controller was used to control the heading of the robot. A manual DSM remote was also set up to make the robot go through any complex path while maintaining balance.**

***Index Terms*—Mobile Inverted Pendulum, PID control, beaglebone, robot control library**

I. Introduction

THE goal of this lab was to control a Mobile-Wheeled Inverted Pendulum (MWIP) to make it go through a set of predetermined trajectories and also to implement manual control of the robot in order to make it go through an obstacle course. For balancing the robot, we implemented a cascaded PID controller, to control the body angle and the wheel angle (position) of the robot. Another independent PID controller was tuned to control the heading of the robot.

We used a combination of odometry and dead reckoning to get feedback about the position and heading of the robot. Using the UMBmark procedure, we corrected the systematic errors in the odometry measurements. These measurements were then used as feedback to two high level controllers, to make the bot go through arbitrary distance and to maintain a given heading.

The organization of the report is as follows: Section II describes the dynamic models of the robot and motors. Section III talks about the PID controllers implemented for the balancing and steering. Section IV elaborates on the methodology used for odometry estimation and systematic error correction by UMBmark. Section V describes the high level controllers and also on the methodology used to plan paths for the two tasks: drag race and square track. Section VI talks about our performance in the competition. Lastly, Section VII gives a brief overview of the results.

II. System Dynamics

A. Mechanical Model of the system

The two wheeled balancebot is modelled as a mobile-wheeled inverted-pendulum system (MWIP). Figure 1 represents the freebody diagram of the BalanceBot, where the mass of the wheels and body are represented by m_w and m_b respectively, whereas the inertias are represented as I_w and I_b. The body of the BalanceBot of length L rotates about the centre of mass of the wheels. The angle of rotation of the body about the COM of wheel is given by θ which represents the body angle of the system, whereas the angle by which the wheels rotate about the COM is given by ϕ which represents the wheel angle.

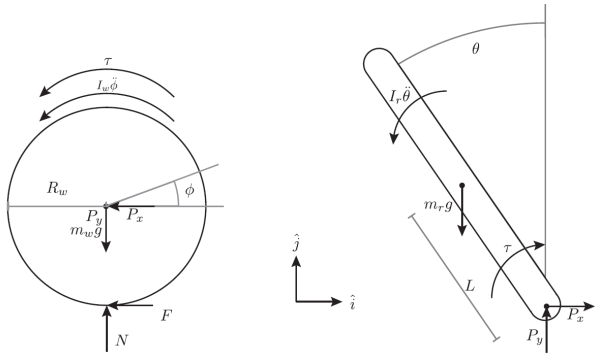

Fig. 1: Free body diagram for MWIP

With the assumption that the wheels of the BalanceBot don't slip, i.e. $\ddot{x} = -R_w\ddot{\phi}$, equations of motion for the BalanceBot can be given by equations in 1.

$$\begin{aligned}&(m_b R_w L cos\theta)\ddot{\phi} + (I_b + m_b L^2)\ddot{\theta} \\ &\quad = m_b g L sin\theta - \tau \\ &(I_w + (m_b + m_w)R_w^2)\ddot{\phi} + (m_b R_w L cos\theta)\ddot{\theta} \\ &\quad = m_b R_w L\dot{\theta}^2 sin\theta - \tau\end{aligned} \tag{1}$$

$$J_{axis} = \frac{m_0 g d^2}{16\pi^2 L} T_0^2 \tag{2}$$

To measure the moment of inertia of the BalanceBot across the three axes, a bifilar pendulum procedure was implemented as is shown in Figure 2. After determining the period of oscillations (T_0) for a small disturbance in the three axes, we use equation 2 to compute the moment of inertia in the three axes, where m_0 denotes the mass of the bot without the wheels, d is the distance between the bifilar setup and L is the length of the setup.

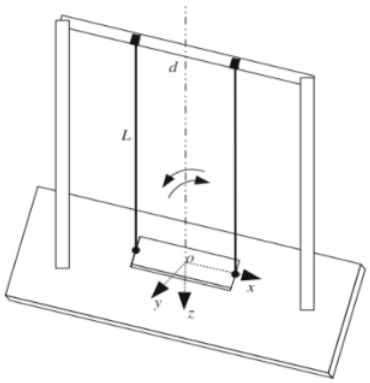

Fig. 2: Bifilar Pendulum Setup

TABLE I: Mass and Moment of Inertia.

Variables	Values
m_b w/o wheels (kg)	1.0886
J_{xx} $(kg \cdot m^2)$	0.0070
J_{yy} $(kg \cdot m^2)$	0.0087
J_{zz} $(kg \cdot m^2)$	0.0043

B. Motor Model

The motors used for this project were Pololu 12V Brushed DC Motors with a gear ratio of $20.4 : 1$. To measure the motor parameters, we started out with measuring the motor resistance (R) with a multimeter. we then use the $measure\_motors$ script to evaluate the no-load speed (ω_{NL}), motor constant (K), stall torque (τ_s), friction coefficients and inertia of the motor armature, shaft and the gearbox (I_M). The measured values are reported in the Table II.

III. Controller Design

The open loop dynamics of the BalanceBot is unstable and requires a control law to control the three Euler angles (roll, pitch and yaw). The nomenclature used for this paper for controlling the body angle is inner loop and for the wheel angle is outer loop which if further

TABLE II: Motor Parameters

Parameters	Left	Right
R (Ω)	6.0	6.0
ω_{NL} (rad/s)	49.846	50.540
I_{NL} (A)	0.0905	0.1011
K (Nm/A)	0.2274	0.2231
τ_s (Nm)	0.4503	0.4417
b $(Nm \cdot s)$	0.0004	0.0004
I_M $(kg \cdot m^2)$	2.06E-3	2.23E-3

TABLE III: Inner loop control gains.

Parameters	Values
K_p	-3.0
K_i	-14.0
K_d	-0.25
T_f	0.0278

defined in Figure 6. This kind of a nested loop is required since we want to control two parameters (i.e. θ and ϕ) with one control input which is the torque to the motors (τ). The plant dynamics which transforms a given torque to the body angle is given as $G_1(s)$ given by equation 3 and $G_2(s)$ defines the dynamics from given body angle to wheel angle defined by equation 4. In the subsequent subsections we discuss more about the body angle controller, the wheel angle controller and the steering (yaw) controller.

$$G_1 = \frac{-106.4s}{s^3 + 6.458s^2 - 72.97s - 206.3} \tag{3}$$

$$G_2 = \frac{-2.025s^2 + 96.64}{s^2} \tag{4}$$

A. Body angle control (Inner Loop)

The main goal of the inner loop is to minimise the error between in the observed body angle (θ) and the reference body angle (which is hard-coded to be zero). We implement a discrete-type PID controller to control the duty cycle and hence the torque input to the motors. The error in the body angle is given as input to the PID block which generates the torque required to maintain the body angle. In addition to using a PID controller, a roll-off filter (low pass filter) is also used to neglect the high frequency noise which is generated by the derivative gain. Figure 3 shows the response of the inner loop when given a reference of 0.1 radians and 0.2 radians. It is observed that the inner loop is able to follow a given step if the input is a small change in angle (i.e. in case of 0.1 radians), but fails to maintain a large input body angle and tips-over (i.e. in case of $>$0.2 radians). The gains of the controller are given in Table III and the controller equation for $D_1(s)$ is given as,

$$D_1(s) = \frac{-0.3334s^2 - 3.389s - 14}{0.0278s^2 + s}$$

B. Wheel angle control (Outer Loop)

The outer loop controller controls the wheel angle (ϕ) of the BalanceBot. The error of the observed wheel angle and the reference wheel angle is given as an

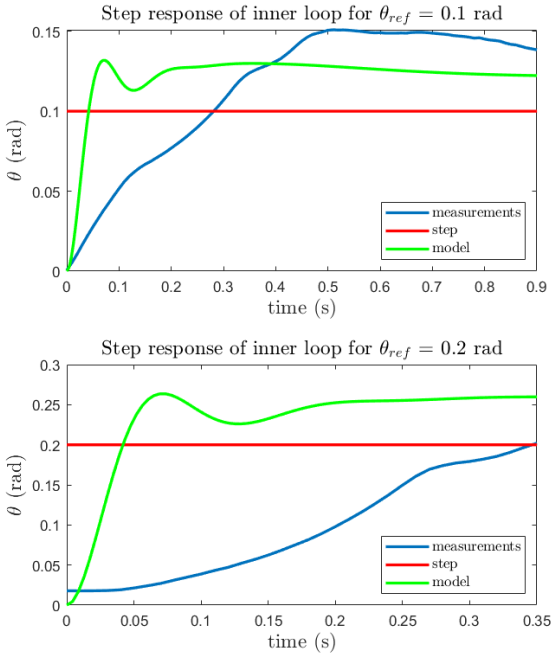

Fig. 3: Step Response of the inner loop to a change of (a) 0.1 radians and, (b) 0.2 radians

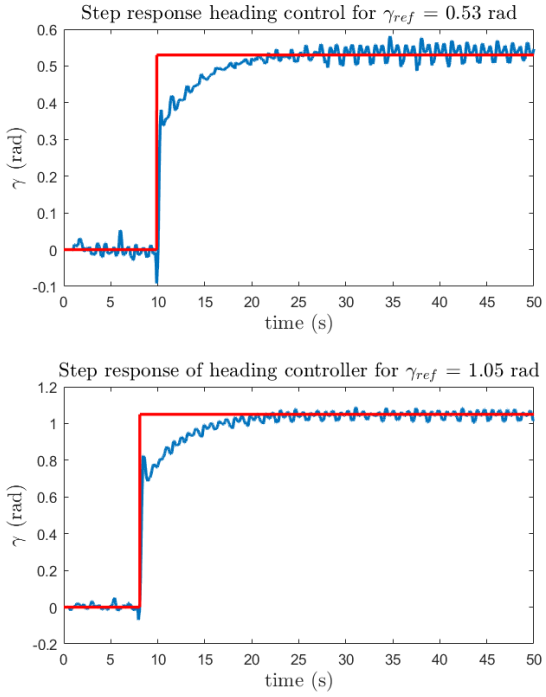

Fig. 5: Step Response of the heading controller to a change of (a) 0.53 rad and, (b) 1.05 rad

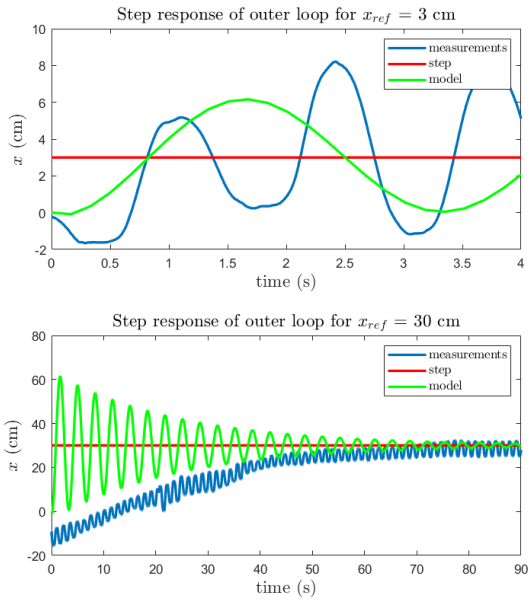

Fig. 4: Step Response of the outer loop to a change of (a) 3 cm and, (b) 30 cm

input to the outer loop controller which produces a corresponding reference body angle (θ). Figure 4 shows the step response for the outer loop controller when given a certain 'x' distance to travel to, which is in turn converted into the reference wheel angle. It is seen that even though the system exhibits high oscillations and has a low rise time, the system is still able to track the command with ≈ 0 steady-state error. This is highly desirable for the system, as the BalanceBot is required to balance on a given spot, maintain its position, and reject disturbance. It is also observed that the model response does not match the real-time data. This is one of the reasons why the gains from the MATLAB model are used as a base point for tuning the controllers and not as the final controller gains. In reality the input to the motor is also saturated to -1 to 1, which is not modelled properly in the MATALB model. The gain of the controller is given in the Table IV and the controller equation for $D_2(s)$ is given as,

$$D_2(s) = \frac{-0.8332s^2 - 5.722s - 14}{0.1944s^2 + s}$$

TABLE IV: Outer loop control gains.

Parameters	Values
K_p	0.02
K_i	0.0007
K_d	0.001
T_f	0.1944

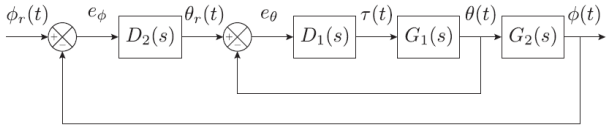

Fig. 6: Representation of the block diagram of the full control system, comprising of the inner loop and the outer loop, controlling the body angle and the wheel angle of the BalanceBot.

C. Steering Control

An additional steering controller is also implemented in the model so as to minimize the change in heading. The yaw angles from the odometry data are used as the input to the controller, which then provides the required torque to the motors. It is found that the heading controller when used with the odometry data provides a better control and less jittery response than when fused IMU data is used. It is because of the low sensitivity of the odometry heading data, which provides an inherent deadband for the controller input. The gain of the controller is given in the Table V and the controller equation for $D_3(s)$ is given as,

$$D_3(s) = \frac{0.09s^2 - 1.012s + 0.3}{0.04s^2 + s}$$

TABLE V: Steering control gains.

Parameters	Values
K_p	1
K_i	0.3
K_d	0.05
T_f	0.04

IV. Odometry

Odometry is used to calculate the location and orientation of the robot in world frame using data from wheel encoders and gyroscope. The equations used for odometry update are as follows:

$$\Delta s_w = R_w \Delta \phi_w$$

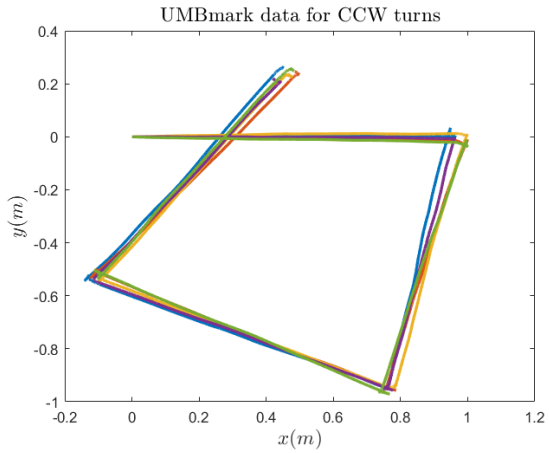

Fig. 7: Data for UMBmark for counter-clockwise turn about the square

$$\Delta d = \frac{\Delta s_L + \Delta s_R}{2}$$

$$\Delta \psi(t_k) = \frac{\Delta s_R - \Delta s_L}{b}$$

$$\Delta x = \Delta d \cos\left(\psi(t_{k-1}) + \Delta\psi/2\right)$$

$$\Delta x = \Delta d \cos\left(\psi(t_{k-1}) + \Delta\psi/2\right)$$

Where w is the wheel index (L or R), R_w is the wheel radius, ϕ_w is the wheel angle of wheel w, ψ is the heading angle of the robot and b is the wheel base.

To calculate the change in the wheel angle for each wheel at every time instant, we store the wheel angle at the last time instant as a variable in the odometry struct. We also store the ψ at the last time instant for calculating $\Delta\psi$.

Now, calculating the current values of x, y and ψ are just a simple addition of their calculated changes to their previous values.

$$\begin{aligned} x(t_k) &= x(t_{k-1}) + \Delta x \\ y(t_k) &= y(t_{k-1}) + \Delta y \\ \psi(t_k) &= \psi(t_{k-1}) + \Delta\psi \end{aligned} \tag{5}$$

A. UMBmark

A technique defined in [1] called *UMBmark* was implemented in order to try and alleviate the systematic errors in odometry. We manually took the bot across the square path of side 1 meter ensuring minimum slip of wheels and recorded the odometry measurements and the yaw angle that we got from the gyro. We took 6 such readings and plotted them in MATLAB (Figure 7 and 8).

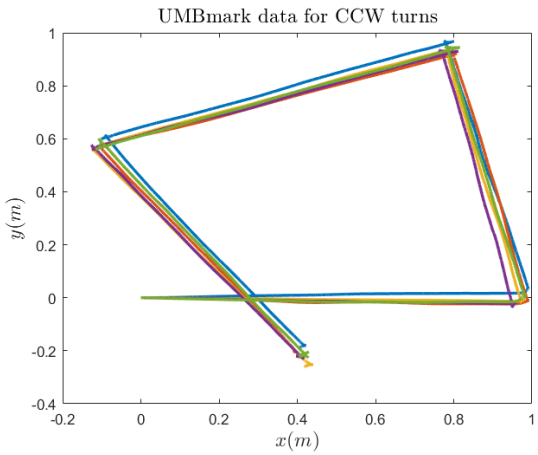

Fig. 8: Data for UMBmark for clockwise turn about the square

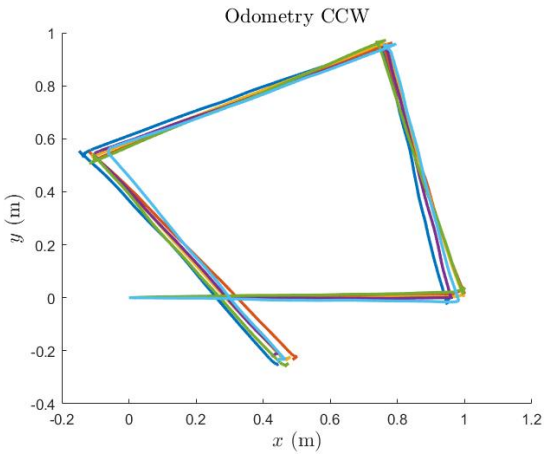

Fig. 9: Using the wheel angle to measure heading

TABLE VI: UMBmark correction factors

Correction factor	Values
E_d	1.01
E_b	1.125
α (degrees)	10
β (degrees)	0.1121

We took the 5 best readings to calculate the corrections for the wheel base and the wheel radius. The values of the correction factors are tabulated in the Table VI.

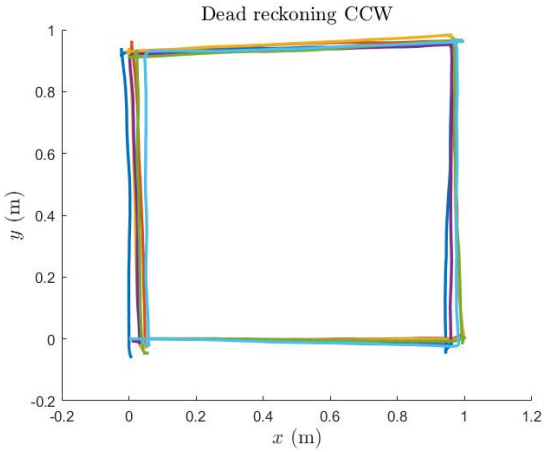

Fig. 10: Using the yaw from IMU to measure heading

B. Gyrodometry

We went through the paper [2] that was shared for doing gyrodometry, and they were using wires of known diameters as bumps in the path of the robot to get the $\Delta\theta_{thres}$ value. We thought that gyrodometry would only be helpful when there are significant bumps in the track. However, we noticed that the given autonomous tracks were smooth enough and so we decided not to implement gyrodometry in our odometry model. While collecting the data for UMBmark, we also logged the Tait-Bryan yaw angle given by the IMU. When we calculated x and y coordinates of the robot using these values, it was observed that these angle values were much better than using angles that was computed from odometry values, as can be seen from Figure 9 and 10. Hence, the fused IMU angles were used to determine the heading of the robot.

We used a Vernier caliper to measure the wheel diameter and the wheel base. The value of the wheel base was corrected using the UMBmark procedure. The encoder polarities were determined by trial and error, by noting the sign of the encoder readings. The gear ratio and the encoder resolution was read from the given datasheets. These parameters are listed in Table VII.

C. Odometry Results

Using the correction factors, we drove the robot through the square path autonomously. Our square path algorithm did not work well, but the odometry data was really close to the actual path that the robot had

TABLE VII: Odometry parameters

Parameters	Values
Right encoder polarity	-1
Left encoder polarity	1
Gear ratio	20.4
Encoder resolution	48
Wheel diameter	0.084 m
Wheel base	0.2216 m

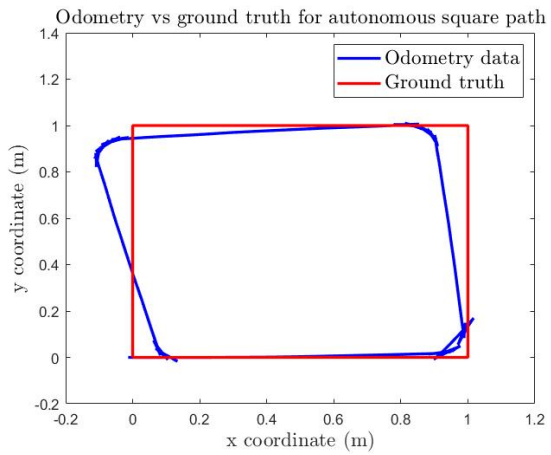

Fig. 11: Odometry data vs Ground truth for autonomously navigating through a square path

traversed. Figure 11 shows the comparison between the odometry data and the actual square path.

V. Path Planning

We designed two specific trajectories for the two tasks: drag race and square motion. For the drag race, we fed a velocity profile to the controller as a function of distance travelled. For the square, we either moved in a straight line motion or a turning motion depending on how much we had travelled. The following subsections describe the trajectories in more detail.

A. High-level Controllers

1) Distance controller: We implemented a proportional controller for making the bot travel arbitrary distances. The input to this controller was the difference between a reference distance and the actual distance travelled by the robot and the output was the required wheel velocity, $\dot{\phi}$. This value of $\dot{\phi}$ is then used as a setpoint for the outer loop controller of the balancebot. The distance is calculated by storing the coordinates of an initial point and measuring the Euclidean distance to the current position.

$$\dot{\phi} = K_p(d_{ref} - d) \tag{6}$$

Where d_ref is the reference distance needed to travel and d is the current amount of distance travelled. The value of K_p is mentioned below.

TABLE VIII: Distance controller gains

Gains	Values
K_p	7

2) Turning controller: We implemented a PD controller to rotate the robot through required angles. The input to this controller was the difference between the reference ψ value and the current ψ value. Since, we saw that the IMU gave really good yaw angles (fig. (10)), we used these values to get the current heading angle. The output of this controller was a required $\dot{\gamma}$ which was fed to the heading controller.

$$\dot{\gamma} = (K_p + \frac{K_d s}{1 + T_f s})(\psi_{ref} - \psi) \tag{7}$$

The parameters in 7 are listed in the table below

TABLE IX: Turning controller parameters

Parameters	Values
K_p	0.7
K_d	0.002
T_f	0.04

B. Square track

We divided this task into two sub-tasks:

1) Drive in a straight line for a specified distance
2) Turn for 90 degrees while maintaining a small forward velocity

For the first sub-task, we used the distance controller. The reference distance used was the side of square $(1m)$.

For the second sub-task, we used the turning controller. We initialized the ψ_{ref} value to zero and added $\pi/2$ to it after every turn.

C. Drag race

The task in this part of the competition was to get the bot through 11 meters of a straight path as fast as possible and stop and balance within a 1 meter distance of the finish line. For this we decided it was best to give a parabolic shape to the velocity profile as a function of distance travelled in the x direction (forward direction).

We gave an initial parabola to the bot for a small distance, to get it to a fast velocity in a short period of time. Then, a longer parabola with shallower gradients was given in the middle portion of the track and finally, a linear deceleration profile was given to get the bot to a stop slowly, making it easier to stay balanced at the end of the track. Figure 12 shows the velocity profile given to the controller. The various parameters used in the velocity profile are mentioned in Table X.

TABLE X: Drag race parameters

Parameters	Values
Initial distance	0.3 m
$\dot{\phi}_{max}$	38 rad/s
$\dot{\phi}_{initial}$	15 rad/s

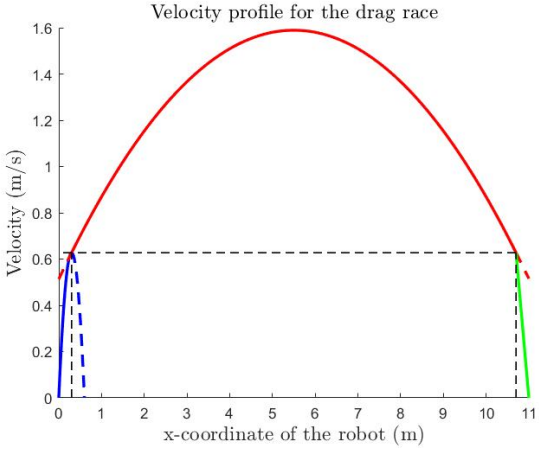

Fig. 12: Velocity profile for the drag race as a function of distance travelled in x direction. The initial parabola extends until the initial start distance and gets the $\dot{\phi}$ to $\dot{\phi}_{initial}$, the second parabola takes this value to $\dot{\phi}_{max}$ and returns it to $\dot{\phi}_{initial}$ after which the linear deceleration brings $\dot{\phi}$ to zero at the specified length

VI. Competition Performance

A. Balancing on the spot

The first task of the competition required the BalanceBot to balance in a 10cm square region without drifting from its equilibrium position, all while rejecting diisturbances. The BalanceBot was tested on its ability to recover to its original position from external disturbances. The controllers implemented were successful in balancing on the spot and return to the original position when given various perturbations. The team was able to get a full score of 200 on this task.

B. 4x4 Square laps

In this task, the BalanceBot was required to navigate 4 x 4 edges of 1m length. It was found that the even though the BalanceBot was able to make squares in each of its iterations, it was not able to make perfect square or be in the region marked by tape. The error in the angle was getting blown up by the end of the 4-turns. The team scored 100 points in this task.

C. Drag Race

The objective of this task was to race the BalanceBot on a 11m track and regain its balance at the end line within a 1m deadzone. The velocity profile described in the earlier section enabled the bot to complete this task in a winning time of 10.15 sec. It was observed that the time could have been further reduced if the length of the race track in the race parameter file had been reduced, to stop the forward motion of the bot much earlier. The team received a full score of 200 points on this task.

D. Race track (Manual Control)

This task required a user to manually drive to an obstacle race course, while avoiding obstacle and finishing the course in the least possible time. The race course was further bifurcated into two parts, the easy course and the hard course. The team was able to finish the easy course with a winning time of 31.23 sec and securing full score of 200 points. However, we timed the hard score at 119.53 sec, which scored us at 130 points. The hard course included a ramp, which the BalanceBot had to climb and then descend to complete a checkpoint. It was observed that the bumpers at the front of the bot were constantly bumping into the slop of the pyramid. During the trial runs we were able to decrease this issue by saturating the body angle at a much lower value, which in-turn provided for a stiff body angle and increased the performace of the bot on the race course.

The team was able to secure a 2^{nd} rank with a score of 830 points.

VII. Discussion

To conclude, a cascaded PID controller was implemented on a Mobile Wheeled Inverted-Pendulum (MWIP) system to balance the system on the spot as well as to balance its upright pose while in motion when commanded via remote control. In addition, two high-level controllers were also implemented to enable the BalanceBot to traverse arbitrary distances and turn specific angles. The outcome of the project showed a very robust BalanceBot which was able to navigate complex environments while rejecting external disturbances. There is still room for improvement in the proposed model. Even though the balancing controller was able to balance on the spot (within $\pm$0.1m), the issue arises when the robot starts any motion and small oscillations in its motion is observed. This is due to the fact that the outer loop control (wheel angle) gets dominated by the inner loop control (body angle). This issue can be alleviated by fine tuning the inner loop and the outer loop PID blocks and ensuring that the outer loop has more command over the system in general than the inner loop. The problems faced in the 4x4 square turns can be

solved as the resulting motion will be much smoother and will thus decrease the slippage of wheels.

REFERENCES

[1] J. Borenstein and L. Feng, "Measurement and Correction of Systematic Odometry Errors in Mobile Robots," *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 869–880, Dec 1996.

[2] ——, "Gyrodometry: A New Method for Combining Data from Gyros and Odometry in Mobile Robots," *IEEE International Conference on Robotics and Automation*, vol. 7, pp. 423–428, Apr 1996.

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