ROB550-F19 BalanceBot: Team 12 Report

Ahmed Alkatheeri, Nalin Bendapudi, and Jianping Lin
{ahmedak, bnalin, jplin}@umich.edu

***Abstract*—This report chronicles our work on a mobile-wheeled inverted pendulum and the methodology adopted to balance it and steer it using a manual remote controller. State space dynamics were formulated using the average wheel angle, the body angle and their velocities as the four states. A linear-quadratic regulator controller was designed to balance the bot about the unstable equilibrium point. A separate proportional controller was added to minimize the in-plane rotation of the robot. The robot could also be externally controlled using a manual DSM remote to make it traverse any complex path.**

***Index Terms*—LQR control, state space, dynamics, robot control library, beaglebone**

I. Introduction

Research of two-wheeled mobile robots has gained increasing interest in the past few years due to the emerging of many self-balancing personal transporters such as the Segway [1] and the toys of two-wheel scooters. Despite the promising results of balancing itself, efforts are still required to enhance the maneuverability to navigate on various terrains and turn with sharp corners. The two-wheeled mobile robots can be modeled as a mobile-wheeled inverted-pendulum (MWIP), where the control of the MWIP consists of the swing-up and balancing. The swing-up control moves the stable pendulum lying on the ground up to its unstable inverted position, and the balancing control balances it about the vertical. This controlling problem is highly nonlinear, and many control techniques have been studied for the control of these underactuated systems including the feedback linearization [2], passivity-based control [3], or the sliding-mode velocity control [4].

In this project, we focus on the balancing control of the MWIP system. We assume the pendulum is vertical initially and design a controller to balance upright even when disturbed or when traversing a commanded path. The balancing controller is based on the linear-quadratic regulator (LQR) method, which is a feedback controller for linear systems with minimized cost functions. We conduct different tasks to navigate a set of waypoints as fast as possible.

The rest of this report is organized as follows. In Section II, we get the modeling of the system and measure the required parameters. The balancing controller design is shown in Section III, where the LQR method is applied. In Section IV, we design the manual control with the ability to turn and change the heading of the robot. Odometry and motion control is illustrated in Section V. Section VI shows the results of the trajectory following. Finally, we present the conclusion and discussion of the project in Section VII.

Authors are listed in alphabetical order. All team members contributed equally to this project.

II. System Modeling

A. Dynamical Model of the System

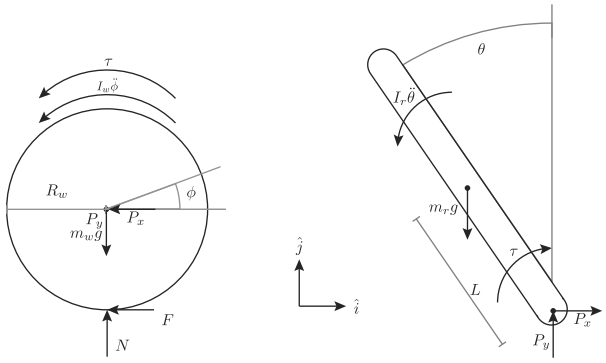

Fig. 1: Free body diagrams of the MWIP system [5].

The free body diagram of the MWIP system is given in Figure. 1, where θ and ϕ represent the body angle and the wheel position. The terms m_w and m_b represent the mass of the wheel and the upper body, respectively. The inertia of the wheel and the body about the center of mass is represented by I_w and I_b. The torque from motors is represented by τ, while R_w represents the radius of the wheel and L represents the length from the end to the center of mass (COM) of the body. Besides using the equations provided in the lecture notes, we also derived the dynamical equations based on the Euler-Lagrange equation [6]. The configuration vector is defined as $q = (\theta, \phi)^T \in \mathbb{R}^{2\times 1}$. The corresponding Lagrangian is $L(q, \dot{q}) : \ TQ \rightarrow \mathbb{R}$, which maps the tangent bundle to a real value as

$$L(q, \dot{q}) = K(q, \dot{q}) - V(q) = \frac{1}{2}\dot{q}^T M(q)\dot{q} - V(q),$$

where $K(q, \dot{q})$ is the kinetic energy based on the generalized mass/inertia matrix $M(q)$, and $V(q)$ is the potential energy. The forced Euler-Lagrange system is with 4-

dimensional configuration space Q, and the corresponding dynamics of $L(q, \dot{q})$ are given as

$$\frac{\mathrm{d}}{\mathrm{dt}}\partial_{\dot{q}}L(q,\dot{q}) - \partial_q L(q,\dot{q}) = B(q)\tau, \tag{1}$$

where $\tau \in \mathbb{R}$ is the control input and $B \in \mathbb{R}^{4\times 1}$ maps the inputs τ to the Euler-Lagrange system with $rank(B) = 1$. We can factor equation (1) into the common form as

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + N(q) = B(q)\tau \tag{2}$$

where $C(q, \dot{q})$ is the Coriolis matrix, and $N(q)$ is the gradient of the potential energy $V(q)$ along the generalized coordinates. To derive the state space model of the system, we have

$$x = [q^T, \dot{q}^T]^T = [\theta, \phi, \dot{\theta}, \dot{\phi}]^T,$$

and

$$\dot{x} = \begin{bmatrix} \dot{q} \\ M(q)^{-1}[-C(q,\dot{q})\dot{q} - N(q) + B(q)\tau] \end{bmatrix}.$$

We controlled the motor voltage V by setting the percentage duty cycle u of a PWM signal, where

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

where τ_s is the stall torque produced when the motor is not moving, and ω_{NL} is the no load speed that drives the torque to 0. Plugging back τ into the dynamical equations and notice that $\omega = \dot{\phi} - \dot{\theta}$, we have the following linearized system around the equilibrium point $x^* = [0\ 0\ 0\ 0]$, where

$$\dot{x} = Ax + Bu$$

$$= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{a_4 a_3}{c_1} & 0 & -\frac{b_2 c_3}{a_3 c_1} & \frac{b_2 c_3}{a_3 c_1} \\ -\frac{a_2 a_4}{a_1 a_3 c_1} & 0 & \frac{b_2 c_2}{a_1 c_1} & -\frac{b_2 c_2}{a_1 c_1} \end{bmatrix} x + \begin{bmatrix} 0 \\ -\frac{b_1 c_3}{a_3 c_1} \\ 0 \\ \frac{b_1 c_2}{a_1 c_1} \end{bmatrix} u,$$

where $a_1 = I_w + (m_w + m_b)R_w^2$, $a_2 = m_b R_w L$, $a_3 = I_b + m_b L^2$, $a4 = m_b g L$, $b_1 = 2\tau_s$, $b_2 = 2\tau_s/\omega_{NL}$, and $c_1 = 1\frac{a_2^2}{a_1 a_3}$, $c_2 = 1 + \frac{a_2}{a_3}$, $c_3 = 1 + \frac{a_2}{a_1}$. Since we are interested in θ and ϕ, the observation is defined as

$$y = Cx + Du = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} x,$$

where y is the output vector and $D = 0$ is the feed-forward matrix.

B. Measurements

We used the method from the lab manual to calculate the moment of inertia around the principle axes. As shown in Figure 2, we had two wires with length L carrying the robot. The distance between the wires was d. The principle axis of the levelled object and the vertical central line of the wires coincided. The object was rotated manually around the principal axis by a small angle. After the object was released, the periodic back-and-forth movement around the axis proceeded with its swing period and moment of inertia derived as

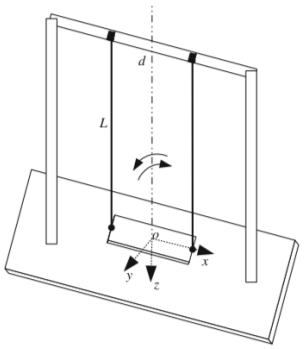

Fig. 2: Bifilar Pendulum setup.

$$T_0 = 4\pi\sqrt{\frac{J_{axis}L}{m_0 g d^2}},\ J_{axis} = \frac{m_0 g d^2}{16\pi^2 L}T_0^2.$$

For the y-axis, which is the most important axis for balancing the bot, we recorded the velocity of the oscillation from the IMU and calculated the average period as $T_0 \approx 3.5937[sec]$ in MatLab as shown in Figure. 3. We used scales to measure the mass of the wheel and the mass of the body without wheels as $m_w = 0.0923[Kg]$ and $m_b = 1.1142[Kg]$, respectively. The corresponding inertia of the body about center was calculated by the equation as $I_r = 0.0034[Kg \cdot m^2]$. Similarly, the moment of inertia around other principal axes was calculated and the values are shown in Table I. The inertia of the wheels were calculated by

TABLE I: Moments of Inertia around principal axes.

Axis	Value $[Kg \cdot m^2]$
x	0.0084
y	0.0034
z	0.0053

$I_w = 2(I_{gb} + \frac{1}{2}m_w R_w^2) = 0.0022[Kg \cdot m^2]$, where I_{gb} is the inertial of motor armature and gearbox and we assumed the mass of the wheel is uniformly distributed.

We measured the resistance of both motors and used the provided script `test_motor_params` in the bin folder to calculate the critical parameters for the motors. Results are given in Table. II, where R represents the motor coil resistance, and K represents the motor constant. We also measured the coefficient b of the viscous friction.

Other important parameters are shown in Table III.

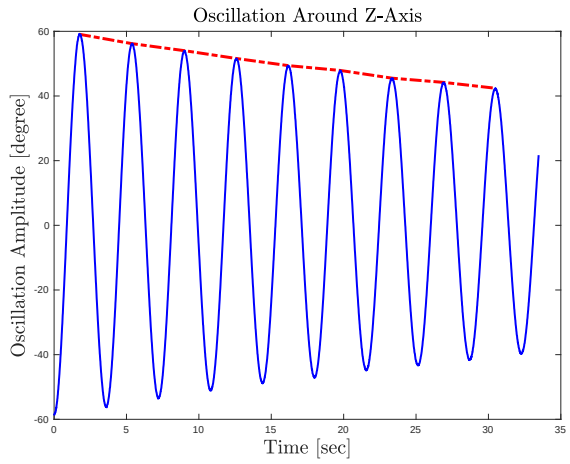

Fig. 3: Oscillation with respect to time around the Y-axis.

TABLE II: Motor parameters for both motors from script versus the manufacturer data.

Parameter	Left	Right	Default
$R[\Omega]$	6.6	9.8	5.7143
$\omega_{NL}[rad/s]$	39.6318	40.0155	38.7463
$K[N \cdot m/A]$	0.0140	0.0136	0.1412
$\tau_s[N \cdot m]$	1.8182	1.2245	0.2966
$b[N \cdot m \cdot s]$	0.0007	0.0006	N/A
$I_{gb}[Kg \cdot m^2]$	5.2624E-5	4.7297E-5	N/A

III. Balance Controller

A. Block Diagram and the LQR method

The block diagram of the system is shown in Figure 4. For a continuous-time linear system $\dot{x} = Ax + Bu$, the LQR method defines a cost function J as

$$J = \int_0^\infty x^T Qx + u^T Ru + 2x^T N u dt,$$

and tries to find the feedback control law that minimizes the value of the cost by $u = -Kx$, where $K = R^{-1}(B^T P + N^T)$, and P is found by solving

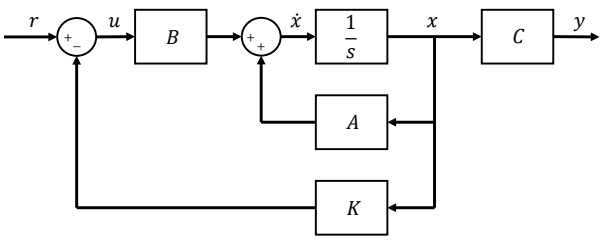

Fig. 4: Block diagram of the state-space equation.

TABLE III: General system parameters.

Parameter	Value
Encoder Res (ticks/motor rev)	48
Gear ratio (motor rev/wheel rev)	20.4
Wheel radius $R_w(m)$	0.042 ± 0.001
Base length d_{base} (m)	0.223 ± 0.001

the continuous time algebraic Riccati equation

$$A^T P + PA - (PB + N)R^{-1}(B^T P + N^T) + Q = 0,$$

where $N = 0$ for our case study. Putting back the control law into the system, we have $\dot{x} = (A - BK)x$, where the poles of the system, i.e., the eigenvalues of $A - BK$, lie in the left half plane and the system is made asymptotically stable with exponential converging speed.

We considered the discrete time system, as a result, we used `c2d` in MatLab to convert the continuous time system into a discrete time state space equation, and got the corresponding $\{A_d, B_d, C_d, D_d\}$. By using `dlqr(A_d,B_d,C_d,D_d)` in MATLAB, we had the corresponding gain $K = [-4.8858, -0.5817, -0.0423, -0.1659]$ with $Q = diag(1, 0.01, 10, 100)$ and $R = 2000$. The control law is $u = r - Kx$, where r is the extra step command. To reduce the steady state error, we also considered the integral of ϕ, where the state x was complemented with one more term $\int \phi dt$. We had $y = [\theta, \phi, \dot{\phi}]^T$ to observe the angle of the upper body, position, and velocity of the robot. After tweaking the gain values and adding the integral term, the final gain values reached was $K = [-4.8858, -0.5817, -0.0423, -0.1400, -0.0119]$.

B. Step Response of the Robot in Reality and Simulation

As shown in Figure 5, simulation requires more time to achieve target position. That is because the system was not modelled accurately and estimates were made to obtain the parameters defining the system. As for the real cases, we can observe high frequency oscillations caused by the system balancing the robot to an upright position. We tried to apply a low pass filter with a corner frequency of 30 Hz to reduce those oscillations, but that caused instability in the controller and it was no longer able to maintain a certain position within $\pm 0.1m$. Also, less oscillation are seen when the reference wheel position is increased. That is because, with a bigger difference between current wheel position and reference wheel position, the controller prioritizes reducing the difference instead of balancing the bot. So, smaller perturbations in the body angle would not result in immediate action by the controller. That resulted in a shorter rise time as observed in the plots.

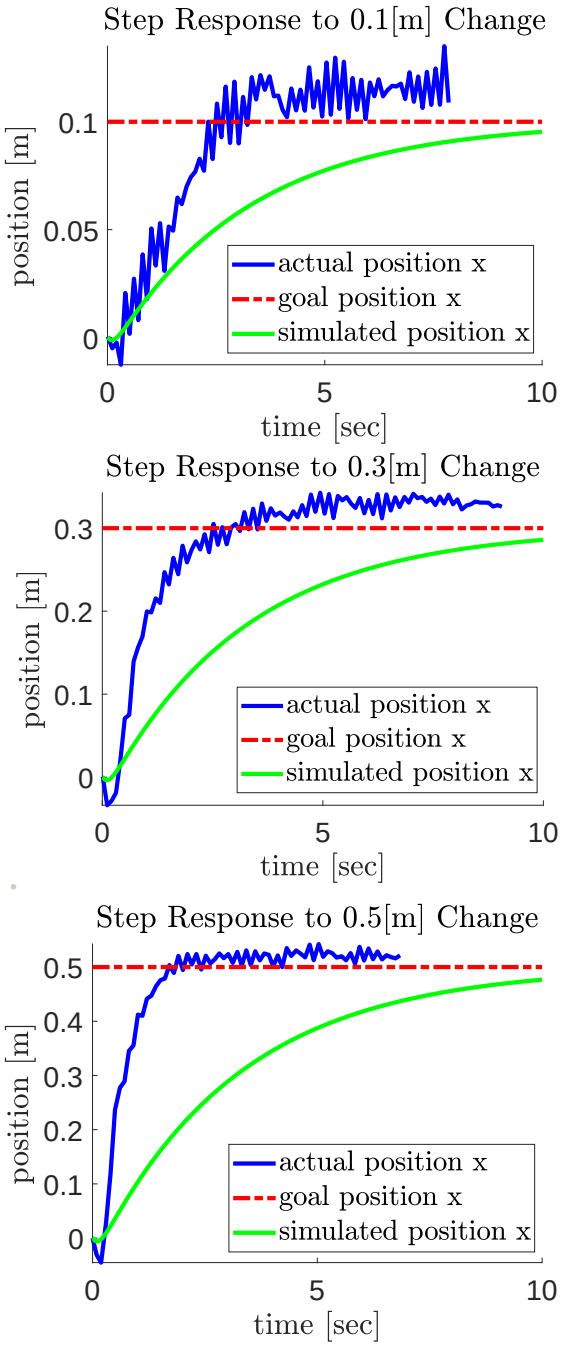

Fig. 5: From top to bottom: step response of the actual robot to a change of 0.1 m, 0.3 m, and 0.5 m in reference wheel position along with the simulated response of the model.

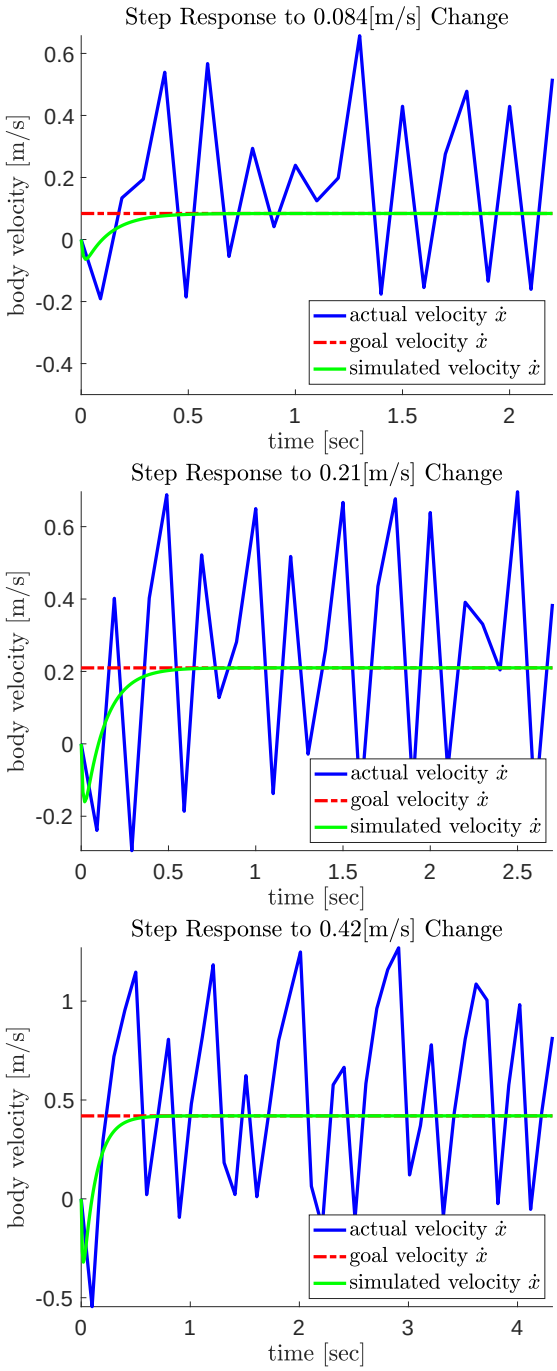

Fig. 6: From top to bottom: step response of the actual robot to a change of 0.084 m/s, 0.21 m/s, and 0.42 m/s in reference body velocity along with the simulated response of the model.

Figure 6 shows the experimental and simulated results to the change of body velocities. A lot of vibrations are observed and can be attributed to the controller balancing the bot into an upright position, however, the averaged body velocity achieves the desired value. Again, we notice that the oscillations in the speed get reduced when a bigger desired wheel speed is given to the system. Reducing the difference between desired and actual speed gets priority in the controller. Once the difference gets reduced, the small perturbations in the body angle start causing observable oscillations.

C. Updating the States and Calculating the Output

The application of the controller in the program required obtaining measurements of the states defined in subsection III-A. The first of which was the body angle (θ). The IMU used in the system along with the `Robot Control Library` provided this angle. Similarly, the rate of change of the body angle ($\dot{\theta}$) was obtained. As for the wheel angle (ϕ), equation (3) was used to obtain the angles of the left and right wheels separately. The equation converts the number of encoder ticks to an angle in radians. It adds the angle (θ) to cancel out the effect of the body angle on the encoder readings. Then, the average angle of the balancebot wheels (ϕ) was obtained using equation (4).

$$\phi_W = \frac{2\pi * EncoderTicks_W}{EncoderResolution * GearRatio} + \theta \quad (3)$$

$$\phi = (\phi_{RW} + \phi_{LW})/2 \quad (4)$$

The rate of change of the wheel angle ($\dot{\phi}$) was obtained by retaining the previous value of the wheel angle and using equation (5). In the equation, DT is the sampling time of the measurements, which was 0.01 s in this case since the sampling loop was repeated at a 100 Hz frequency.

$$\dot{\phi} = \frac{\delta\phi}{\delta t} = \frac{\phi - \phi_{previous}}{DT} \quad (5)$$

Finally, the integral of the wheel angle was calculated using equation (6). The integral term retains its value between iterations and is reset to 0 when the difference between the desired wheel angle and current wheel angle is bigger than a threshold of 10 radians.

$$\phi_{integral} = \phi_{integral} + \phi * DT \quad (6)$$

With the states measured, the duty cycle output (u) was calculated using the following controller equation. The same output was given to both wheels.

$$\begin{aligned} u = {} & K1 * (\theta - \theta_{desired}) + K2 * (\dot{\theta}) \\ & + K3 * (\phi_{integral} - \phi_{integral,desired}) \\ & + K4 * (\phi - \phi_{desired}) + K5 * (\dot{\phi} - \dot{\phi}_{desired}) \end{aligned} \quad (7)$$

IV. Manual Control

A. Heading Controller

To control the MWIP either manually or autonomously, we need to ensure that the robot is heading in the direction we intend it to. More often than not, the angular displacement of both the wheels is different from each other even though the duty cycle provided to both motors is the same. This maybe because of the difference in characteristics of the left and right motors, or difference in wheel sizes. That difference between the angles of the wheels causes the heading angle to change.

To control the heading angle, we employed a simple proportional controller. First we need to calculate the heading angle (γ) using the balance-bot parameters and sensor values. There are two ways to obtain the heading angle, the first is to simply read the gyro data from the IMU. The second is to use the difference in wheel angles between left and write as shown in equation (8).

$$\gamma = \frac{R_w}{d_{base}}(\phi_{RW} - \phi_{LW}) \quad (8)$$

Here ϕ_{RW} and ϕ_{LW} are the encoder readings from the left and right encoders, R_w is the radius of the wheels and d_{base} is the length of the base of the balancebot.

The duty cycle of the left and right motors are modified according to a proportional gain term, corresponding to the error in heading angle.

$$u_L = u + K_g(\gamma_{desired} - \gamma) \quad (9)$$

$$u_R = u - K_g(\gamma_{desired} - \gamma) \quad (10)$$

where K_g is the proportional gain and $\gamma_{desired}$ is the desired heading angle. If we want the MWIP to balance in place, $\gamma_{desired}$ should be zero. Otherwise, it is obtained through the manual steering input. The gain K_g had to be tuned to achieve good observable performance. The optimum value was simply 1.

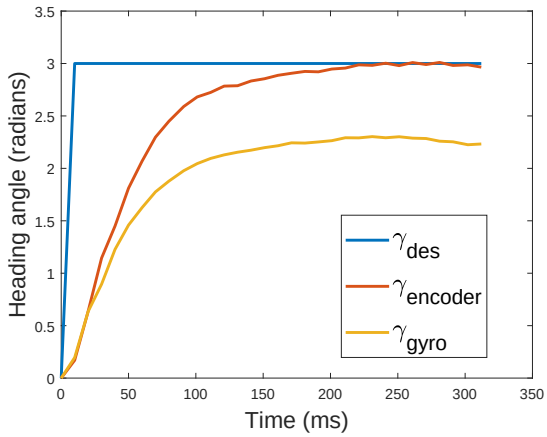

Fig. 7: Step Response of the robot to a change in three radians in the reference heading angle

To test the performance of the controller, a step input of $\gamma_{desired}$ = 3 rad was provided and the response is recorded in Figure 7. We note that the γ_{gyro}, which is the heading angle measured using the IMU achieves the given input in about 200ms using the proportional gain controller. Also note that the heading angle calculated from wheel angles, $\gamma_{encoder}$ is quite different from that of the gyro. This could be due to a number of environmental and systematic errors. This is discussed in detail in Section V.

B. RC Controller

A DSM receiver-transmitter was used to remotely communicate with the BeagleBone. The binary `rc_bind_dsm` was used to bind the remote controller to a receiver connected to the BeagleBone. Next, the binary file `rc_test_dsm` was run to get the exact values transmitted by the remote. `rc_calibrate_dsm` could be used to calibrate the values. By doing this we could find out which channels represented which buttons on the transmitter, and what's its corresponding range of values. Finally only 3 channels were used:

1) *Channel 3*: This was used to move the bot forward or backward. We changed the rate of change of $\phi_{desired}$ proportionally with the values of channel 3.
2) *Channel 4*: This was used to turn the bot left or right. We changed the rate of change of $\gamma_{desired}$ proportionally with the values of channel 4.
3) *Channel 5*: This gave a discrete output. We used this to switch between manual and autonomous mode.

V. Odometry and Motion Control

The application of odometry enables a system to take upper level instructions such as desired position or orientation and follow them. With odometry, the balancebot could get a list of waypoints that it can follow to reach a certain target.

A. Odometry Measurements

All the necessary parameters are mentioned in Table III in Section II. Equation (4) was used to obtain the average wheel angle. Equation (11) then calculated the actual distance travelled in meters based on the average radius of the wheels (R_w).

$$distance = \phi * R_w \tag{11}$$

As for the orientaion of the balancebot (γ), the comparison was made between the gyro-obtained angle and the encoder-obtained angle in Section IV. It was decided to use the angle obtained from the gyro since it was more reliable and consistent with actual measurements. As for the encoder-obtained orientation angle, there is a number of possible errors that would cause it to lose accuracy. Those errors, systematic and environmental, affect the system in ways that are not captured in the encoder readings. Those errors include:

1) Differences in wheel diameters
2) Error in base length measurement
3) Slipping of the wheel(s)
4) Uneven floors
5) Hitting obstacles
6) Differences in motor characteristics

One of the possible issues with dependence on the IMU reading for the orientation angle was the possibility of a drift in the readings after a certain duration. For the application of the balancebot and the intended tasks, reliable orientation readings were not necessary for a long duration. That is why no logic was implemented to fuse ($\gamma_{encoder}$) and (γ_{gyro}) into an overall orientation angle.

B. Error Correction in Odometry

It was decided to pursue correcting the systematic errors to increase the accuracy of the balancebot sensors and reduce any deviation in the desired trajectory. The specific errors pursued were difference in wheel diameters between right and left wheel, as well as any error in the measurement of the base length. Those errors were pursued because they are assumed to be the major sources of error in the balancebot. The process followed to characterize the error is described in article [7]. The first step was to have the balancebot moved around a square of known edge length while the x and y coordinates were measured. The final coordinates after a loop was completed should be (0,0), however, because of the systematic errors, they were not. That difference would be referred to as the error. Multiple loops were completed in both clockwise and counterclockwise directions to get an average of the error. Table (IV) shows the obtained error. Then, the equations were followed to obtain a

TABLE IV: Odometry Average Error.

Parameter	Clockwise	Counter Clockwise
e_x (m)	0.0305	0.00579
e_y (m)	-0.0216	-0.0104

characterization of both the wheel diameter error (E_d) and base length error (E_b). L mentioned in the equations is the length of the edge of the square, which was 1 m. Since two equations were provided for each (β) and (α), the average of the results was calculated and used.

$$\beta = \frac{e_{x,CW} - e_{x,CCW}}{-4L} \; or \; \beta = \frac{e_{y,CW} + e_{y,CCW}}{-4L} \tag{12}$$

$$\alpha = \frac{e_{x,CW} + e_{x,CCW}}{-4L} \; or \; \alpha = \frac{e_{y,CW} - e_{y,CCW}}{-4L} \tag{13}$$

By plugging in those results and the value of L into the equation of (E_d) from [7], equation (14) was obtained. Equation (15) shows how (E_b) was obtained.

$$E_d = \frac{1 + d_{base} Sin(\frac{\beta}{2})}{1 - d_{base} Sin(\frac{\beta}{2})} \tag{14}$$

$$E_b = \frac{\frac{\pi}{2}}{\frac{\pi}{2} - \alpha} \tag{15}$$

Table (V) shows the results obtained from those equations. (E_b) can be used to obtain a corrected (d_{base})

TABLE V: Odometry Error Characterization.

Parameter	Value
β (rad)	0.000907
α (rad)	-0.00314
E_d	1.0002
E_b	0.998

by multiplying it with the current (d_{base}). That changed the value of (d_{base}) from 0.223 m to 0.222 m. As for (E_d), equations (16) and (17) can be used to obtain a corrected wheel angle for the left and right wheels respectively. [7] The value found for (E_d) was small enough that when multiplied by the wheel diameter, it was not changed within the considered significant figures. Those corrections were not implemented since they would only become apparent with prolonged usage, which was avoided during the implementation of the desired tasks.

$$Corrected \; \phi_{LW} = \phi_{LW} * \frac{2}{E_d + 1} \tag{16}$$

$$Corrected \; \phi_{RW} = \phi_{RW} * \frac{2}{\frac{1}{E_d} + 1} \tag{17}$$

C. Odometry Controller

To get the system to travel a certain distance or follow a certain path, the application of a higher level controller was necessary. To change the setpoint of either distance (ϕ) or orientation (γ) to a value that is far from the current state would throw the system out of balance. Instead, the the system was provided with incremental setpoints that were followed until the overall setpoint was reached. The following equations were used to provide those incremental setpoints to the LQR controller. First, for the balancebot to travel an arbitrary distance ($distance_{overall}$), equation 18 was used to obtain a final wheel angle.

$$\phi_{final} = \frac{distance_{overall}}{R_w} \tag{18}$$

Then, that final wheel angle was used in a conditional statement that affected the setpoints of (ϕ), ($\dot{\phi}$), and (θ) as shown in the following equations. While the wheel angle setpoint ($\phi_{desired}$) is less than (ϕ_{final}), the equations were used to obtain new desired setpoints for those states. Once that condition no longer held, the states were set to maintain (ϕ_{final}).

$$\dot{\phi}_{desired} = \frac{0.4}{R_w} \tag{19}$$

$$\phi_{desired} = \phi + 6 * \dot{\phi}_{desired} * dt \tag{20}$$

$$\theta_{desired} = 0.001 * \dot{\phi}_{desired} \tag{21}$$

Equation (19) sets the desired speed of the wheels to 0.4 m/s. This speed was found through trials to find the fastest possible speed without the system going out of balance. Equations (20) and (21) were used to provide the system with setpoints for the positional angles so that they do not conflict with the desired ($\dot{\phi}$) and cause the terms in the controller equation to cancel each other out. As for the desired orientation ($\gamma_{desired}$), it was observed that giving the system a step of (π) did not cause it to go out of control. With that, logic similar to the one used for the desired distance above can be implemented to get to a desired orientation. Considering that the gyro returns an orientation angle between (-π) and (π), any desired orientation angle would need to be within that range. The following equations were found to get a normalized desired orientation angle.

$$normalized \; \gamma_{desired} = (2\pi) * (\gamma_{desired} \% (2\pi)) + \pi \tag{22}$$

Where (%) is the remainder symbol. This would return a desired orientation between (-π) and (3π). A (2π) is subtracted from that if it yielded a result over (π). That allows for a result that is between (-π) and (π). Then, any desired orientation can be reached in a maximum of two incremental setpoints each having a maximum value of (π). This can be improved further by utilizing conditions that find the shortest distance between the setpoint and current orientation. One possible implementation of this is to check the difference between desired and current orientation. If the distance was more than (π) then the controller would need to flip the direction of rotation. Considering the defined tasks for the balancebot, a maximum change in orientation angle of ($\frac{\pi}{2}$) was necessary. Therefore, this logic was not implemented, however, it can be a possible improvement on the system.

VI. Trajectory Following

Figure (8) shows the readings from the balancebot while given the task of going around a square with 0.9 m edge length. In real life, the bot was observed to follow the square lines, however, the encoders do not show that exactly. The way the distance controller

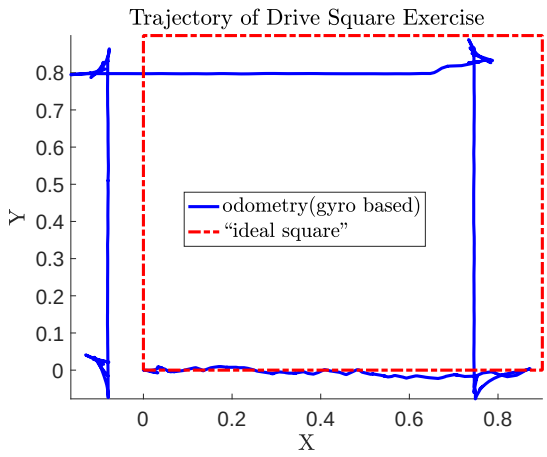

Fig. 8: The trajectory as determined by odometry (gyro based) and the "ground truth".

was implemented, as discussed in section V, was not perfected to reach a desired location with no overshoot. The desired speed was set to maximum while the desired position was not reached. That does not allow the system to slow down if the target location was closer. Instead, it is observable that the balancebot goes beyond the desired position and then returns to the setpoint. Another noticeable issue is the calibration of the LQR controller. The controller was not calibrated to allow the wheel angle (ϕ) to maintain a desired location. Instead, there is a steady state offset that was not removed completely. The overall shape of the motion was promising. With a few changes to the controllers, the square-shaped motion could be followed perfectly.

VII. Conclusion

Overall, an LQR controller was designed and used to balance the system as well as control its motion. Higher level controllers were designed to control the distance travelled and orientation of the system. The outcome showed a very stable balancebot when faced with any external disturbance. It could maintain an average position and speed within 20% of the desired respective values. And all the parts in the check list were completed as well as all the tasks in the competition. Even though the goal was reached, there are still many issues that can be tackled. The balance controller we designed was stable when faced with external disturbances but couldn't hold a constant position within a $\pm 0.05m$. The forward velocity of the robot was limited since the controller was trying to stabilize θ at 0 irrespective of the given velocity. The distance controller outputs a maximum speed request instead of giving a speed proportional to the distance to be travelled. The dependence on the orientation angle from the gyro would introduce errors with prolonged tasks. Those are all possible areas where the system can be improved if more time was given.

References

[1] "Segway personal transportation that simply moves you," http://www.https://www.segway.com/, accessed: 2019-11-06.

[2] M. W. Spong, "The swing up control problem for the acrobot," *IEEE control systems magazine*, vol. 15, no. 1, pp. 49–55, 1995.

[3] I. Fantoni, R. Lozano, and M. W. Spong, "Energy based control of the pendubot," *IEEE Transactions on Automatic Control*, vol. 45, no. 4, pp. 725–729, 2000.

[4] J. Huang, Z.-H. Guan, T. Matsuno, T. Fukuda, and K. Sekiyama, "Sliding-mode velocity control of mobile-wheeled inverted-pendulum systems," *IEEE Transactions on robotics*, vol. 26, no. 4, pp. 750–758, 2010.

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

[6] R. M. Murray, *A mathematical introduction to robotic manipulation*. CRC press, 2017.

[7] J. Borenstein and Liqiang Feng, "Measurement and correction of systematic odometry errors in mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 6, pp. 869–880, Dec 1996.