DP2: Controller for a Differential-Drive Segbot

Shishir Bhatta[1]
University of Illinois Urbana-Champaign, IL, USA

This paper describes the creation and optimization of a controller for a segbot that is designed to be on a space station that generates artificial gravity towards the center of it. The dynamics of the segbot were linearized and standardized to create a standard state space model. A Linear Quadratic Regulator (LQR) was used to minimize a cost function to pick the best controller gains and optimize the performance of the system. Requirements and verification methods centered around controllability by a non-engineer were set to create a successful controller, which was accomplished.

I. Nomenclature

$e_{lat}, \dot{e}_{lat}$	=	lateral error (m), change in lateral error with respect to time (m/s)
v	=	change in lateral error with respect to time (m/s)
$\phi, \dot{\phi}$	=	heading error or yaw (rad), turning rate or yawing velocity (rad/s)
$\theta, \dot{\theta}$	=	pitch angle (rad), pitch rate (rad/s)
τ_R	=	right wheel torque (N * m)
τ_L	=	left wheel torque (N * m)
ω_ϕ	=	change in heading error with respect to time (rad/s)
ω_θ	=	change in pitch angle with respect to time (rad/s)
A, B	=	coefficient matrices
u	=	input vector
x	=	state vector
K	=	linear feedback control gain matrix
W	=	controllability matrix
Q	=	state cost matrix
R	=	control cost matrix

II. Introduction

The differential-drive robot is a method of creating mobile robots. An example of a differential-drive vehicle can be seen in a Segway, which employs two wheels that may differ in speed to allow the vehicle to move straight, left, or right [1]. Due to their simplicity, they are very effective robots for space applications. The differential-drive robot works perfectly with a space station that generates "artificial gravity." This report aims to create an effective controller for a differential-drive robot. The various requirements determining the success of this segbot will be detailed later, as well as the methodology and analysis of the results.

III. Theory

A. Linearization and Standardization of State Space Model

This project required a controller that allows a differential-drive robot to avoid obstacles while aboard a space station generating artificial gravity. The dynamics of the robot were determined as such:

[1] Undergraduate Student, Aerospace Engineering.

$$\begin{bmatrix} \dot{e_{lat}} \\ \dot{v} \\ \ddot{\phi} \\ \ddot{\theta} \end{bmatrix} = f(e_{lat}, v, \phi, \dot{\phi}, \theta, \dot{\theta}, \tau_l, \tau_r) \tag{1}$$

where e_{lat} is lateral error (m) or the distance from the wheel center of the robot (positive values indicate it is far too left and negative values indicate it is far too right), $\dot{e_{lat}}$ is the change in lateral error with respect to time (positive values indicate moving left), ϕ is the heading error or yaw (rad) which is the difference in the orientation of the robot and the direction of the station ring (positive values indicate it is far too left), $\dot{\phi}$ is the turning rate or yawing rate (rad/s) (positive values indicate it is turning left), θ is the pitch angle (positive indicates it is pitching forward), $\dot{\theta}$ is the pitch velocity (positive values indicate turning forward), τ_R is the right wheel torque applied by the chassis on the right wheel (positive values indicate forward rotation of the wheel) and τ_L indicates the left wheel torque applied by chassis on the left wheel (positive values indicate forward rotation of the wheel).

Given the current form of the dynamic equations, it is challenging to solve the differential equations due to their nonlinearity and second-order variables. For this reason, it is necessary to turn this model into standard form, a set of linear differential equations describing the system's dynamics. This is done by defining:

$$\dot{\theta} = \omega_\theta \tag{2}$$
$$\ddot{\theta} = \dot{\omega_\theta} \tag{3}$$
$$\dot{\phi} = \omega_\phi \tag{4}$$
$$\ddot{\phi} = \dot{\omega_\phi} \tag{5}$$

With the new equations, $\ddot{\phi}$ and $\ddot{\theta}$ can be substituted by $\dot{\omega_\phi}$ and $\dot{\omega_\theta}$ to created first – order differential equations, but 2 new equations must be added to the state vector to define the new variables. Thus, the new states of our segbot can be represented as:

$$\begin{bmatrix} \dot{e_{lat}} \\ \dot{v} \\ \dot{\omega_\phi} \\ \dot{\omega_\theta} \\ \dot{\phi} \\ \dot{\theta} \end{bmatrix} = f(e_{lat}, v, \phi, \omega_\phi, \theta, \omega_\phi, \tau_l, \tau_r) \tag{6}$$

Where *f* is redefined as:

$$\begin{bmatrix} v\sin(\phi) \\ \dfrac{-18.4\tau_l - 18.4\tau_r - 14.4(\dot{\phi}^2 + \dot{\theta}^2)\sin(\theta) + 37.4(0.48\dot{\phi}^2\sin(2\theta) - \tau_l - \tau_r + 23.5\sin(\theta))\cos(\theta)}{89.9cos^2(\theta) - 93.5} \\ \dfrac{-1.06(\dot{\phi})(\dot{\theta})\sin(2\theta) - 2.4(\phi)v\dot{sin}(\theta) - 1.\dot{0}8(\tau_l) + 1.08(\tau_r)}{0.96sin^2(\theta) + 1.03} \\ \dfrac{-7.49\dot{\phi}^2\sin(2\theta) + 15.6\tau_l + 15.6\tau_r + 2.4(3.08\tau_l + 3.08\tau_r + 2.4(\dot{\phi}^2 + \dot{\theta}^2)\sin(\theta))\cos(\theta) - 367.0\sin(\theta)}{5.76cos^2(\theta) - 5.99} \\ \omega_\phi \\ \omega_\theta \end{bmatrix} \tag{7}$$

The dynamics can now be represented in the standard state-space model, which describes our systems' dynamics with vectors that represent how our states change with respect to time and how certain control inputs affect our system as well. We can describe it as such:

$$x = \begin{bmatrix} e_{lat} \\ v \\ \omega_\phi \\ \omega_\theta \\ \phi \\ \theta \end{bmatrix} \quad u = \begin{bmatrix} \tau_l \\ \tau_r \end{bmatrix} \tag{8}$$

where *x* represents the vector of states with no control input and *u* represents the control input. The typical state-space model can be represented as such:

$$\dot{x} = Ax + Bu \tag{9}$$

where *A* is the state matrix, which determines how out states evolve with respect to time with no control input. *B* represents the input matrix, which defines how the control inputs impact the system. Since we are modeling the system in state-space form, we must compute the *A* and *B* matrices. Since the dynamics are nonlinear due to the trigonometric functions, it must be linearized by computing the dynamics' Jacobians. We must also find equilibrium points that satisfy our system to be linearized around. For *A*, the Jacobian should be computed with respect to e_{lat}, v, ω_{ϕ}, ω_{θ}, ϕ, θ, since these are the values in the state matrix. For *B*, we should compute it with respect to τ_l, and τ_r. The equilibrium points must, when substituted, result in the $\dot{x}$ vector becoming a zero vector, so our equilibrium points are as follows:

$$e_{lat,e} = 0, \omega_{\phi e} = 0, \omega_{\theta e} = 0, \phi_e = 0, \theta_e = 0, \tau_{le} = 0, \tau_{re} = 0, v_e = 1 \tag{10}$$

We can now find the linearized A and B matrices centered around our equilibrium points.

$$A = \begin{matrix} 0 & 0 & 0 & 0 & 0.2 & 0 \\ 0 & 0 & 0 & 0 & 0 & -245.06 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1592.92 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{matrix} \tag{11}$$

$$B = \begin{bmatrix} 0 & 0 \\ 15.53 & 15.53 \\ -1.05 & 1.05 \\ -99.68 & -99.68 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{12}$$

With the *A* and *B* matrices converging on the equilibrium points, we will have a segbot that will remain upright if the controller works properly.

B. Controllability of the System

To ensure that our system is controllable, we will have to construct a controllability matrix. The matrix is defined by:

$$W = [B \quad AB \quad A^2B \quad \dots \quad A^nB] \tag{13}$$

where *W* is the controllability matrix. *n* indicates the order of the system, which is 6 in our system. When computing this matrix, we found the rank of the matrix to be 6, which is equal to the order of the system. In order to have a controllable matrix, the rank of the controllability matrix must match the order of the system. Thus, our system is controllable.

C. Linear Quadratic Regulator

To find the gains that fulfill the requirements for our controller, we must re-write our state-space model equation to:

$$u = -Kx \tag{14}$$

$$\dot{x} = Ax + B(-Kx) = (A - BK)x \tag{15}$$

where *K* is the gain matrix. The controller follows a simple linear state feedback law which makes it easier for us to find more optimal gains using a linear quadratic regulator (LQR). We must generate weights for the *Q* and *R* matrix, which will helps us find the controller gains. *Q* and *R* are diagonal matrices, with each value along the diagonal affecting a weight related to a different value. The values in *Q* affect values in the *x* vector, while the *R* matrix affects the values in the *u* vector. A higher value indicates a higher weightage on the time needed for the value to converge. The goal of the LQR is to minimize the cost function as defined below:

$$J = \int_0^\infty (x^T Q x + u^T R u)\, dt \tag{16}$$

where *J* is defined as the cost function of this time-invariant system. The cost was minimized by changing the gains for *Q* and *R* matrices, reducing the cost to this least amount possible. We also wanted to emphasize certain inputs, so certain gains are higher than others. In order to solve for the controller gains, we must solve for the Ricatti equation, which we can do by using the scipy library and accessing the solve_continuous_are() function. We will define the solution to the Ricatti equation as P.

$$Q = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 \end{bmatrix} \tag{17}$$

$$R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{18}$$

$$P = \begin{bmatrix} 12.84 & 0 & 1.17 & 0 & 3.00 & 0 \\ 0 & 4.00 & 0 & 0.63 & 0 & 3.37 \\ 1.17 & 0 & 1.17 & 0 & 1.00 & 0 \\ 0 & 0.63 & 0 & 0.12 & 0 & 0.69 \\ 3.00 & 0 & 1.00 & 0 & 2.34 & 0 \\ 0 & 3.38 & 0 & 0.70 & 0 & 40.56 \end{bmatrix} \tag{19}$$

Now that the solution for the Ricatti equation has been found, we can find the gain matrix,

$$K = \begin{bmatrix} -1.22 & -0.71 & -1.22 & -2.39 & -1.05 & -16.90 \\ 1.22 & -0.71 & 1.22 & -2.39 & 1.05 & -16.90 \end{bmatrix} \tag{20}$$

Where K is our controller gain matrix. Thus, we have created a controller that allows our system to be asymptotically stable.

IV. Experimental Methods

A. Requirements

The requirements were deemed by necessity to create a controller that kept the segbot upright and in contact with the space station in the simulation. It should also be able to follow the directions of a non-engineer controlling the bot. For these reasons, the requirements were centered around the lateral error, pitch, and yaw angle, as these are the measurements that determine whether the segbot completes these tasks.

1) The segbot shall not let the absolute magnitude of the lateral error exceed more than 0.25 meters after 5 seconds of a 10-second simulation if the initial lateral error lies between -0.5 to 0.5 meters.
2) The segbot shall remain upright at 0 radians with the absolute magnitude of the difference of actual pitch angle to the equilibrium pitch not exceeding 0.1 radians after 5 seconds of the 10-second simulation.
3) The segbot controller shall not allow the absolute magnitude of the difference of yaw angle from equilibrium to exceed 0.2 radians 5 seconds after an adjustment to the target direction (which is defined as the distance from the pointer in the simulation at the initial target direction to the adjusted target direction's pointer) of less than 0.5 meters in either direction for a 10-second simulation. The adjustment should be done within any point of the first 2 seconds of the simulation.

B. Verification

The verification for all the requirements can be done by using the Condynsate simulator library. We can simulate the environment the segbot will have to go through the library and by running 10-second trials. Data will be collected on the lateral error, pitch, and yaw angle at each time step of the simulation and graphed after the simulation has ended. The first requirement will be fulfilled if the absolute magnitude of the lateral error remains below 0.25 meters when the absolute magnitude of the initial lateral error lays between 0 to 0.5 after 5 seconds of the simulation being run. The lateral error before 5 seconds does not matter, but when it reaches 5 seconds, we must meet the requirement. The second requirement is satisfied if the segbot remains upright, maintaining a pitch angle of 0 radians, while not deviating more than 0.1 radians in either direction from this equilibrium throughout the simulation. The third requirement is fulfilled if the difference between the equilibrium yaw angle and the robot's yaw angle does not exceed 0.2 radians after 5 seconds of the adjustment to the target direction. The target direction can be adjusted by using the 'l' and 'j' keys, adjusting the target direction right or left respectively. The target direction will be adjusted once within the first 2 seconds with a magnitude of between 0 - 0.5 meters in either direction and the segbot must reduce the yaw error magnitude to between 0-0.1 radians. If it does not meet these conditions during verification, the controller has failed.

C. Extensive Testing

In order to find the most optimal controller, we needed to run multiple trials of the simulation with different weights for our Q and R matrices, emphasizing different states and inputs. We began by keeping Q and R as the identity matrices for their respective shapes, and then running the program to see how it would act. Then, we adjusted the weights for the values that affected lateral error, then reset it back to 1. We did the same thing for pitch angle and yaw angle, so we could see how the segbot acted when under those conditions. We then kept experimenting and changing the various gains related to those values. As we did this, we found the cost function (see Eq. 27) and tried to minimize that as much as possible as well to come up with the most optimal controller with the least amount of control input. Once we landed on a Q matrix that minimized our cost, we adjusted the R matrices as well. Since uneven torques cause the segbot to turn, we decided the gains of the R matrix should match. After various changes we found an R matrix that generated a low gain and met our requirements. We then started adjusting the initial values of our simulation and changing various parts of the simulation to find the limits of the controller. This process helped us find the flaws in our controller and helped us make a few more adjustments again.

V. Results and Discussion

A. Results of Requirement 1

The verification of requirement 1 required completing trials with different initial later errors and looking at the absolute magnitude of the lateral error, ensuring it sits below the 0.25 meters limit after 5 seconds. We began testing with an initial lateral error of 0 meters, and gradually increased the error until we reached 0.5 meters. Fig. 1 (below) shows the lateral error starting off at 0.5 meters, but eventually coming below a magnitude of 0.25 meters, oscillating around 0 meters. Fig 1 demonstrates one of the extremities our controller was required to face, and it clearly passes by staying below the ±0.25 lateral error constraint after 5 seconds.

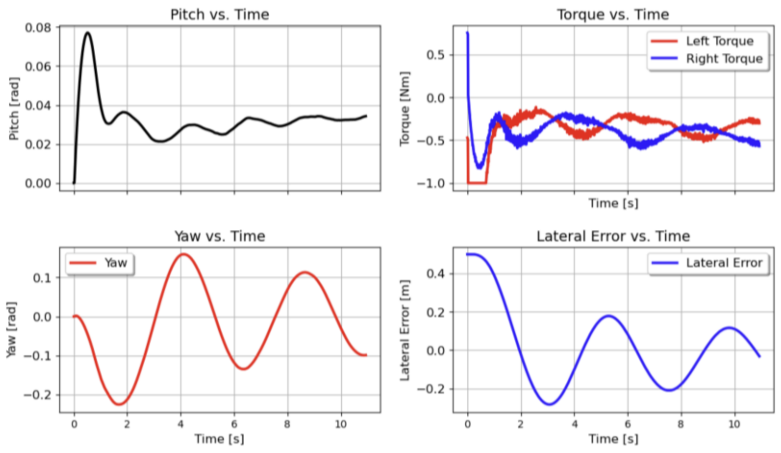

Figure 1: initial e_{lat} at 0.5 meters

B. Results of Requirement 2

The results of this requirement were tested with every trial completed, as the segbot would have to remain upright for the segbot to show forward movement throughout the 10-second simulation. The difference between the equilibrium pitch angle and the actual pitch angle was always less than 0.1 radians after 5 seconds of the simulation, as seen in Fig. 1 and Fig. 2. Thus, requirement 2 is satisfied for our controller.

C. Results of Requirement 3

The results of this requirement were tested by changing the target direction every 5 seconds and assuring the absolute magnitude of the difference between the equilibrium yaw angle and the current yaw after 5 seconds of that change stays below 0.1 radians. This was done similarly to how requirement 1 was satisfied, by gradually increasing the amount the target direction is adjusted in the simulation. As seen in Fig. 2, we can see that 5 seconds (timestamp 6 seconds) after the adjustment (which is around 1 second) the yaw error magnitude stays below 0.2 radians throughout the 10-second duration of the simulation. This is one of the extremities required to be tested by the controller, which it satisfies.

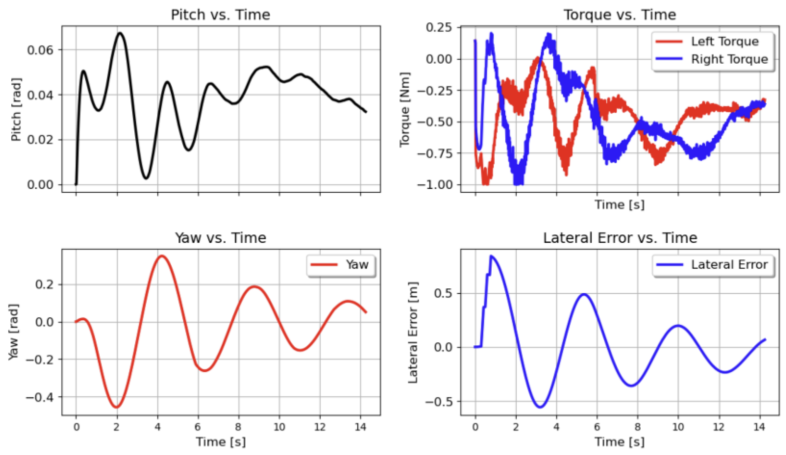

Figure 2: Target direction adjustment by 0.65 m

D. Discussion and Analysis

The best controller was determined to have a *Q* and *R* matrices that emphasized the weights related to the lateral error, pitch angle and pitch angle velocity. Since these states were the ones that were important to our requirements and created a working controller, we emphasized testing those weights. We also found that very large equilibrium *v* made the segbot fall over very quickly and it remained stable at slower speeds, so we decided to stay at 0.2 m/s. This may be due to the mechanical constraints on the wheels, which prevented which were limited to only 1 Nm of torque, so they could not maintain the higher speed. We found that the segbot's controller failed to keep the segbot on course when the initial lateral error exceeded 1 meter and if the target differed by more than 1 meter, the segbot would fall over. The maximum initial lateral error the segbot's controller could handle without tipping over was about 0.78 meters, while the maximum target distance change was around a similar value. Once again, this may be due to the mechanical constraint on the segbot, which caused the segbot to attempt to overdo the torque, causing it to tip over. Overall, we can see the segbot can handle slow speeds with relatively small changes in direction.

VI. Conclusion

This report details the process and theory behind the design of a controller for a differential-drive robot that was intended for the use of a non-engineer to control the robot. The controller gains were found through LQR and a linear feedback controller was used to create an asymptotically stable system. The segbot's lateral error never exceeded 1 meter after simulating for 5 seconds after the target was adjusted, which met the first requirement. The pitch angle remained within 0.1 radians of the upright function after 5 seconds of the simulation being run, meeting the second requirement as well. Finally, the controller kept the segbot's yaw angle within 0.1 radians of the equilibrium after simulating for 5 seconds, meeting the last requirement. It was found that LQR is a viable method to find the controller gains and a linear feedback law is sufficient for this application. This controller works well for applications where there isn't a huge time constraint or the need for large and sudden changes in direction. An area for growth could be exploring how more complex control laws could improve the performance of a segbot and exploring how the station's centripetal velocity could affect the segbot.

VII. Acknowledgements

We would like to thank Grayson Schaer for his valuable feedback on the draft of this report, as well as Arna Bhardwaj for helping debug the coding portion of the project and feedback on the report. We would like to thank Professor Wayne Chang as well for providing the resources and background to get started on this project.

VIII. References

[1] "Cheng W. "DP-2 Differential Drive Robot in Artificial Gravity," AE353 Aerospace Control Systems Available:https://canvas.illinois.edu/courses/43797/pages/dp-2-differential-drive-robot-in-artificial-gravity?module_item_id=3278932 Accessed: February 24, 2024

IX. Appendix

Date	Task	Person
2/27/24	Theory	Shishir Bhatta
2/27/24	Linearization and standardization of state-space model in code	Shishir Bhatta
2/28/24	LQR in code	Shishir Bhatta
2/28/24	Experimental Methods	Shishir Bhatta
3/4/2024	Simulation Code Completed	Shishir Bhatta
3/7/2024	Results, Conclusion etc. Completed	Shishir Bhatta

Team Reflection: I think I could’ve worked on this a little earlier and started on the report so I didn’t have to rush portions of it. It would’ve also made it easier if I had a team.