AE 353 Design Project 2

Ganesh Ravisankar
Department of Aerospace Engineering, University of Illinois at Urbana-Champaign

The goal of this design project is to design and implement a controller for a differential drive robot (segbot) that moves along a circular track. The controller and simulation are implemented and have been tested in a Jupyter Notebook

I. Introduction

A differential-drive robot consists of a chassis, a right wheel, and a left wheel. The wheels are driven by motors. By applying torques to these wheels, the robot can generate movement. Depending on how much torque is applied by the motor on the wheel, the differential-drive robot can either turn left or right:

- If the torque on the right wheel is larger, the robot turns left.
- If the torque on the left wheel is larger, the robot turns right.

The main objective of this project is to design a controller that will keep the differential-drive robot close to the center of the track. This means minimizing the lateral and heading error of the robot while maximizing the forward velocity of the robot.

II. Background Theory

A. Equations of motion

Upon looking closely at the problem statement, it is noted that the influence of gravity is ignored. It can be assumed that the forces of gravity are completely counteracted by reactionary forces that arise from the ground. This suggests that there are no system dynamics in the vertical direction. This results in five degrees of freedom: planar movement, pitch, roll, and yaw. However, the roll motion is governed by the variable `ground_pitch` which refers to the track inclination to the horizontal. As such, the system dynamics of the robot itself have four independent variables corresponding to planar movement, pitch, and yaw. These four independent variables characterize six internal states of the robot, suggesting that this system is non-holonomic. As such, that equations of motion for linear motion must be disregarded and the function f that characterizes the internal state of the robot is given by:

$$f\left(e_{lateral}, e_{heading}, v, \omega, \theta, \theta, \tau_L, \tau_R\right) = \begin{bmatrix} v \sin\left(e_h\right) \\ w \\ -\frac{2400\tau_L 2400\tau_R 2808\left(\theta^2 w^2\right)\sin\left(\theta\right) 13\cdot\left(250\tau_L 250\tau_R - 195w^2\sin\left(2\theta\right) - 8829\sin\left(\theta\right)\right)\cos\left(\theta\right)}{11700\cos^2\left(\theta\right) - 12168} \\ \frac{32\left(-875\tau_L 875\tau_R - 1443\theta w\sin\left(2\theta\right) - 2925vw\sin\left(\theta\right)\right)}{13\cdot\left(3120\sin^2\left(\theta\right) 2051\right)} \\ \theta \\ \frac{42250\tau_L 42250\tau_R - 32955w^2\sin\left(2\theta\right) 300\cdot\left(100\tau_L 100\tau_R 117\left(\theta^2 w^2\right)\sin\left(\theta\right)\right)\cos\left(\theta\right) - 1492101\sin\left(\theta\right)}{1404\cdot\left(25\cos^2\left(\theta\right) - 26\right)} \end{bmatrix} \tag{1}$$

It is desirable to obtain a system of the form

$$x = Ax \;\; Bu \tag{2}$$

where $x = \left(e_{lateral}, e_{heading}, v, \omega, \theta, \theta\right)$ is the state variable and $u = \left(\tau_R, \tau_L\right)$ is the system input. The input u can also be expressed in the form $-Kx$, where K is the gain matrix (explained in an upcoming section).
The system can be linearized in two steps:

1) First, choose a suitable equilibrium point. At the equilibrium point, it is desirable that the function f drops out. This means that the system dynamics described by x also drops out. A trivial equilibrium point would be when all states are set to 0. However, a better option would be to set v alone to some real number and the rest of the states to 0. This would ensure that the robot would be in motion when the system is at equilibrium.
2) Second, obtain the matrices A and B by taking the Jacobian of f at the equilibrium point.

After performing the two steps above, the following are the matrices A and B:

$$A = \begin{bmatrix} 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -245.25 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 \\ 0 & 0 & 0 & 0 & 1062.75 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 12.0726495726496 & 12.0726495726496 \\ 1.05014439485429 & -1.05014439485429 \\ 0 & 0 \\ -51.460113960114 & -51.460113960114 \end{bmatrix}$$

B. Verifying controllability

In order to obtain a system for which the state can be manipulated through a linear relationship, the system first needs to be controllable. A system is controllable if the controllability matrix W is invertible (i.e., is of full rank). The controllability matrix is defined below:

$$W = \begin{bmatrix} B & AB & A^2B & ... & A^{n-1}B \end{bmatrix} \tag{3}$$

where n is the number of rows or columns in A. The matrix W was verified to be of full rank (using the NumPy method `numpy.linalg.matrix_rank()`), indicating that the system is controllable for the given value A and B. Verifying the controllability of the system is a significant step - a controllable system is one whose eigenvalues can be placed. It is thus possible to manipulate the rate at which the system reaches it equilibrium state.

C. K, the gain matrix

It was previously suggested that the input u could be written in form $u = -Kx$. The gain matrix K could be construed as a linear transformation that relates the input to state-space model. K answers the question of by how much the robot's internal state will change when an input is applied.

There are two ways to calculate K - pole placement and linear quadratic regulator (LQR). Pole placement can be done using the method `scipy.signal.place_poles()`. Pole placement is an effective method for finding the gain matrix K. It is possible to select 'very negative' eigenvalues that will cause the system to quickly reach asymptotic stability. However, the path the controller sets for the robot using pole placement may not be the optimum one. Pole placement ensures reaching the correct destination, but without regard for the best *path* to do so. LQR is a solution to this.

Linear Quadratic Regulator (LQR)

LQR is given by the following integral:

$$\int_{t0}^{tf} x^T Q x \;\; u^T R u \, dt \tag{4}$$

subject to:

$$x = Ax\left(t\right) \;\; Bu\left(t\right)$$
$$x\left(t0\right) = x_0$$

The matrices Q and R can be arbitrarily chosen:

$$Q^{6x6} = diag\left(q_i\right)$$

$$R = I^{2x2}$$

Using various values for the matrix Q, one can easily obtain different gain matrices K that yield different eigenvalues. The pipe line for solving for the gain matrices of the robot for specified Q are found through the method `scipy.linalg.solve_continuous_are()`, a numerical method to find the solution to the algebraic Ricatti equation. The following code block demonstrates this:

```
P = la.solve_continuous_are(A, B, Q, R)
K = np.linalg.inv(R) @ B.T @ P
```

This gain matrix is the most optimal ones for the values of Q and R provided. In order to tune to achieve the desired equilibrium states, the entries of Q may be adjusted relative to each other to result in different gain matrices K. These new K will apply a different weights to the state to achieve different inputs into the system, which will result in the desired behavior of the robot.

D. Verifying asymptotic stability

For asymptotic stability, the eigenvalues of the control system must possess negative real parts. Having obtained the gain matrix, it is easy to verify asymptotic stability:

```
is_True = np.zeros(len(K)) # 1 if negative re, 0 is non-negative re
dummy_var = []
for i in range(len(K)):
    dummy_var.append(la.eig(A - B@K[i])[0])
    if (np.all(np.real(dummy_var[i]) < 0)):
        is_True[i] = 1

if (np.all(is_True == 1)):
    print('Since all the eigenvalue real parts are negative, system is asym stable.')
```

This check was performed for all the matrices K that were checked, and they passed the test.

III. Methods

A. Requirements

In this project, success and/or failure will be measured by numerical benchmarks.

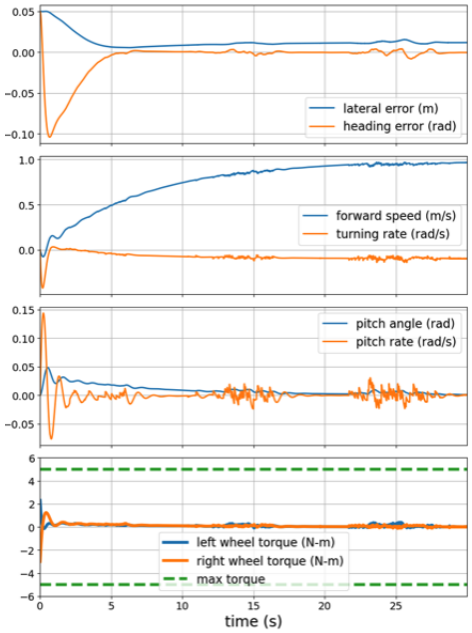
1) Heading error must be within $\pm$ 0.120 rad from the center line of a circular track.
2) Lateral error must be within $\pm$ 0.130 m from the center line of a circular track.
3) 1 and 2 must be satisfied for a minimum duration of 60s.
4) 1, 2, and 3 must be satisfied regardless of the initial condition.
5) If the robot topples over or falls at any point before the 60s mark, it is considered a failed attempt.

B. Verification

To verify the results, PyBullet will be used to simulate the robot. This will provide both a visual and numerical basis by which to verify the results produced in the Python notebook. At each time step, the lateral and heading error are calculated. Using a NumPy command np.all, all absolute values in the data set for lateral and heading error are compared to check that they are lower than the 0.13m and 0.12 rad respectively. Furthermore, sicne there are multiple gain matrices that are being generated, the comparison is made for each and every gain matrix.

IV. Results

The following are the results for equilibrium at $v_e = 0$. The initial conditions are $\theta_g = 0, v_i = 0., e_{li} = 0.05, e_{hi} = 0.05, \theta_i = 0$.



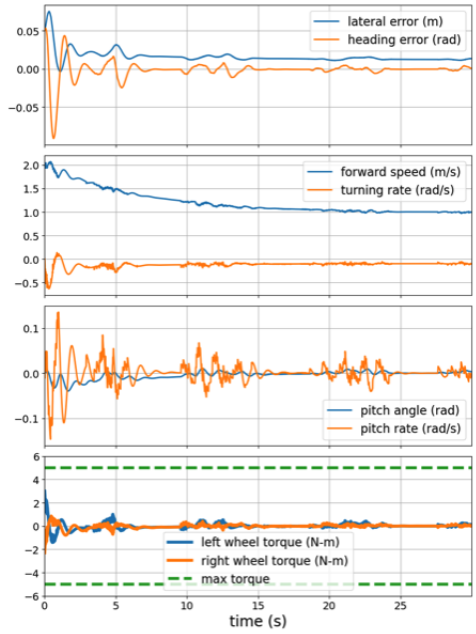
The results displayed in this graph match the expectations - all variables reached their equilibrium values after some time. There were mild fluctuations that can be seen in the graph for pitch angle, pitch rate, and turning rate. This could be the result of approximating a curved surface by a series of small linear corrective motions the robot has to make to stay on track. Furthermore, the lateral error and the heading error eventually converged to values within acceptable limits. However, the absolute value of the heading error first increased sharply. This may be due to the initial position of the robot which caused it to move linearly in a certain direction to decrease the lateral error but increasing the heading error. However, the robot took corrective measures to ensure that equilibrium will be achieved. Similar results were observed for all the other **valid** matrices (valid in this case referring to those gain matrices which were associated with acceptable error). Thus, the robot is seen to perform well as it reaches the equilibrium conditions quickly from the initial conditions.

V. Analysis

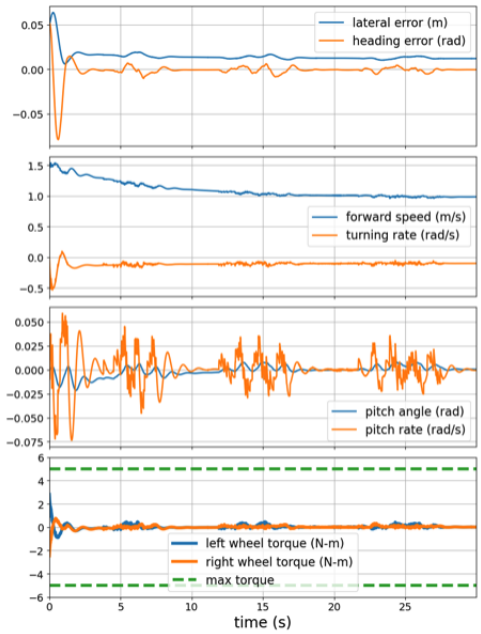
This section will explore changing certain values of initial and equilibrium coniditions and their effects on the motion.

A. Changing initial velocity

The graph on page 4 showed the effect when the body accelerated from rest to its eventual equilibrium position. Now, the initial velocity will be increased to 2m/s.



As before, the equilibrium velocity eventually approaches 1 m/s. However, in this case, there is higher initial fluctuation in the error, turning rate, pitch angle, and pitch rate. This is most certainly caused by the higher initial velocity. This is understandable however. At a higher initial velocity, the segbot is expected to increase its error drastically first as it moves away from the center line. This motion is caused by the turning rate and pitch rate. In order to move itself back within the acceptable limits, the controller sends signals to perform corrective maneuvers - increase pitch rate and reduce turning rate. This reduces lateral error, but increased heading error. To reduce heading error, the opposite is done. This sequence occurs repeatedly until equilibrium is reached, where minor changes occur.



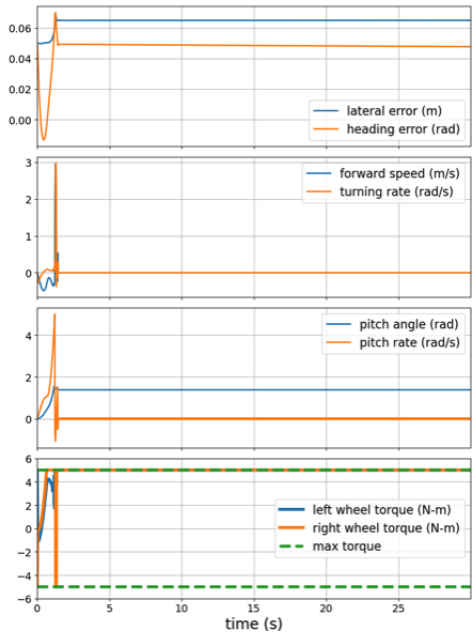
The above image is when the initial velocity is 1.5m/s. As it can be seen, the error is smaller, indicating better performance.

It depends on how high an initial velocity a robot possesses to determine the performance of the controller. If initial velocity is too far away from equilibrium velocity (>7 m/s difference), there is a chance for catastrophic performance if the controller is not able to send strong enough corrective signals to re-orient the robot, causing it to exceed the

error limit. If the initial velocity is not too far away from the equilibrium velocity (1-3 m/s difference), then a better performance can be expected from the controller.

B. Comparison of LQR and pole placement

The following graph is obtained through pole placement using the eigenvalues for the same gain matrix as the one for the graph in page 4.



It is seen from this image that there are very few oscillations observed for a controller using pole placement. Furthermore, zero error is reached much faster than in LQR. However, one major difference is that in pole placement, a constant maximum torque is required to keep the segbot moving along a circular track with 0 error. From the standpoint of error performance, the pole placement controller performs far better than the LQR controller. However, often times, path optimization requires to take into consideration energy expenditure. The LQR controller has a far lower energy expenditure as compared to the pole placement controller, which could make it more desirable in many situations.

VI. Conclusion

This differential drive bot is capable of achieving good performance while traveling along a circular path. It stays close to the center of the track and achieves minimal error. It is observed that the closer the system's initial conditions are to equilibrium conditions, the better it performs. It is also noted that LQR controllers and pole placement controllers have some trade-offs involved with them in terms of energy expenditure and error. However, the LQR controller seems to be the better controller when initial conditions are sufficiently close to equilibrium.