Balance Control of a Segway Robot using PI, PID, and Full State Feedback

Kaleb Blake
M1 -3
1 Introduction (20 Points)
A Segway is a two-wheeled motorized platform designed to stabilize a person standing on top of it. The rider can control where the vehicle moves through leaning slightly in a desired direction. The original Segway was invented by Dean Kamen in 2001, but now there are many different versions of it including the mini hoverboard Segway, and the one-wheeled Segway. The goal of this project was to make a construct and stabilize a model of a Segway robot using different control strategies. Specifically, the control strategies to be used were PD, PID, and Full State Feedback control. This report will discuss how the Segway robot was modeled, what the previously mentioned control strategies are and how they were used to stabilize the Segway model, results obtained from experiments with the modeled Segway, and the effectiveness of the different control methods and possible improvements to be made.

2 Modeling and Control Methods (10 Points)
The Segway robot was modeled as an inverted pendulum on a moving cart (see Fig. 1). All viscous frictional forces were ignored. This system is nonlinear and unstable about its upright position, which makes sense because the pendulum will easily fall due to gravity if it is slightly perturbed from a perfectly upright position. But the pendulum can be balanced by carefully moving the wheels on the cart and using the reaction force of the pendulum against the cart.

Figure 1: Inverted pendulum on a moving cart model

The linearized equations of motion of the system show in Fig. 1 are:

\[ \dot{\theta} = \frac{mgl(m+M)}{q} \theta - \frac{ml}{q} f \]  

\[ \dot{x} = \frac{-m^2 gl^2}{q} \theta + \frac{f + ml^2}{q} f \]

where
• $M = 0.088$ kg: mass of wheels
• $m = 0.680$ kg: mass of body
• $L = 0.15$ m: length of body from pivot point to center of gravity
• $q = J(M + m) + MmL^2$: common denominator
• $J = mL^2$: moment of inertia

You can use these linearized equations to get the following transfer function for the system:

$$\frac{\Theta(s)}{F(s)} = \frac{mL}{qs^2 - mgL(M + m)}$$

As I mentioned in the introduction the three control strategies used to stabilize the Segway robot were PD, PID, and Full State Feedback control. PD stands for Proportional and Derivative and it is a negative feedback control method, in which your compensator is of the form:

$$k_d(s + z) \text{ or } k_p + k_ds$$

where
- $z = \frac{k_p}{k_d}$: the zero on the left side of the imaginary axis
- $k_p$: proportional gain
- $k_d$: derivative gain

PD control can improve stability of a system and the speed of a system response, but steady state error may increase. PD controllers are also one of the easiest to implement. On the other hand, PID control is slightly harder to implement. PID stands for Proportional, Integral, and Derivative and it is also a negative feedback control method, though your compensator will have a different form:

$$\frac{k_d(s+z)^2}{s} \text{ or } k_p + k_ds + \frac{k_i}{s}$$

where
- $z = \frac{k_p}{2k_d} = \frac{k_i}{\sqrt{k_d}}$: the double zero on the left side of the imaginary axis
- $k_i$: integral gain

PID control can decrease steady state error to zero, though the stability of the system may worsen. With both PD and PID control you can manipulate the root locus of the system so that it bends through a point of interest and adjust the gains so that the system will have a pole at that point. Full state feedback gives much more flexibility in where you can place the poles of the closed-loop system because you can essentially place them anywhere in the s-plane and not just along a root locus. Full state feedback is a more modern technique because of this attribute. Unlike our PD, and PID controllers the full state feedback controller can account for damping in the Segway robot system. To understand full state feedback the state space model of the system must be observed (Fig. 2).
In general, a state space model is of the form:

\[
\begin{align*}
\dot{x}(t) &= Ax(t) + Bu(t) \\
y(t) &= Cx(t) + Du(t).
\end{align*}
\]

In our case \( A \) is a \( 4 \times 4 \) matrix, and the determinant of the matrix \((sI - A)\) gives a fourth-order polynomial in \( s \). The roots of the characteristic equation are the poles of the open-loop system and are given by:

\[
\det(sI - A) = (s - p_1^{OL}) (s - p_2^{OL}) (s - p_3^{OL}) (s - p_4^{OL}) = 0. \tag{8}
\]

To perform full state feedback control, we apply a proportional gain to each of the four state variables and sum them together to produce the controller command signal:

\[
u(t) = -k^T x(t) = -\left(k_1 \theta(t) + k_2 \dot{\theta}(t) + k_3 x(t) + k_4 \dot{x}(t)\right). \tag{9}
\]

Applying this control law to the state equation, we get:

\[
sX(s) = AX(s) + Bu(t) = AX(s) + B (-k^T X(s)) = (A - BK^T) X(s) \tag{10}
\]

The characteristic equation of this closed-loop system can be written as:

\[
\det(sI - (A - BK^T)) = (s - p_1^{CL}) (s - p_2^{CL}) (s - p_3^{CL}) (s - p_4^{CL}) = 0. \tag{11}
\]

As you can see if the closed loop pole locations are chosen based on a desired response, the required gain vector \( k \) can be computed.

### 3 Experiments and Observations (10 Points)

The Segway robot contained 4 key electrical components to allow for control of the Segway robot. (1) An Inertial Measurement Unit (IMU) was connected to the microcontroller, which collected all the tilt angle measurements. (2) An Arduino UNO microcontroller was attached to the Segway to read sensor data from the IMU, so the robot knows what angle it is at. Depending on the angle the robot is the Arduino UNO sent the appropriate commands to the (3) two 12V DC motors. (4) A dual channel motor shield (a 2A per channel source) was used to drive the two 12V DC motors.
In lab 7, PD and PID control was used on the robot. The given MATLAB file “segway_model.m” was used to implement the two control methods for the robot. For the PD controller design, first the stable pole of the transfer was found by setting the denominator of equation (3) to zero and solving for s. Then the zero of the PD compensator was set to double that pole. Next using the sisotool the poles of the closed loop system were changed until a damping ratio of 0.8 or less was achieved. The gains were kept below Kp = 80 and Kd = 8 because a high gain controller may have saturated the system frequently even when the error signal is small. The Kp and Kd gains were then plugged into the prewritten Arduino code. The behavior of the robot was observed and further optimized by adjusting the gains, trim_val (to adjust for forward/backwards bias), and bal_val (to make the robot move in a straight line). Though the idealized model shows long term stability of the system, the Segway robot may not achieve that long term stability, due to some combination of residual offset, steady-state error, and/or IMU drift in the system.

The steps for the PID controller design were the same as that of the PD controller design. This time, however, two complex zeros, and a real pole at zero were added to the compensator. The two complex zeros were at around -6 ± 1j. Then as in the PD controller design the gains were adjusted to meet the design requirement of a damping ratio of approximately 0.8 or less. The Kp and Kd gains had the same restrictions as in the PD design, and the Ki restriction was just that it should not be excessively large. The Kp, Kd, and Ki gains were then plugged into the prewritten Arduino code. The behavior of the robot was observed and further optimized by adjusting the gains, trim_val, and bal_val. The Segway robot achieved a much more stable and longer lasting stability under PID control, but it was much more difficult to adjust for the correct gains that would work for the robot in real life. Depending on the observed Segway behavior you would have to change the damping, the spring-like behavior, or decrease the range of oscillation by changing Kd, Kp, and Ki respectively. It was still difficult to achieve true long term stability due to the offset between motors.

In lab 8, full state feedback was used to control the robot. To implement the full state feedback control, the given MATLAB file “segway_state_feedback.m” was used. The poles for the full state feedback control were places in a “Butterworth” configuration (Fig. 3).
All the poles were to have a natural frequency (also equal to the crossover frequency) between 7 and 8 rad/s. To find the poles based on this requirement the following equation was used:

\[
s_k = \omega_c \exp\left(\frac{j(2k + n - 1)\pi}{2n}\right)
\]

where
- \(s_k\): a pole
- \(\omega_c\): crossover frequency
- \(k = 1, 2, 3, ..., n\)
- \(n\): order of the Butterworth filter

These poles were then plugged into the MATLAB file to compute the state-feedback gain matrix K. The values from this gain matrix were used in the given Arduino file. The behavior of the robot was observed and further optimized by adjusting the gains, trim_val, and bal_val. After fine tuning, the Segway robot exhibited long term stability with the Full State Feedback controller.

### 4 Results (40 Points)
Sensor readings with offsets: -7 4 16391 0 0 1
Your offsets: 1579 1358 1192 84 -49 9
PD:
- $K_p = 75.7191$
- $K_d = 4.9425$
- $trim\_val = -0.75$
- $bal\_val = 1.005$

Figure 4: Sisotool Graphs for PD Control

PID:
- $K_p = 60$
- $K_d = 4$
- $Ki = 485.39$
- $trim\_val = -0.75$
- $bal\_val = 1.005$
Figure 5: Sisotool Graphs for PID Control

Full State Feedback:
- $K_1 = 34.9816$
- $K_2 = 4.55$
- $K_3 = 42.1994$
- $K_4 = 3.8855$

Figure 6: Closed Loop Step Response of Full State Feedback Controller
5 Discussion (10 Points)
The best controller for the Segway robot was the full state feedback controller. It could follow the square wave, sine wave, circle, and figure 8 setpoints best. It also took much less tinkering with the gains to finally achieve stability. It displayed the ability to have long term stability as well. The PID controller was the next best controller. This controller took the most tinkering with gains to achieve the proper stability. When it did achieve its stability, it did a pretty good job of following the setpoints for the sine wave and square wave. Sometimes it did not evenly travel forward and backward (traveled more forward than backward for some periods and vice versa). Also, the transition between forward and backward was jittery for the square wave. It could balance for close to as long as the full state feedback controller too, but it had a little more
of a circular and irregular balancing pattern. The PD controller was the worst and could only perform stationary balancing. It could do stationary balancing well, though not as well or as long as the PID or full state feedback controllers.

Mismatch between the model and the actual system could be from the linearization of the inverted pendulum system. In the PD and PID control there could be mismatch from not considering the damping of the system. Not having the correct trim_val or bal_val to correct the robot’s tilt offset and torque difference between the motors will also cause mismatch.

If I were to work on the robot system more, I would work on the full state feedback controller because it had the best results. I would further tune the full state feedback controller by trying to minimize the tilt angle offset and torque difference between the motors. I would also try to adjust the K3 and K4 gains to try and minimize drift and oscillation. I may also add an integral control action to try and reduce the effect of steady-state error.