Ball on a plate controller using a MPU6050 sensor embedded inside the plate

nijatabbasov06

Jan 2025

Table of Contents

1 Abstract **2**

2 Introduction **2**

3 Theoretical background **2**

4 Experimental procedure **6**

5 Conclusion **6**

6 Reference **6**

1 Abstract

The ball on a plate control problem is a classic control system challenge where the goal is to balance a ball on a flat plate by tilting the plate in response to the ball's position. This control problem is the extension of the ball on a beam system where a ball undergoes a 2 dimensional motion on a beam to a 3 dimensional motion. For this reason, most of the equations used in this paper, will be similar to those in ball on a beam problem after doing some approximations.

There are some literature about developing PID controller utilizing touchscreen sensor embedded inside the plate [1], developing LQR controller using touchscreen sensor[2]. In this project, we will build a PID controller utilizing only MPU6050 sensor embedded inside the plate. Accessing the data from accelerometer and gyroscope inside the MPU6050, we will estimate the position of the ball, and build PID control algorithm to retrieve the ball to zero position. However, using only accelerometer or gyroscope data would cause the controller to underperform. That is why, we will use Kalman Filtering algorithm to fuse the data and have a more correct position of the ball.

2 Introduction

MPU6050 sensor provides us only with the data of the rotational orientation of the plate. For this reason, we need to build a mathematical model for the ball position to be able to estimate it. This means, we are unable to get the correct data for the ball position, and we are only left with open loop transfer function between sensor data and the ball position. Therefore, doing a lot of approximations would result in slightly incorrect model and this in turn, would result in error being accumulated over time. On the other hand, we also need to do some approximation for the computation to not be computationally extensive. The solution to deal with this type of tradeoff is to balance the approximations and experiment with different models.

One strategy we can utilize is to fuse the accelerometer and gyroscope data with Kalman filter to get relatively correct data for the ball position. The reason that this type of sensor fusion algorithm is important is because, when using only accelerometer data, we can see that, small perturbations to the plate would result in extremely fast oscillations in the derived euler angles of the plate. On the other hand, using gyroscope data, as we need to integrate the gyroscope data to get the euler angles of the plate, while we would not have the problem of high perturbations, the small errors in the gyroscope data would accumulate and result in a high offset between real euler angles and derived euler angles. To handle this problem, we would fuse those two derived euler angles to reach to the relatively correct euler angle data[3].

We will test our controller by using MATLAB Simulink. We will utilize MG996R servos , 5kg ball and a 30cm * 30cm plate. The servos will be regulated with angle inputs, instead of PWM pulses as MG996R servos have positional feedback with a P controller.

3 Theoretical background

Our implementation will be mostly similar with the paper in the 1st reference[1]. The biggest diference will be that instead of getting the ball position directly, we will get the value with Kalman Filtering, but our control algorithm will be similar.

To be able to create a mathematical model, we would make some assumptions and approximation about the system to make the model simpler and work-around non-linearity. While we could, utilize LQR, or State feedback algorithm to still be able to work with non - linearities, the main focus of this paper is to inspect if MPU6050 is enough to get correct estimation.

The followings are the assumptions we will use during our derivation. These are the same assumptions made in[1]:

1. The ball is rolling and not slipping on the platform. 2. No kinetic friction is considered. 3. The geometry of the ball is perfectly spherical and homogenous. 4. The ball has no translation upwards relative to the platform.

Furthermore, to represent the motion of the ball on the plate, we will switch to the body frame of the plate and represent the motion of the ball in that frame considering the fictituous forces such as coriolis force, azimuthal force, etc. We will consider a body frame with axes x_b, y_b residing inside the plate and z_b axis pointing downwoards. When there is no ball on the plate, the x_b and y_b axis is pointing towards the lab frame x and y axes.

The following is the equation relating the accelerations in the two frames:

$\mathbf{a} = \dot{\omega} \times \mathbf{r} + \omega \times (\omega \times \mathbf{r}) + 2\omega \times \mathbf{v_b} + \mathbf{a_b}$

Reader familiar with accelerating frames can see that the azimuthal fictituous force is not represented in the equations. The reason for this is that, resulting azimuthal fictituous force is directed perpendicular to the plate surface. That is why, we emitted that force from our calculation because we assume there is no motion upwards relative to the platform. Moreover, throughout the derivation of the mathematical model, we assume the angular acceleration of the plate is small anyways. Because, otherwise, we have no other choice but treat our system as a nonlinear system. To keep the system linear, we will regulate the plate with small angular velocities.

Moreover, in the above equation, a_b is the acceleration in the body frame, v_b is the velocity in body frame, r is the position of the ball in body frame and w is the angular acceleration of the plate relative to the lab frame.

While we could seperate the motion of the ball into two planes x-z plane and y-z planes and write down the equations for motion in each plane considering only component of the angular velocity perpendicular to that specific plane and position of the ball only inside that plane as in [1], we would not be able to prove it, because this method includes some approximations, and we need to adress those approximations. To see this, we could seperate vector r into x_b and y_b, vector w into w_y and w_x, in the body frame, and write down the above equation - acceleration relation between two frames again. Neglecting the acceleration vectors perpendicular to the plate, we would end up with acceleration vectors paralel to x_b and y_b axis. However, we would see terms such as $xw_x\vec{w_y}$, $yw_y\vec{w_x}$ which would not exist if we would seperate the motion in x-z plane and y-z plane. But considering the w_x and w_y is not that big, it is plausible for us to get rid of that terms. After seperating acceleration vector into x_b and y_b axis, we are left with the following equation for x_b axis:

$\mathbf{a_{x}}_b = \left(\ddot{x}_b - x_b\dot{\theta_y}^2\right)\mathbf{e}_{x_b}$

where $\mathbf{e}_{x_b}$ is the unit vector paralel to x_b. Note that, we would get the same equation for the y_b axis by only changing vectors in x_b axis to according vectors in y_b axis. This way, we can see that we have basically split our motion into two different beam systems.

Now we can see that, by neglecting that $xw_x\vec{w_y}$ terms, we could get rid of the coupling between motions in two different planes, thus, we can design a controller for the each axis seperately. However, we can see that acceleration of the ball in lab frame is non-linearly dependent from angular acceleration of the plate. Remembering w_x and w_y is again very small, we can get rid of the $x_b\dot{\theta_y}^2$ term in the above equation only to be left with $\mathbf{a_{x}}_b = \ddot{x}_b\mathbf{e}_{x_b}$. While we could just say this from the start, for completeness, I wanted to show the details in the approximation to be on the same page.

Now, we can say that ball's angular acceleration vector passing from its center is equal to $\ddot{\alpha} = \frac{\mathbf{F}r}{I_b}$ where F is the friction force and as we assumed that the ball is rolling without slipping, $\alpha r = a_b$ where a_b is the module of the acceleration vector in body frame. Because of the linearity, we can seperate the motion in x_b and y_b axis and assert: $F_x = -\frac{I_b \ddot{x}_b}{r^2}$ and the same thing for the y_b axis. After this, we can say $m\mathbf{a_{x}}_b = \mathbf{F_x} - \mathbf{mg}sin\theta_y$ and $m\mathbf{a_{y}}_b = \mathbf{F_y} + \mathbf{mg}sin\theta_x$ as a positive $theta_x$ angle means the gravity is pulling the ball downwoards. After inserting these inside the equation relating accelerations in two frames to get the following:

$\left(\frac{I_b}{r^2} + m\right)\ddot{x}_b + mg\sin\theta_y = 0$

$\left(\frac{I_b}{r^2} + m\right)\ddot{y}_b - mg\sin\theta_x = 0$

Approximating $sin\theta$ as θ, we get the followings:

$$\ddot{x} = -\frac{mg\theta_y r^2}{mr^2 + I_b}$$
$$\ddot{y} = \frac{mg\theta_x r^2}{mr^2 + I_b}$$

Assuming a solid sphere with inertia $I_b = \frac{5}{7}mr^2$

$$\ddot{x}_b = -\frac{5}{7}g\theta_y$$
$$\ddot{y}_b = \frac{5}{7}g\theta_x$$

Finally, we have the transfer function for the angle and position relation:

$$s^2X = -\frac{3}{5}g\Theta_y$$
$$s^2Y = \frac{3}{5}g\Theta_x$$

However, to be able to develop a control algorithm, it would be much easier to work with a new variable $\Theta_{y'} = -\Theta_y$

$$s^2X = \frac{3}{5}g\Theta_{y'}$$
$$s^2Y = \frac{3}{5}g\Theta_x$$

After this, we can design the following type of PID controller for our system:

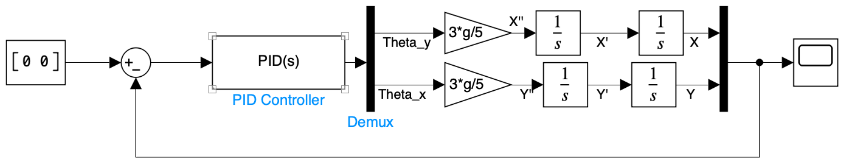


Figure 1: PID controller for a ball on a plate system

Note that, in the above picture, instead of using MPU6050 data, I am using X Y positions as feedbacks directly. To test how MPU6050 works with our controller, testing will be conducted in real life. However, before moving on to real devices, it would be good to test how our PID controller works for the hypothetical case when the x and y data used as feedback has no error and if our mathematical model is correct:

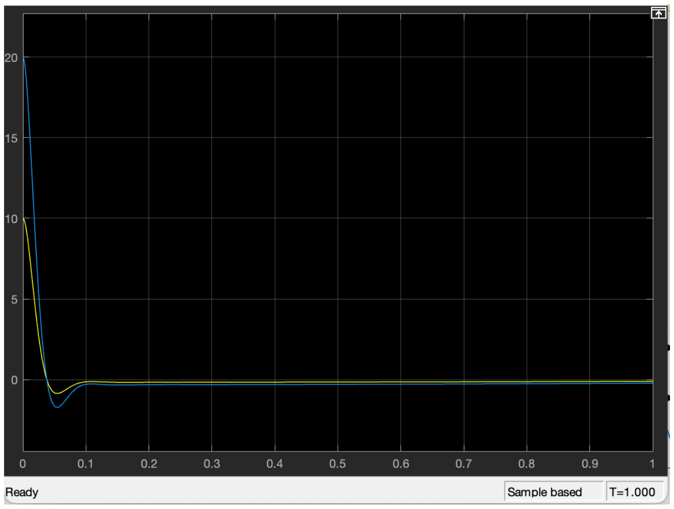


Figure 2: initial positions = [20 10]

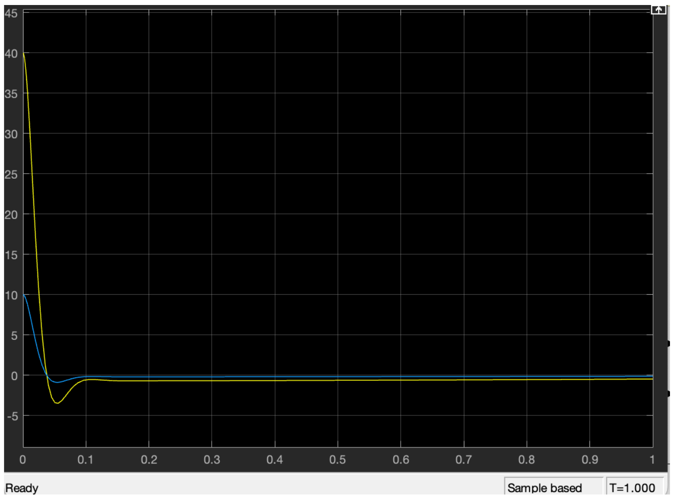
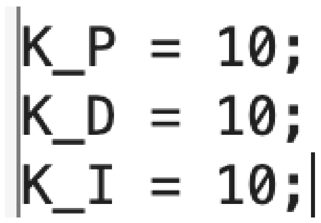


Figure 3: initial positions = [10 40]

For our PID controller, the following values were used as parameters, where K_D is derivative, K_I is integral and K_P is the proportional controller parameter:



```
K_P = 10;
K_D = 10;
K_I = 10;
```

Figure 4: PID controller parameters

One can play around with these values to achieve the desired performance of the PID controller.

4 Experimental procedure

5 Conclusion

6 Reference

[1] https://www.diva-portal.org/smash/get/diva2:1373784/FULLTEXT01.pdf
[2] https://icts2019.tve.gov.ly/2019/PDF/PDFEC/EC4019.pdf
[3] https://youtu.be/5HuN9iL-zxU?si=-qNEdVLx1I1aBRVn