**Documentation of Self balancing bot.**

**Topic Name:**

Self-Balancing Robot using MPU6050

**Name:** Daattavya Aggarwal, Anirudh Garg

**Year:** I

**Mentors:** Vishal Singh, Sunil Saini, Pankaj Singh

**Category:** Algorithm

**Sensor:** MPU6050 is a 6-dof, triple axis inertial measurement unit.

**Position of IMU**: The IMU should not be kept at height as the data of the accelerometer fluctuates over a very wide range. The instantaneous acceleration of the sensor will be r\*alpha where alpha is the angular acceleration of the bot. When the IMU is mounted at height, r increases and hence the accelerometer fluctuates over a greater range. If we mount the IMU at the base of the bot, then the accelerometer data fluctuates over a smaller range which is imperative for proper working of the robot. We used a Kalman filter to fuse the data of the accelerometer and gyrometer. Initially a complementary filter was applied but we experienced data lag. On increasing the weightage of the accelerometer, the data lag was reduced but the accuracy of the data was also affected. It is advisable to keep maximum possible weight of the bot along the axes of its wheels.

**Size of wheels**: The wheels used should be thick in size and large in diameter as it helps increase the recovery time for the bot. The distance between the wheels should be small so that the maximum possible weight of the bot lies near its axes which helps in balancing torque.

**Position of COM:** The centre of mass should be kept as low as possible as keeping it at a height will require more torque from the motors. However, if your motors are capable of producing large torque, then it is preferable to keep the COM at a larger height as it will improve the sensitivity of the bot. In general try to keep the COM low if you are having difficulty in balancing the bot as it will demand less of your motors.

We further applied a moving average filter to the final pitch given by our Kalman filter as our data was vibrating a great deal while the bot was in motion.

We used PID control to provide the appropriate pulse width modulation signals to the motor. Initially we used the gyrometer data directly for the derivative term in our PID control loop as the gyrometer data gives angular velocity which is theoretically the derivative of our error. However, we later realized that we require the average rate of change and not the instantaneous rate of change in the PID control loop. Hence we shifted to a traditional PID control loop.

**Note:** Sometimes our data would randomly freeze. On inspection we found that this happened due to certain while loops in wire.h library which would not time out and hence kept the scl pulse line low indefinitely. This stopped any further data communication. This problem is more prevalent in Arduino clones. We had to shift to an original Arduino( made in Italy) to solve this issue.