Course Project Documentation

CS308 Project

# *Two Wheel Self Balancing Bot*

TEAM: 8

Nitish Jhawar, 09005032
Vinayak Gagrani, 09005035
Tarique Aziz, 09005036
Vaibhav Krishan, 09005042

# Table of Contents

# 1. Introduction

Making a self-balancing robot is essentially solving the **classic inverted pendulum problem**. While the math is a bit complex, the essence is quite simple: the goal of the control loop is to adjust the wheels' position so that the inclination angle remains stable at a pre-determined value (e.g. the angle when the robot is balanced). When the robot starts to fall in one direction, the wheels should move in the falling direction to correct the inclination angle. And heuristics tell us that when the deviation from equilibrium is small, we should move "gently" and when the deviation is large we should move more quickly.

# 2. Problem Statement

The primary aim of the project is to construct and balance a two-wheel bot. The basic idea is explained in the previous section. The specific algorithm used to perform this task is called PID (**proportional–integral–derivative).** Secondary aim includes tuning it, for better performance. One of the steps is to filter the input (from accelerometer/IR-Sensor). This is done by Kalman Filtering.

# 3. Requirements

**Hardware Requirements**:

- Spark V5 sister board
- Two 300rpm DC motors
- 1-dimensional accelerometer
- Gyroscope (optional)
- Acrylic (to built the outer structure)
- Nuts/Bolts
- Acrylic Cutter
- Drill Machine
- Additional Hardware (like L-clamps etc.) may also be required

**Software Requirements**:

- AVR Studio 4

# 4. Implementation

**Construction of bot**:

We were initially provided by a completely built bot. But during tuning runs we found it to be inadequate (in terms of balance potential). So we started from scratch. We built a more stable and vertical bot (to enhance its balancing). This required a lot of handwork on our part (cutting the acrylic, drilling accurate holes in it, joining it to perfection etc.). The main point here was to make a sturdy bot that would easily sustain collisions and also balance itself in the future.

**Balancing the bot**:

The balancing stage began, after the initial stage of bot construction. The basic idea is:

- Take one/two feedback sensors (accelerometer/IR-sensor).
- A Tilt or Angle sensor to measure the tilt of the robot with respect to gravity.
- Wheel Encoders to measure the position of the base of the robot.
- Four terms are sufficient to define the motion and position of this bot and thereby by balance it (this is basically PID) :-
  - The tilt angle
  - Error from mean position (**Proportional term**)
  - Sum of previous errors (**Integral term**)
  - Difference from previous error (**Derivative term**)
- These four measurements are summed (after multiplying by suitable tuning constants $K_p$, $K_i$, $K_d$ that are found by very

specific tuning) and fed back to the platform as a motor voltage, which is proportional to torque, to balance and drive the robot.

# 5. Testing Strategy and Data

Our test criteria were pretty simple. We would simply record the time the bot would balance without collision. The higher the time the better the tuning.

**Performance**:-

1. The initial bot on significant tuning managed to balance for 2-3 sec at times.

2. The intermediate bot was also no better, it also managed to balance 3-4 sec, that too with Kalman filter.

3. Finally, the final bot showed some promise as it at times balanced for 6-7 seconds and we still have to tune the integral and differential parameters of PID.

# 6. Discussion of System

## A) What all worked as plan?

### a. PID Code:

The PID code which we had studied from one of our many references, worked fine (in my opinion). The implementation was quite simple, as far as the programming part was concerned.

## B) What we added more than discussed in SRS?

### a. Bot Design:

The initial bot design (the one discussed in the SRS), was faulty as far as balancing was concerned. Since it was flat (horizontal), it provided a very short balancing time. We altered the bot design, making it vertical and raising its center of mass, away from the axis of rotation. This means more time to balance the bot.

### b. Noise in the input (accelerometer):

During initial test runs, we found that the readings from the accelerometer were very noisy, mainly due to the movement by the motor. The values varied from (-30 to +30) on slight motion, which led to great difficulty. Read about Kalman filter (to filter out noise

from readings), but that required a gyroscope as well. But on further search we found a one dimensional and only accelerometer version that we then implemented. The parameters for Kalman filter were found out by hit and trial, but finally the readings were noise free.

# 7. Future Work

- There is lot of scope if the battery and hardware support from ERTS lab is enhanced. Lot of time was wasted which could have been used better.

- You can include Gyroscope for second source of error and thus enhance the Kalman filter implementation.

- Bot design can be more robust in a way to allow for more time for feedback to tuning software.

- Once the balancing at static position is achieved then you can add the movement features to it.

- Wi-Fi and other add-ons' are secondary once the bot is in movement conditions. It can be then easily used to replace even firebird.

# 8. Conclusion

To conclude, our final bot is still in its tuning stage. We still have to tune the integral constant ($K_i$) and the derivative constant ($K_d$). Anyone who studies a little bit on PID can easily finish/improve the existing project. Tuning the bot just requires a lot of time, but no major mental effort on your part. We have provided links to references that give certain methods for manual tuning of PID controllers.

# 9. References


- Self Balancing Bot Introduction (http://www.geology.smu.edu/~dpa-www/robo/nbot/) Last Viewed on 4 April 2012
- Sample Implementation Video (http://www.youtube.com/watch?gl=IN&v=iDAUMCVHV8g) Last Viewed on 6 April 2012
- Sample Implementation Video (http://www.youtube.com/watch?v=oxsCyere8hk) Last Viewed on 6 April 2012
- PID Algorithm (http://en.wikipedia.org/wiki/PID_controller) Last Viewed on 9 April 2012
- Kalman Filter (http://en.wikipedia.org/wiki/Kalman_filter) Last Viewed on 9 April 2012