SELF BALACING BOT

Summer project
completed as a part of
electronics club, IIT Kanpur

Project documentation

: By Vipul Gupta

Team members :

Vipul Gupta

Sonu Agarwal

Shubham Singh

Ritesh Kumar

Shubham Singh

Suyash Somvanshi

Vaibhav Gupta

Team Mentors :

Rohinish Gupta

Nitish Shrivastav

Contents :

BASIC AIM :

- ⊡ To demonstrate the techniques involved in balancing an unstable robotic platform on two wheels.
- ⊡ To design a complete discrete digital control system that will provide the needed stability.

- To complete the basic signal processing which will be required in making a unicycle.

How was it done :

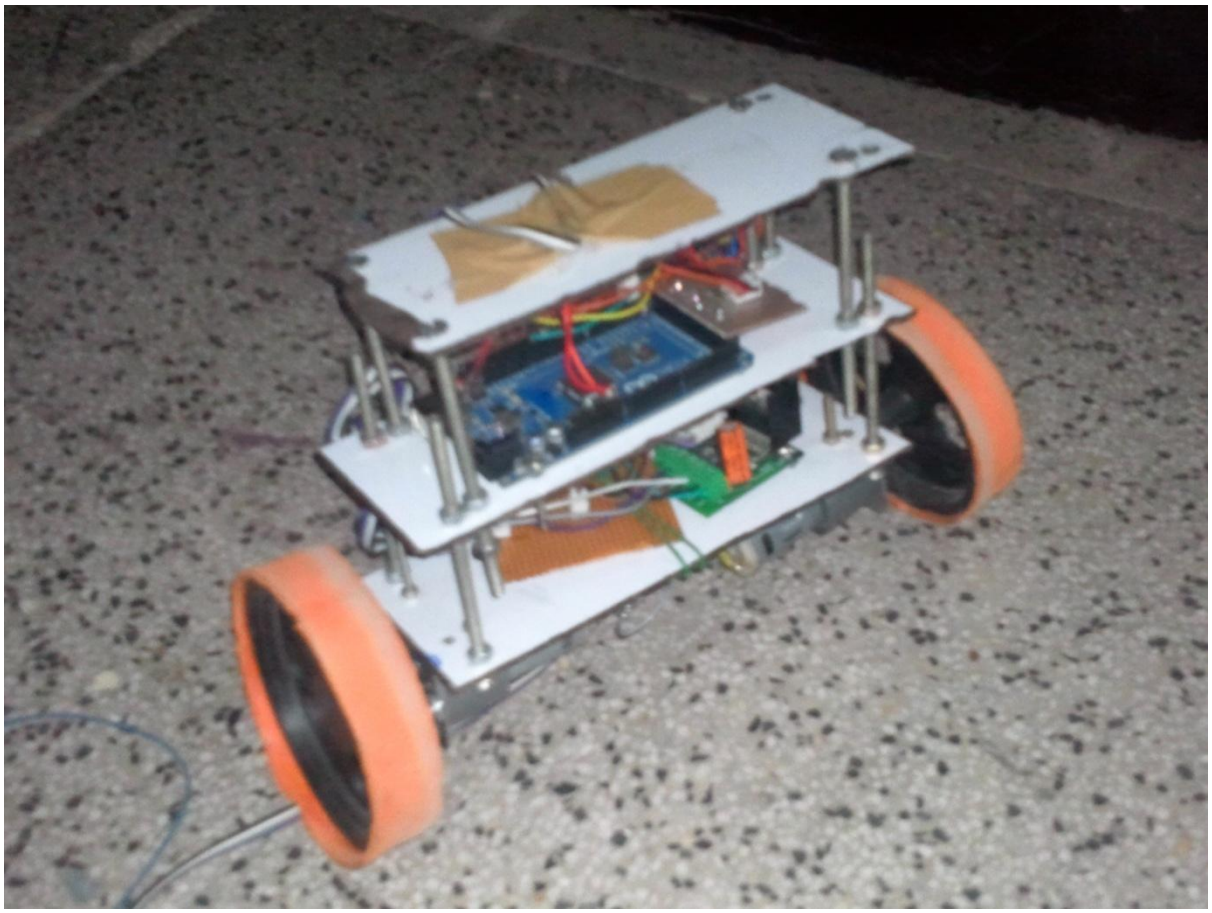
It will be prevented from falling by giving acceleration to the wheels according to its inclination from the vertical. If the bot gets tilts by an angle, than in the frame of the wheels, the centre of mass of the bot will experience a pseudo force which will apply a torque opposite to the direction of tilt.

How do we get the idea:

Keeping in mind of our ultimate aim, the unicycle, we had to complete the basics in summer and need to understand what efforts would it require. Also at the very first we thought of making a self balancing platform, which used data from accelerometer and gyroscope. A self balancing bot is an advanced version of this platform. Self balancing bot includes the basic signal processing part of the unicycle.

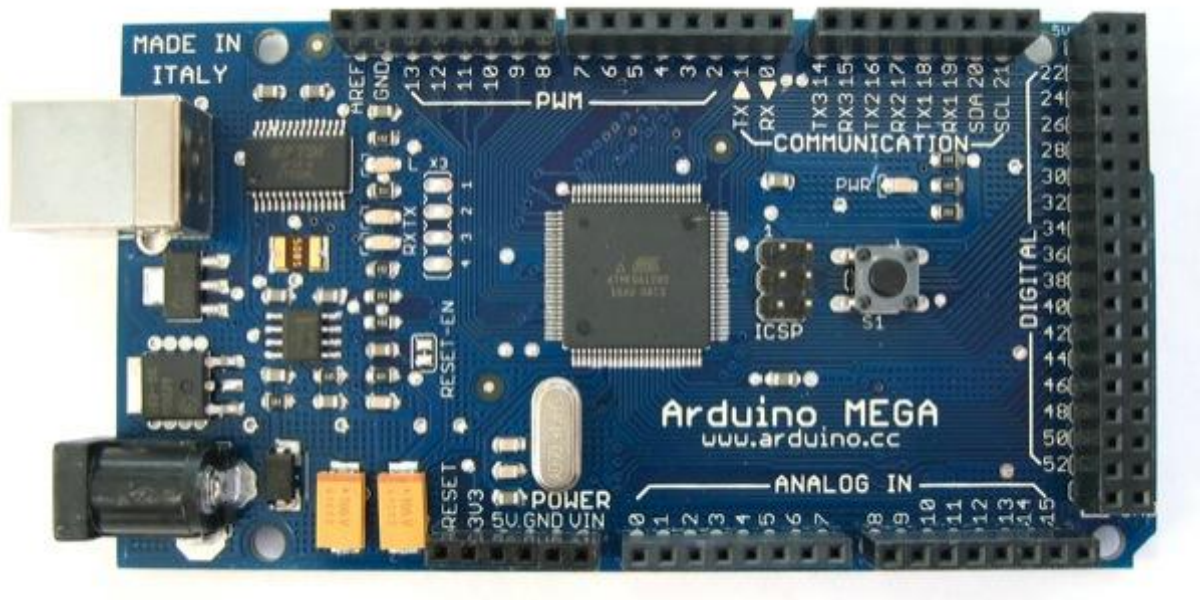
Mechanical Structure:

The bot consists of three platforms which have arduino, IMU, motor driver mounted on it. On the lower part of the base platform, the two high torque motors (300 rpm) are clamped. The whole bot gets balanced on two wheels having the required grip providing sufficient friction (as there are large chances for wheels to skid). Here is a pic of the bot :



Arduino Mega 2560 :

The Arduino Mega is a microcontroller board based on the ATmega1280. It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.



The power pins are as follows:

- **VIN.** The input voltage (7-12 volts) to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3V3.** A 3.3 volt supply generated by the on-board FTDI chip. Maximum current draw is 50 mA.
- **GND.** Ground pins.

6 DOF IMU (Inertial Measurement Unit)

Features:

- Tiny!
- Two mounting holes
- ADXL345 accelerometer
- ITG-3200 gyro
- 3.3V input
- I2C interface

⊡ Accelerometer – Gives the components of acceleration(g) along its three axis.

- More sensitive and noisy data.

⊡Gyroscope- Gives the components of angular velocity along its three axis.

-Less sensitive but its output gets DRIFTED along with the time.

Description: This is a simple breakout for the ADXL345 accelerometer and the ITG-3200 gyro. With this board, you get a full 6 degrees of freedom. The sensors communicate over I2C and one INT output pin from each sensor is broken out.

Hooking it up

The both components of the Digital IMU 6DOF are I2C devices on a single bus. I2C is a 2-wire serial connection, so we just need to connect the SDA (Data) and SCL (Clock) lines to your Arduino for communication. On our Arduino (arduino mega), SDA is digital 20, and SCL is digital 21. . Other than these 2 lines, we just need to connect power(3.3v), ground and we are all set.

I2C Interface:

Inter-Integrated Circuit (I²C pronounced I- Squared- C) is a 2 wire serial bus typically used to communicate with sensors and other small components.

The two lines of the I2C bus are SDA (Data) and SLC (clock) which can be run in parallel to communicate with several devices at once. I2C allows up to 112 "slave" (such as a sensor) devices to be controlled by a single "master" (such as Arduino, in our case). Each slave device on the bus must have it's own unique address so the master can communicate directly with the intended device. These addresses are typically hard-coded into the slave device, but often allow it to be changed by simply pulling one of the pins of the sensor high or low. This allows more than one of the same device to be on the same bus without conflicting addresses.

I2C is often referred to as TWI or 2-wire-serial.

I2C Tutorial : I2C basic command sequence.

- 1. Send the START bit (S).
- 2. Send the slave address (ADDR).
- 3. Send the Read(R)-1 / Write(W)-0 bit.
- 4. Wait for/Send an acknowledge bit (A).
- 5. Send/Receive the data byte (8 bits) (DATA).

- 6. Expect/Send acknowledge bit (A).
- 7. Send the STOP bit (P).

Kalman Filter :

The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) means to estimate the state of a process, in a way that minimizes the mean of the squared error. The filter is very powerful in several aspects: it supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modelled system is unknown.

The filter is used in control engineering to remove measurement noise that can affect the performance of system under control. It also provides an estimate of the current state of the process or system. The Kalman filter is a tool that can estimate the variables of a wide range of processes. In mathematical terms we would say that a Kalman filter estimates the states of a linear system. The Kalman filter not only works well in practice, but it is theoretically attractive because it can be shown that of all possible filters, it is the one that minimizes the variance of the estimation error.

The process or system that is being measured must be modelled by linear system. A linear system can best be described by the following two state space representation equations :

$x(k) = Bu(k-1) + w(k-1)$
$z(k) = Hx(k) + v(k)$

First equation represents the process state equation. *A, B*, and *H* represent the state matrices, the *w(k*-1) represents the process noise, *k* is the time index, x(k-1) is the state of the process, and *u(k-1)* is the known input to the process. Second equation above represents the output state equation. The *z(k)* represents the measured process output and the *v(k)* represents the measurement noise. The sensor data for the system are taken at discrete time sample points. The measurements are dependent on the state of the process or system.

Finally what we conclude about Kalman Filter is:

- ⊡It is being used for the fusion of the outputs of the two sensors used.
- ⊡Will reduce the noise of the two sensors by taking appropriate weightage which will vary according to time.

- Estimate the optimal state using the prediction from the last stage and current state measurement.

Why is Kalman Filter necessary :

- Based on recursive process so consume less memory of processor.
- Gives theoretically and practically accurate data.
- It is based on weighted average giving more weightage to the reading which is more certain.

PID Control System:

The control algorithm that was used to maintain it balance on the autonomous self-balancing two wheel robot was the PID controller. The proportional, integral, and derivative (PID) controller, is well known as a three term controller.

The input to the controller is the *error* from the system. The Kp, Ki, and Kd are referred as the proportional, integral, and derivative constants (the three terms get multiplied by these constants respectively). The closed loop control system is also

referred to as a negative feedback system. The basic idea of a negative feedback system is that it measures the process output *y* from a sensor. The measured process output gets subtracted from the reference *set-point* value to produce an *error*. The error is then fed into the PID controller, where the error gets managed in three ways. The error will be used on the PID controller to execute the proportional term, integral term for reduction of steady state errors, and the derivative term to handle overshoots. After the PID algorithm processes the error, the controller produces a control signal *u*. The PID control signal then gets fed into the process under control.

The process under PID control is the two wheeled robot. The PID control signal will try to drive the process to the desired reference setpoint value. In the case of the two wheel robot, the desired set-point value is the zero degree vertical position. The PID control algorithm can be modelled in a mathematical representation.

PID is used to calculate the 'correction' term :

$$Correction = Kp*error + Ki* \int error + Kd* d/dt(error);$$

Kp , Ki and Kd are constants which are set experimentally.

If only the first term had been used to calculate the correction, the robot would have reacted in the same way as in the classical line following algorithm. The second term forces the robot to move towards the mean position faster. The third term resists sudden change in deviation.

The integral term is simply the summation of all previous deviations. Call this integral- 'totalerror'. The derivative is the difference between the current deviation and the previous deviation. Following is the code for evaluating the correction.
These lines should run in each iteration :

```
correction = Kp*deviation + Ki*totalerror +
Kd*(deviation - previousdeviation);

totalerror += correction;

previousdeviation = deviation;
```

PID Tuning :

Here are some steps which will help to get Kp, Ki and Kd faster :

- Set *I* and *D* term to 0, and adjust *P* so that the robot starts to oscillate (move back and forth) about the balance position. *P* should be large enough for the robot to move but not too large otherwise the movement would not be smooth.
- With *P* set, increase *I* so that the robot accelerates faster when off balance. With *P* and *I* properly tuned, the robot should be able to self-balance for at least a few seconds.
- Finally, increase *D* so that the robot would move about its balanced position more gentle, and there shouldn't be any significant overshoots.

A word of thanks

We would like to thank our team mentors Rohinish Gupta, Nitish Shrivastav and our club coordinators Anurag Dwivedi, Rudra Pratap Suman and Nikhil Gupta for guiding us to complete our project in time and providing a chill learning experience.