e-Yantra Robotics Competition

eYRC-BB#2403

Team Leader Name	**Heethesh Vhavle**
College	**B.M.S. College of Engineering**
E-Mail	**heetheshvn@yahoo.com**
Date	**16-01-2017**

Think and Answer

****

Instructions:

- Maximum **40 marks** will be awarded
- There are no negative marks
- **Unnecessary explanation** will lead to less marks even if answer is correct
- Use the same **font and font size** for writing your answer

****

Q1:

Describe hardware design of the Balance Bot your team has constructed.

(4 + 3 + 3 marks)

Answer:

The design of our Balance Bot is very similar to what was proposed in Task 2 with a few minor changes. The goal of our design was to keep the frame as light as possible in weight. The design consists of 3 tiers of Acrylic Board which were Laser Cut to create precise cavities to mount the components and to hold the support frame. The plastic support frames are 3D Printed. The lower part (underneath) of the chassis consists of the DC Motors and the GY-80 IMU sensor, which is placed at the center. Above the first tier, the motor driver and the battery is placed. The second tier consists of the Development Board. Although the top-most tier is quite empty, we found that the height of this particular design, helped us in achieving the best stability for our robot. The acrylic boards were designed to accommodate various modifications in the design and to mount other components in the future.

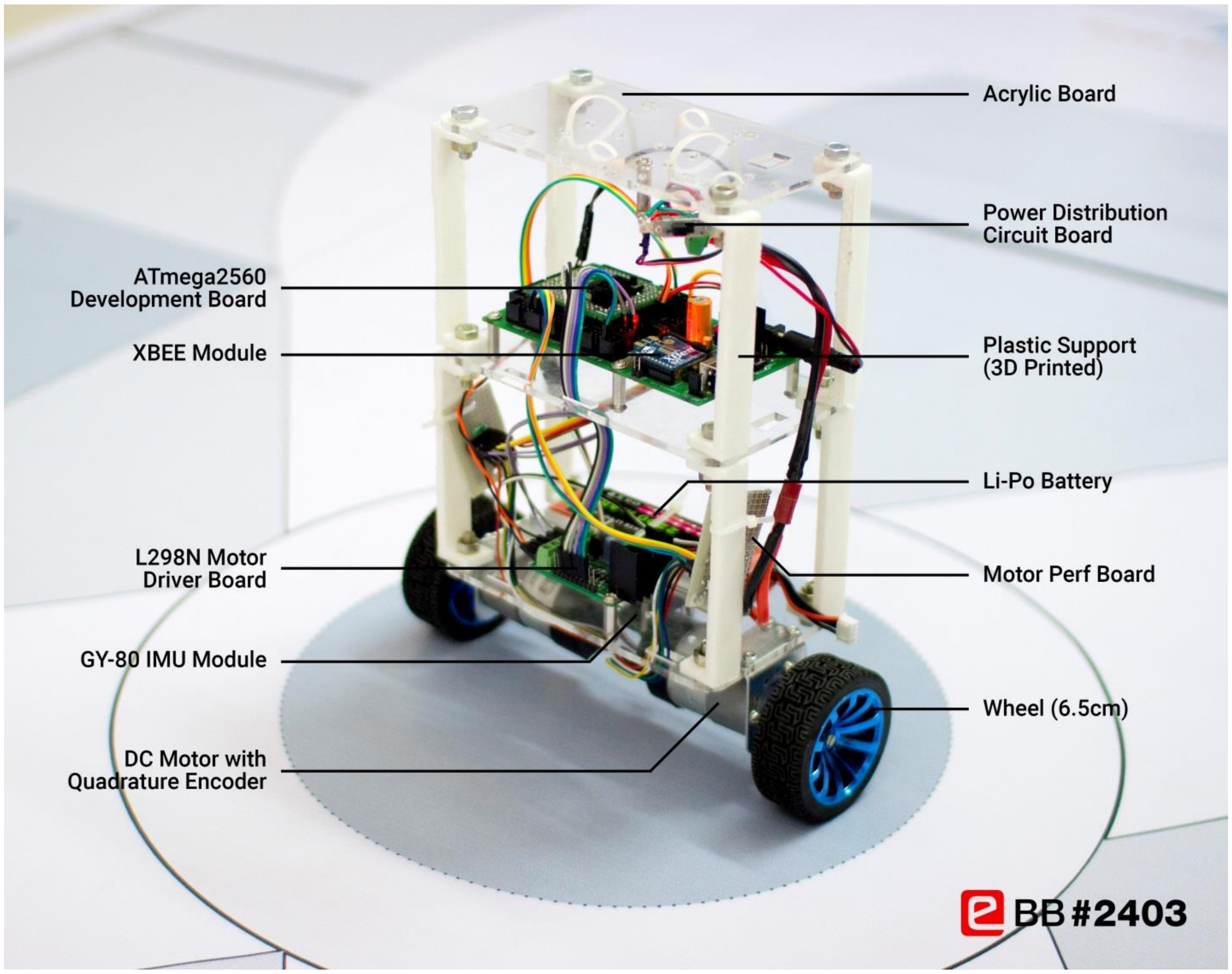

Acrylic Board
Power Distribution Circuit Board
ATmega2560 Development Board
XBEE Module
Plastic Support (3D Printed)
Li-Po Battery
L298N Motor Driver Board
Motor Perf Board
GY-80 IMU Module
Wheel (6.5cm)
DC Motor with Quadrature Encoder
BB #2403

Figure 1.1: Labeled diagram of the Balance Bot showing the placement of the various components.

Reasons for Selection of Design

1. The body frame was made as light as possible in order to decrease the moment of inertia and hence, the torque acting on the body due to gravitational force. Consequently, this gives the motors more time to respond quickly to a disturbance applied and also increases the maximum recoverable angle.
2. Although the top tier is mostly empty and could be removed to reduce the weight further, it is still kept in order to increase the height of the robot, which decreases the angular acceleration. This also allows for future modifications in the design.
3. The IMU sensor is placed at the bottom most tier in order to avoid false readings from the accelerometer due to linear motion.
4. The battery is placed at the bottom most tier to lower the CG slightly in order to achieve better stability and response of the robot. The final design is a trade-off between slower angular acceleration and maximum recoverable angle of the bot.

Challenges faced in Designing

1. Initially, the support frame consisted of threaded metal rods, as proposed in Task 2. However, this increased the weight of the robot, which effectively decreased the maximum recoverable angle.
2. A similar problem was with the position of the battery. The battery was placed at the top-most tier in order to achieve a higher CG. However, after a certain angle, even though the motors were at their full speed, the bot was unable to stabilize. Placing the battery down provided us a better range of angle from which the robot could recover from disturbances.
3. Another challenge faced was, after a certain angle, the robot used to constantly accelerate and finally fall down. Velocity feedback using encoders was implemented, to change the angle set point in the opposite direction, to prevent this.

Q2:

What is the effect of Proportional, Integral and Differential terms in PID control algorithm?

(3 + 3 + 3 marks)

Answer:

Proportional Term
The response of the proportional term is simply a constant (Kp) times the current error. Error is the difference between the desired position (Set Point) and the current position (Tilt Angle). Higher the magnitude of the error, higher will be its response in the overall PID output in order to minimize the error in the current position. A very low gain (Kp) will make the robot very unresponsive to disturbances. Whereas, a high gain will make the robot over responsive. The controller will not only correct the present error but will also introduce its own error, and in turn tries the correct this self-induced error again. Hence, it induces oscillations about the desired position. As oscillations become larger and larger, the robot will became unstable and finally fall down. It is evident that a steady state error always persists. Response of the proportional term is given as:

```
proportional_term = kp * error
```

Integral Term
The integral term sums up the amount of error over time. This term depends on the present magnitude of error as well as the past errors. The integral term helps in accelerating the current position towards the desired set point. This term will continually contribute to the overall response until the error becomes zero. It tries to eliminate any residual error (steady state) that should have been previously corrected. Higher the magnitude of the error, faster will be the response to minimize it. However, since it keeps accumulating errors over time and accelerates towards the set point, an overshoot will occur before it reaches the desired set point. Another problem is the wind-up phenomenon, which occurs when the accumulated error is beyond the maximum actuator output and can be solved by limiting its response in the output. The response of the integral term is given as follows:

```
    integral_sum += (error * sampleTime)
integral_term = ki * constrain(integral_sum)
```

Derivative Term
The derivative term is proportional to rate of change of error. It is not dependent on the current magnitude of error and is incapable of minimizing the error on its own. The purpose of the derivative term is to anticipate the future behavior of the error. Thus, it helps in achieving the desired position much faster and reduces the unwanted overshoot caused by the integral term. However, a high magnitude of derivative gain (Kd) can make the robot very jittery. Derivate response is given by:

```
derivative_term = kd * (current_error - previuos_error)/sampleTime
```

Finally, the desired output response can be achieved by tuning these PID gains properly to control their individual contributions in the output. The output of the PID control algorithm is given as:

```
PID_output = proportional_term + integral_term + derivative_term
```

Q3:

For balancing robot at one place, two parameters have to be controlled. One is tilt angle and another is position. How will you control the tilt angle and position of Balance Bot? Explain with functional Block Diagrams. (4 + 4 marks)

Answer:

Tilt Angle Control
The tilt angle of the balance bot is measured using the accelerometer and the gyroscope fused using a complimentary filter. To control the tilt angle, the first step is to determine the static angle set point or the angle at which the robot perfectly balance about its CG when static. Then the actual tilt angle is measured using the GY-80 sensor. A PID controller is designed to minimize the error between current tilt angle and the desired set point. The output of the PID controller is directly used to set the PWM of the motors to control its speed and the direction (forward or backward) based on the direction of tilt. This feedback system works well to keep the robot upright but the robot drifts in position over time.

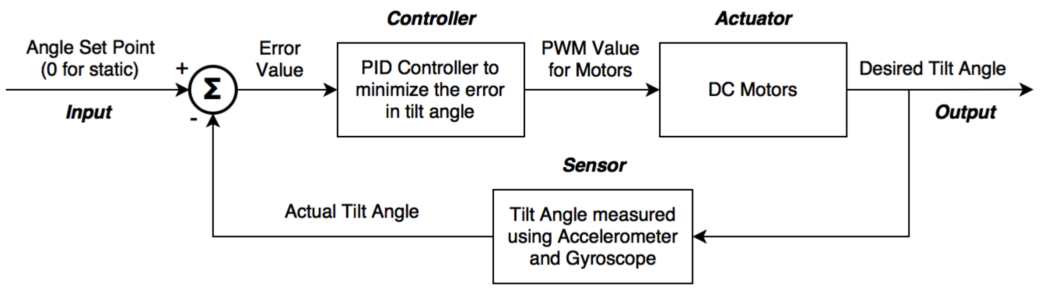

Controller
Actuator
Angle Set Point (0 for static)
Input
+
-
Σ
Error Value
PID Controller to minimize the error in tilt angle
PWM Value for Motors
DC Motors
Desired Tilt Angle
Output
Sensor
Actual Tilt Angle
Tilt Angle measured using Accelerometer and Gyroscope

Figure 3.1: Block diagram representing tilt angle control.

Position Control

To avoid the drifting of the robot when balancing, the position of the robot needs to be controlled. Position control involves linear position as well as rotational position. For a static robot, the position set point will be zero. The current position of the robot is measured in terms of the counts provided by the quadrature encoders. Another PID controller is designed which minimizes the error between the current position and the desired position (zero for static). The output of this PID controller changes the *Angle Set Point*. Initially consider the angle set point to be zero. If the robot moves linearly for a given distance, the position PID controller will change the angle set point in the opposite direction (say +5 degrees). This in turn makes the robot now to balance itself at +5 degrees instead of 0 degrees (upright). Thus, the angle PID controller will make the robot move in a direction opposite to the direction in which the robot had moved initially. Thus, the position PID controller indirectly controls the speed of the motors. As the position error approaches to zero, the angle set point also approaches to zero and thus the motor will automatically slow down and position error is corrected.

Therefore using a cascaded PID controller, the tilt angle as well as the position of the robot is maintained to balance it at one place. The inner PID loop controls the tilt angle and the outer PID loop controls the position of the robot.

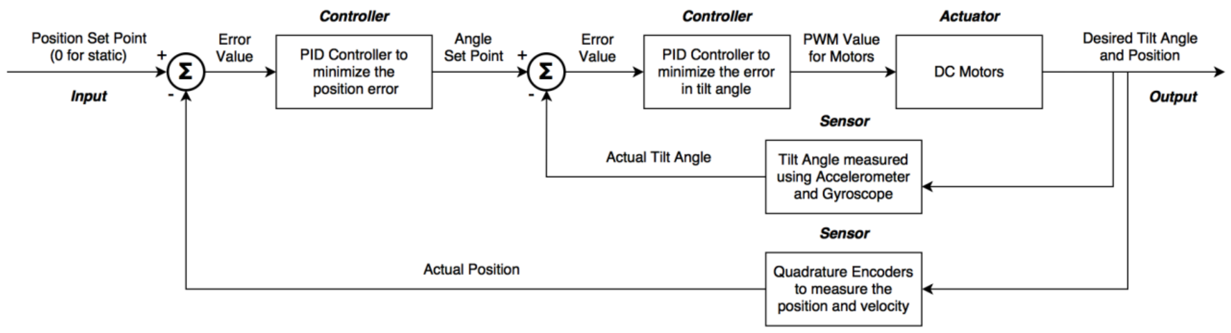

Controller
Controller
Actuator
Position Set Point (0 for static)
Input
Error Value
PID Controller to minimize the position error
Angle Set Point
Error Value
PID Controller to minimize the error in tilt angle
PWM Value for Motors
DC Motors
Desired Tilt Angle and Position
Output
Sensor
Actual Tilt Angle
Tilt Angle measured using Accelerometer and Gyroscope
Sensor
Actual Position
Quadrature Encoders to measure the position and velocity

Figure 3.2: Block diagram representing the cascaded PID control for tilt angle and position.

NOTE: Due to non-identical properties of the two DC motors, a rotation offset is also introduced. To correct this offset, another proportional controller, whose output is proportional to the difference in the left and right encoder position, can be implemented in parallel, which directly tries to nullify this offset in the PWM control of the motors. For the sake of simplicity, this is not shown in the block diagram.

Q4:
How to provide motion commands so that the Balance Bot could move in linear or circular motion?

(3 + 3 marks)

Answer:

Motion control of the robot is slightly different from static control. In order to move the robot a constant linear velocity has to be maintained. Therefore, in addition to position control, velocity control also has to be introduced in a parallel block.

First, the velocity set point has to be changed. To move the robot, a non-zero value should be given to the velocity set point. Suppose a forward motion is desired, let the velocity set point be some positive non-zero value. The velocity PID controller will try reducing the error from the current zero velocity to the new set point, thus changing its output, i.e., the angle set point. Let the changed angle set point be +5 degrees. Therefore, the robot will try to now balance at this new angle. In order to do this, the robot has to move forward to reduce the tilt angle error and hence linear motion is achieved.

To achieve circular motion, an offset must be applied directly to the PWM values of the motors. To rotate the robot around a single point, the offset should be such that it is positive for one motor and negative for the other motor.

To make the robot move in circular paths a combination of both linear and rotational motion is used where velocity set point is changed and an offset is also applied to the PWM of the required motor.

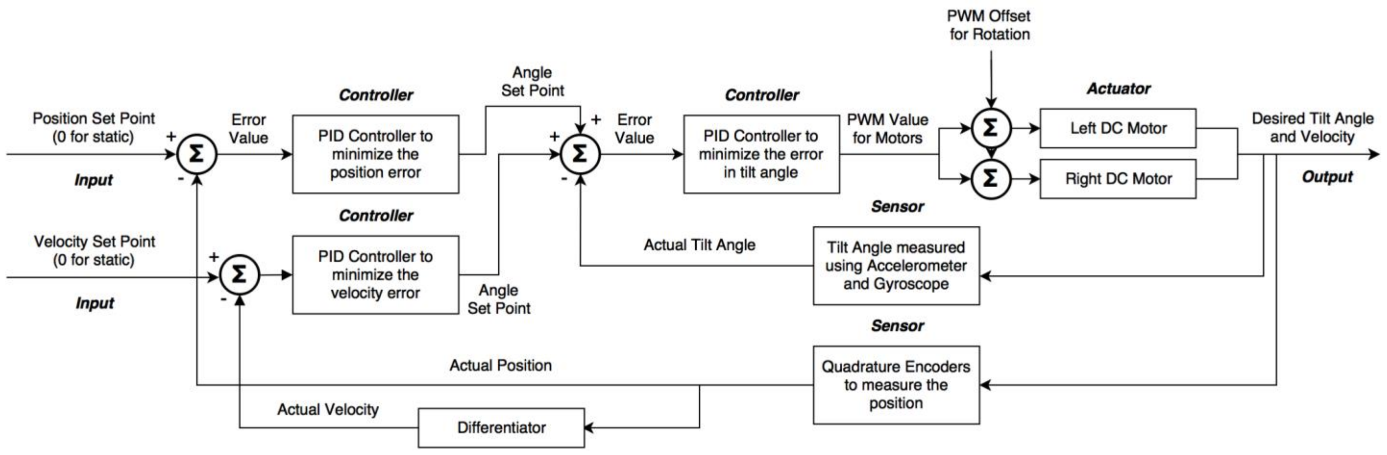

PWM Offset for Rotation
Position Set Point (0 for static)
Input
Error Value
Controller
PID Controller to minimize the position error
Angle Set Point
Error Value
Controller
PID Controller to minimize the error in tilt angle
PWM Value for Motors
Actuator
Left DC Motor
Right DC Motor
Desired Tilt Angle and Velocity
Output
Velocity Set Point (0 for static)
Input
Controller
PID Controller to minimize the velocity error
Angle Set Point
Actual Tilt Angle
Sensor
Tilt Angle measured using Accelerometer and Gyroscope
Sensor
Quadrature Encoders to measure the position
Actual Position
Actual Velocity
Differentiator

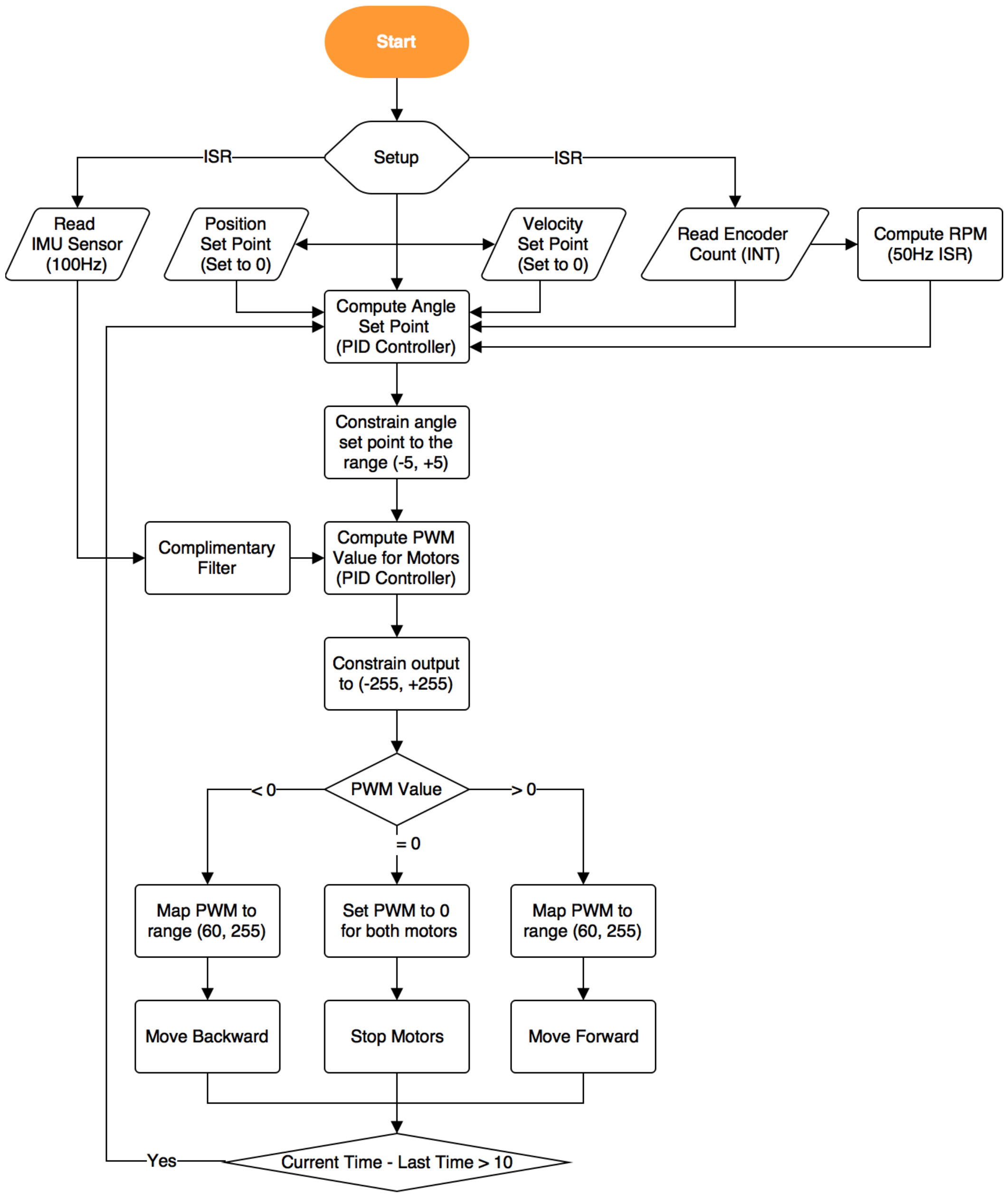
Figure 4.1: Block diagram representing the cascaded PID control for tilt angle, position and velocity.

Q5:

Describe the flow of your code for the Task 3-Think and Do. Use flow charts and state charts to describe the flow of code.

(2 + 1 + 4 marks)

Answer:



Start
Setup
ISR
ISR
Read IMU Sensor (100Hz)
Position Set Point (Set to 0)
Velocity Set Point (Set to 0)
Read Encoder Count (INT)
Compute RPM (50Hz ISR)
Compute Angle Set Point (PID Controller)
Constrain angle set point to the range (-5, +5)
Complimentary Filter
Compute PWM Value for Motors (PID Controller)
Constrain output to (-255, +255)
PWM Value
< 0
= 0
> 0
Map PWM to range (60, 255)
Set PWM to 0 for both motors
Map PWM to range (60, 255)
Move Backward
Stop Motors
Move Forward
Yes
Current Time - Last Time > 10

NOTE: *The flowchart shown is a brief representation of the program. Not all details are represented. The XCOS communication is not shown, as it is not used while balancing the robot. If required this XCOS task can be scheduled for every 100ms along with the other scheduled tasks such as PID computation.*

The tilt angle from the IMU sensor is computed in parallel every 10ms with the help Timer 3 ISR. The encoders' channel A is connected to interrupt pins and encoder count is incremented/decremented based on the state of channel B inputs. Timer 1 ISR is used to compute the RPM of the motors every 50ms. PID computation task is scheduled for every for 10ms.

First, the encoder PID output is computed to determine the angle set point. Process variables of this PID loop are the position and velocity (RPM of motors) of the robot. Velocity is measured in order to prevent the robot from accelerating too much and falling down.

Once the angle set point is computed, the angle PID loop is run to compute the PWM values for the motors. This output is limited to the range (-255, 255). If the output is positive, the motors are set to rotate in the forward direction and for negative output, motors are set to rotate in the backward direction in order to balance the robot as required. The PWM values for the motors are constrained to the range (60, 255) so as to not operate in the dead zone of the motors. Finally, the entire process in run in a while loop.