Group 13: Self Balancing Robot Documentation

Comentale Alfonso 0622701438
Orazzo Benito 0622701498
Parlato Ettore 0622701497
Sabatino Gerardo 0622701499

Introduction

Self-balancing robot is a two-wheeled robot which balances itself so that it prevents itself from falling. This concept is somewhat similar to the operation of a unicycle, the rider of the unicycle balances by moving himself in the same direction of the inclination so that he stays vertical, similarly the self-balancing robot balances by moving in the same direction of the inclination.
A more complete model of this problem is represented by the theory of the Inverted Pendulum.The inverted pendulum is a common problem to solve in control theory. The pendulum is usually mounted on a cart and is balanced by controlling the movement of the cart. This setup can be varied in many ways to make the system more complex and it can be controlled in many ways. For example the company Segway introduced their solution of a transport vehicle in 2001, the Segway Personal Transporter, which is based on the same dynamics as the inverted pendulum. It has two wheels with a platform between them where the passenger stands. The passenger then acts as the pendulum to be balanced and the passenger runs it by leaning forward or backward. Since Segway introduced it’s solution the area has gained more momentum both through public interest and in the transportation field.

Purpose and Goals

The main purpose was to design and construct a robot which will balance on two wheels and be able to work on its own without any supervisor. The provided precise goals for the robot are as follows:

- Be able to balance on two wheels
- Be resistant to small bumps or pushes
- Be able to balance on a 5% surface slope
- Should also be controlled by a wired joystick that make him move forward and backward

To achieve all the goals a proper structure must be built and the same should be done for the software part , in order to satisfy some performance requirements such as stability , resistance to interference and autonomy.

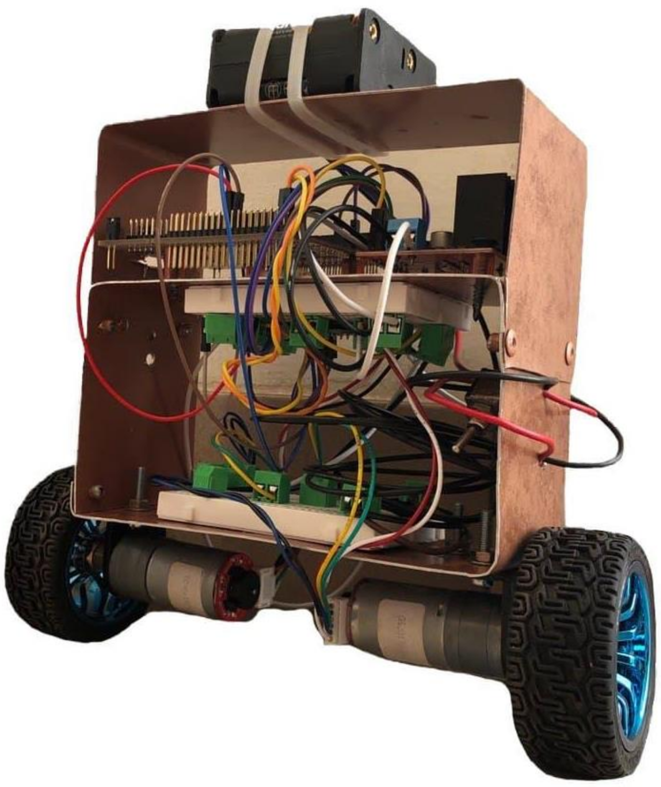
Solution

For the realization of the Self Balancing Robot there are several software modules that manage: two DC motors through L298N, MPU6050 and a joystick.
The two motors are controlled by the values taken from MPU6050 which take the acceleration and rotation values with respect to the plane (x, y, z).
The robot works in two modes and uses a PID for the first mode and a PI controller for the second mode.

1) Self - Balancing mode → from MPU6050 the current angle is calculated and is used to maintain the balance position.

2)Forward/Backward movement →using a joystick the robot can move forward and backward.

A 12V power supply is provided to the motors while a 5V battery supplies STM32 Nucleo-F401RE. More details will be explained in this documentation



Hardware Architecture

- ### Mechanical System

  The general design of the robot was a rectangular body on two wheels. The wheels are placed parallel to each other. On top of the body is placed the battery as high as possible in order to place the center of gravity as high as possible. It is desirable because increase the mobility of the robot and the falling time of the robot. This condition was tested , placing the battery in different levels of the robot and testing which position was the best for our purpose.

  The body can be described as a bookshelf where the components were placed on three different shelves.

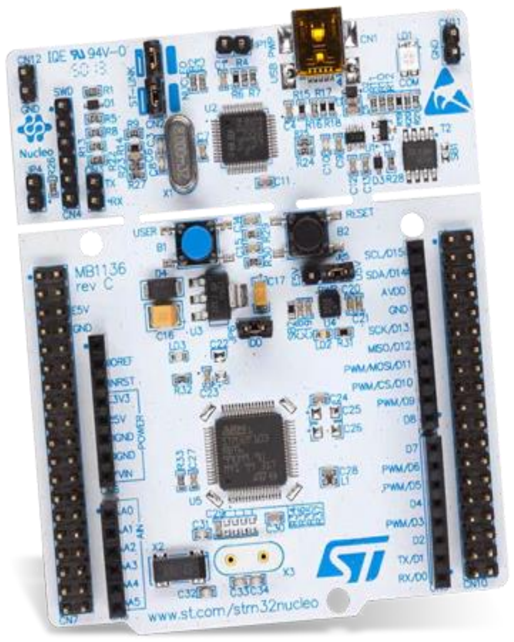
1. **Main Frame**

   The main frame of the body consists of a 1mm thick galvanized aluminum , in order to achieve a solid but very light structure. The structure was also covered with a thin layer of plastic material so as to isolate any possible contact between the pins of all the mounted devices. All the devices present on the structure have been spaced, in order to reduce the risk of overheating (especially due to the motor bridge) and to make sure that you have the space to be able to work better with the various connections and wires present on the structure. The first layer is the biggest of the structure , due the presence of a high number of wires involved. The second layer is instead the core of the robot, in which the main components of the structure are located. All the components have been fixed to the structure in a solid way, using screws and a series of rubber grommets for the correct isolation of the robot. Finally the last layer of the structure is , as described before , used for the battery allocation.

- **Electrical System**

1. STM32 Nucleo-F401RE

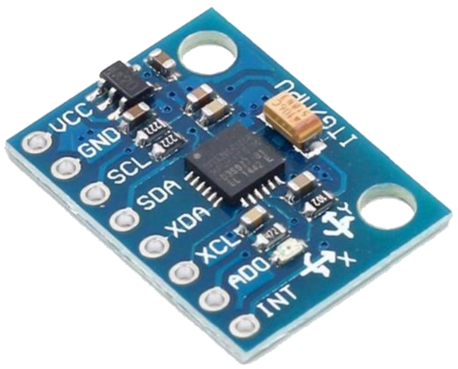
The STM32F401xD/xE devices are based on the high-performance



ARM®Cortex® -M4 32-bit RISC core operating at a frequency of up to 84 MHz.The STM32F401xD/xE incorporate high-speed embedded memories (512 Kbytes of Flash memory, 96 Kbytes of SRAM), and an extensive range of enhanced I/Os and peripherals connected to two APB buses, two AHB buses and a 32-bit multi-AHB bus matrix. All devices offer one 12-bit ADC, a low-power RTC, six general-purpose 16-bit timers including one PWM timer for motor control, two general-purpose 32-bit timers.

2. GY-521 MPU6050

The MPU-6050 is the world's first integrated 6-axis MotionTracking device that combines a 3-axis gyroscope, 3-axis accelerometer. The MPU-6050 features three 16-bit analog-to-digital converters (ADCs) for digitizing the gyroscope outputs and three 16-bit ADCs for digitizing the accelerometer outputs. For precision tracking of both fast and slow motions, the parts feature



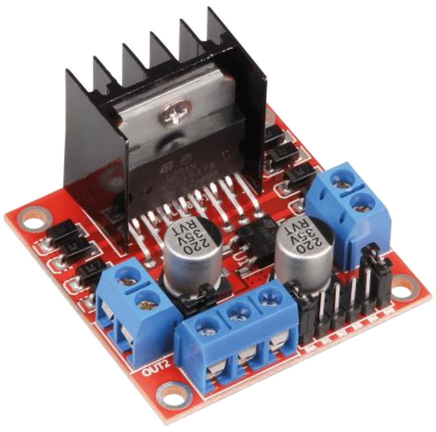
a user-programmable gyroscope full-scale range of ±250, ±500, ±1000, and ±2000°/sec (dps) and a user-programmable accelerometer full-scale range of ±2g, ±4g, ±8g, and ±16g. Communication with all registers of the device is performed using either I2C at 400kHz

3. DC 12V Motor

A DC motor is any motor within a class of electrical machines whereby direct current electrical power is converted into mechanical power. Two DC 12V are used in this project, both integrated with encoder but they are not used.

4. L298N H-bridge

The motors are supplied with a motor driver. It was based on the L298N H-bridge and could supply up to 2A per channel. The motors were controlled by setting a digital 0 or 1 on the direction pin for deciding which direction the motor would go and by supplying a PWM signal for determining the speed of the motors.



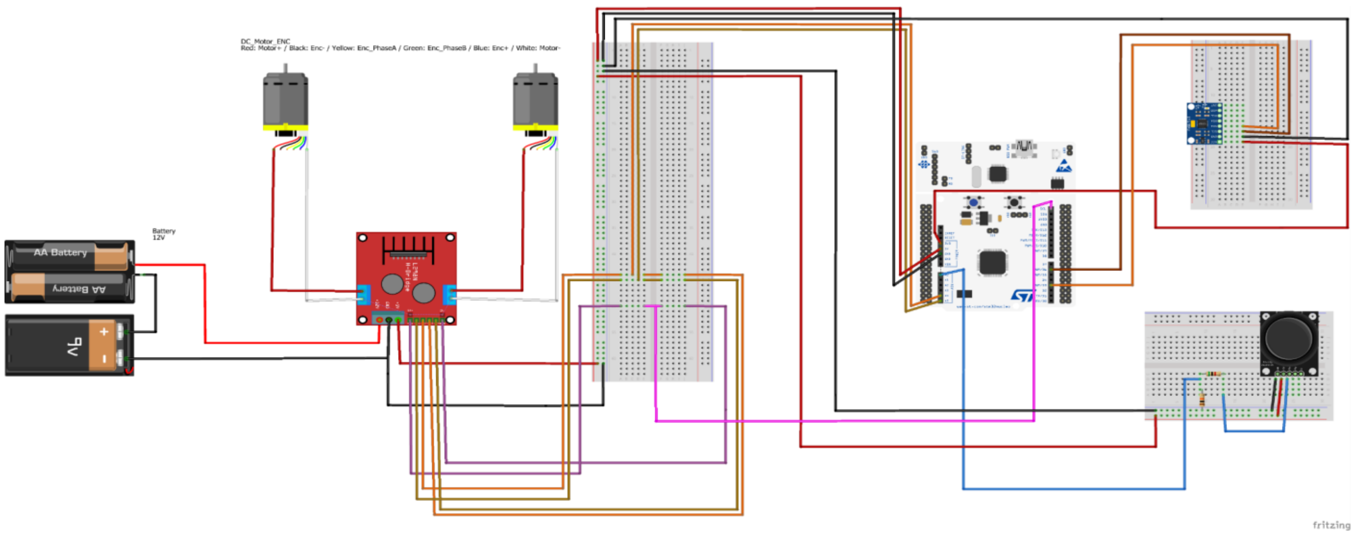
5. Joystick

It is an analog device, made by mounting two potentiometers at a 90 degrees angle. The potentiometers are connected to a short stick centered by springs. This module will generate an output from 0V to 5V depending on its direction (according to x-axis and y-axis).
To read the analog value of X-axis, we need an ADC device. In STM32-F401RE the ADC has a Vref of 3.3V and for that reason the output read from joystick is mapped using a voltage divider.

6. STM32-F401RE Configuration

Pin involved in this project are:

1. PB8 is the output pin of Timer 10 used in PWM mode. It goes in input to Enable A and Enable B pins of the motor driver that supplies the DC motors.
2. PB3 and PB10 are used for communication with MPU6050. They are respectively SDA and SCL pins of the MCU.
3. PC0 and PC1 are pins involved in motors direction. They go in input to IN1,IN2,IN3 and IN4 pins of the motor driver.
4. PA0 is the input pin of the ADC used to read direction from the joystick.
5. PA5 is used to verify joystick direction. It is turned on when it goes forward while it is turned off when goes backward


DC_Motor_ENC
Red: Motor+ / Black: Enc- / Yellow: Enc_PhaseA / Green: Enc_PhaseB / Blue: Enc+ / White: Motor-
Battery
12V
AA Battery
AA Battery
9V
fritzing

DESIGN CHOICES

1. **Robot structure**
   Although there are already built structures on the market, it was decided to build the main structure independently to save money.
2. **MPU6050 positions and configuration**
   The best position is in the middle layer of robot structure because there are less oscillation than the upper layer and less noise than the lower layer.
   In order to have a major sensitivity the full scale of the accelerometer and gyroscope are as low as possible so they are set to ±2g and ±250°/s.
3. **Battery case**
   As described before, a battery case is located on the upper layer such that a greater weight on this layer helps the robot to balance it, in order to place the center of gravity as high as possible. This is desirable because it increases the mobility of the robot and the falling time of the robot.
4. **PWM**
   In this project the value of ARR (of Timer 10 used in PWM mode) is set to 99 so the precision of control law is better than a greater value. In fact, a one unit increment of PWM with ARR set to 99 is stronger than a greater value of ARR so the robot gains more stability. At higher PWM frequencies, the pulses from the motor controller board change too quickly to provide enough energy to spin the motor.
5. **I2C Configuration**
   For communication with MPU6050 it is necessary to use the I2C protocol at a frequency of 4kHz. This communication is made using polling mode to overcome prioritity problem and for semplicity.
6. **ADC Configuration**
   ADC settings used to read data from joystick are:
   - Resolution: 8 bit because there are three different threshold values:
     BACKWARD range in [0,40]
     STOP range in [74,180]
     FORWARD range in [200,255]
     Lower resolution has as consequence hysteresis thresholds too small
   - Scan Conversion Mode Disable because there is only a channel to read
   - Continuous Conversion disable because it is necessary to read direction only before execution of control law.
   - DMA Continuous Requests Enable such that is possibly request data before execution of control law
7. **Interval time of control law**
   Interval time of control law is set to 0.005s in order to be as accurate as possible. That is implemented through timer 11 of MCU that generated an interruption every 0.005s.

Control Design

- **Control Scheme**

  With the next parameter we will make an overview of the robot's control mechanism and how self-balancing works. The first actors involved are certainly the sensors, used to know and transmit the status of the robot. These sensors are the accelerometer and the gyroscope which, present in the MPU, carry out all the measurements useful for control. The measurements made are not directly transmitted to the PID controller, but are processed through a filter, the so-called

  Complementary Filter:

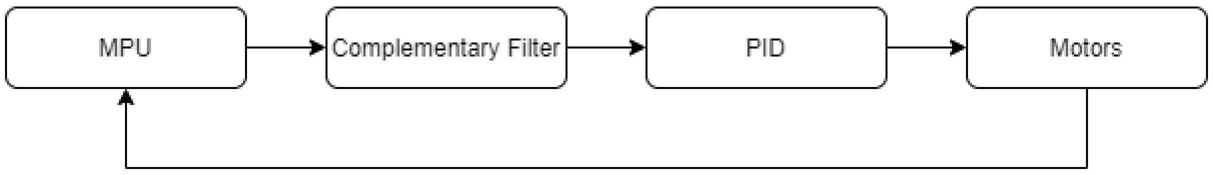
  We will use both the accelerometer and gyroscope data for the same purpose: obtaining the angular position of the object. In both these cases, there is a big problem, which makes the data very hard to use without a filter. As an accelerometer measures all forces that are working on the object, it will also see a lot more than just the gravity vector. Every small force working on the object will disturb our measurement completely.The accelerometer data is reliable only on the long term, so a "low pass" filter has to be used. With the gyroscope we are instead faced with a problem that is the exact opposite of the one we had with the accelerometer , in fact because of the integration over time, the measurement has the tendency to drift, not returning to zero when the system goes back to its original position. The gyroscope data is reliable only on the short term, as it starts to drift on the long term. The complementary filter gives us a "best of both worlds" kind of deal. On the short term, we use the data from the gyroscope, because it is very precise and not susceptible to external forces. On the long term, we use the data from the accelerometer, as it does not drift. In it's most simple form, the filter looks as follows:

  $$angle = 0.98 * (angle + gyrData * dt) + 0.02 * (accData)$$

  The gyroscope data is integrated every timestep with the current angle value. After this it is combined with the low-pass data from the accelerometer (already processed with atan2). The constants (0.98 and 0.02) have to add up to 1 but can of course be changed to tune the filter properly.

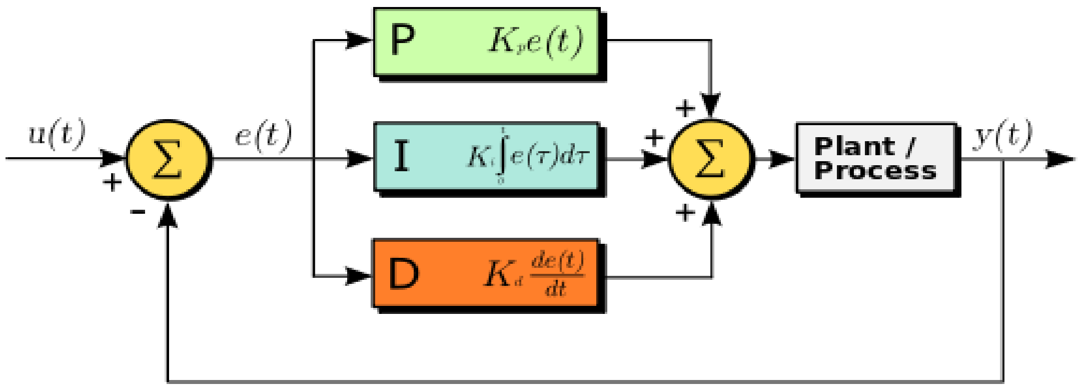
  The data coming out of the filter can finally be used for the real control of the robot. After we get the final angle it’s time to work with it. As we said before to make the robot balancing it is necessary that the wheels move according to the angle, for this and even though the simulation was done using the PID method to balance the robot. A PID controller attempts to correct the error between a measured process variable, which in our case is the robot angle, and a desired setpoint, for us the setpoint is the equilibrium angle, i.e., the angle when the robot is at an equilibrium position, by calculating and then instigating a corrective action that can adjust the process accordingly and rapidly, to keep the error minimal, by corrective action we understand the wheel’s speed. We have tuned each of the PID values manually by trial and error.Then with PID’s corrective action we feed the motors and if everything works well it should balance.

PID:



Purpose

The control used in this project is a PID controller (proportional–integral–derivative controller) that continuously calculates an *error value* as the difference between a desired setpoint and a measured process variable and applies a correction based on proportional, integral and derivative terms (denoted *P*, *I*, and *D* respectively). These 3 terms are calculated using 3 equations that are multiplied by 3 coefficients Kp, Ki and Kd.



$$Out(t) = Kp\ e(t) + Ki \int e\ dt + Kd\ \frac{de}{dt}$$

Parameters

Every of these 3 coefficients have a certain purpose and for that reason the values assigned to them should be chosen according to the structure and weight of the robot to be balanced.

1. **Kp:** is used to indicate how much control should generate a control action in response to the incoming error. The use of that constant permits to have a proportional sensitive response in output from the **Proportional Term**, P.

2. **Ki:** the use of that constant permits to accelerate the movement of the process towards setpoint and eliminates the residual steady-state error. The value of this constant generates a high or a low overshoot in output of the **Integral Term**, I.

3. **Kd:** the **Derivative Term**, D, permits to improve the settling time and stability of the system proportional to the value assumed by it. Higher is the value of this constant and more time is needed for stability.

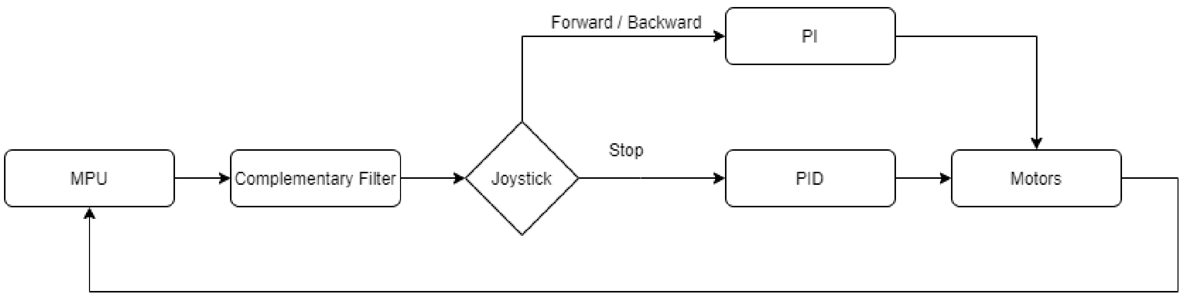
This controller takes in input the degrees of error to stay balanced on its reference (ref) and computes a value of duty cycle to move the motors on its set point.

Tuning

Then Kp value is increased from zero till a point where the system starts to oscillate from its mean position. And as the Kp value is increased, it was seen that response of the system increases gradually. Keeping the Kp value a constant the Ki value is then increased till a point where it is seen from the system that a small inclination towards a direction tends to accelerate the vehicle towards that direction. When the Kp and Ki values are properly obtained, then the Kd value is increased. The Kd value is increased such that rapid acceleration is reduced considerably. Proper Kd value results in lesser overshoots and oscillations. After obtaining the rough Kp, Ki and Kd values a fine tuning is done. The above steps are repeated a number of times to obtain the best combination of gain values.

- ## Final Control Scheme using the Joystick

Differently from the previous control scheme , when the joystick is used the flow of data varies its path, having to submit to the commands of the joystick which has a higher priority than the usual balance control.



In fact, when the analog stick is moved forward or backward, the derivative term of the controller is canceled and the integrative term is attenuated so that the robot can move freely. In this way when the robot hat to move the oscillation are reduced because Kd equal to zero reduced oscillation and a lower value of Ki braking effect. At the same time, however, the robot continues to control its own equilibrium returning to steady state as soon as the analog returns to the equilibrium position.

Software Architecture

Software Architecture chosen is an Interrupt Driven Architecture. The interruption is generated by timer 11 of the board every 0.005 seconds.

Modules:

1. accel.h: This module contains utility functions and a data structure needed to manage accelerometer data
2. angle.h: This module contains utility functions needed to manage pitch.
3. gyro.h: This module contains utility functions and a data structure needed to manage gyroscope data.
4. control_loop.h: This module contains utility functions needed to read data and execute control law.
5. joystick.h: This module contains utility functions and a data structure needed to manage joystick data.
6. motor_diver.h: This module contains utility functions needed to manage direction and movement of motors.
7. mpu6050.h: This module contains utility functions and a data structure needed to manage MPU6050 communication.
8. PID.h: This module contains a utility function needed to execute control law.

Software Interfaces

Through this link is possible to view UML diagram regarding software modules of this project:

https://app.creately.com/diagram/zmvYvXgW6mT/view