Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

2E10 Gold Report

Group Y12

Challenge Documentation

GOLD CHALLENGE IDEA

For the gold challenge, 2 constraints were given. The buggy must use both the Arduino's IMU and the wheel encoders. We were encouraged to be creative with the orientation/acceleration data. Immediately, we knew we wanted to design a buggy with the ability to balance on 2 wheels. This satisfies the constraints as it heavily relies on the Arduino's onboard IMU. The encoder speed could then be reported back to processing.

PLAN

We decided to tackle this similar to the Bronze and Silver Challenges. We started by creating a timeline:

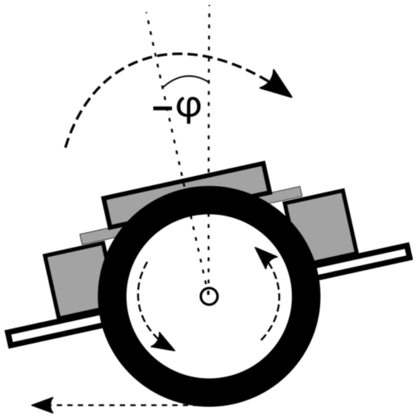
Timeline:

1. Work with Processing to ensure stable sending of Buggy Speed
2. Set up IMU to correctly and learn how to interpret the data
3. Determine the angle the Buggy is currently at using the IMU
4. Configure Motors to work with the angle data keeping the buggy in Balance
5. Tune PID to create a stable, balanced Buggy
6. Possibly implement other features (Time Dependant)

We succeeded in making the buggy balance, however it was not as stable as we would have liked. The original plan was then to build upon this, once the basic constraints had been met. Unfortunately, due to time constraints we did not have the chance to expand upon this challenge further, but are proud of what we achieved in the time given.

HOW DID WE MAKE THE BUGGY BALANCE

Using the Arduino's IMU accelerometer & gyroscope, we could collect data on the buggy's orientation. From experimenting with different angles, we found the angle at which the buggy wants to balance at. This is the angle at which the buggy's centre of gravity (COG) is directly above the wheels. We needed to find a way to keep the buggy at this target angle and to stay in this position.



This was achieved by rotating the wheels in the direction in which the buggy is falling. As seen in the diagram on the right (1), as the buggy tips to the left, the robot attempts to stabilise itself by moving left. Assuming the correct speed is applied, the buggy's angle would return to its target position, and the COG would return back to being directly above the wheels. Finding the correct speed in which to rotate the wheels was a far more challenging task than we first thought.

HOW DID WE FIND THE CORRECT MOTOR SPEED

Using a proportional integral derivative (PID) controller, the correct speed can be applied to the motors. The downside of PID controllers is that they require tuning, which can be a tedious process. We heavily underestimated the time we would spend tuning the PID. Unfortunately this hindered our progress.

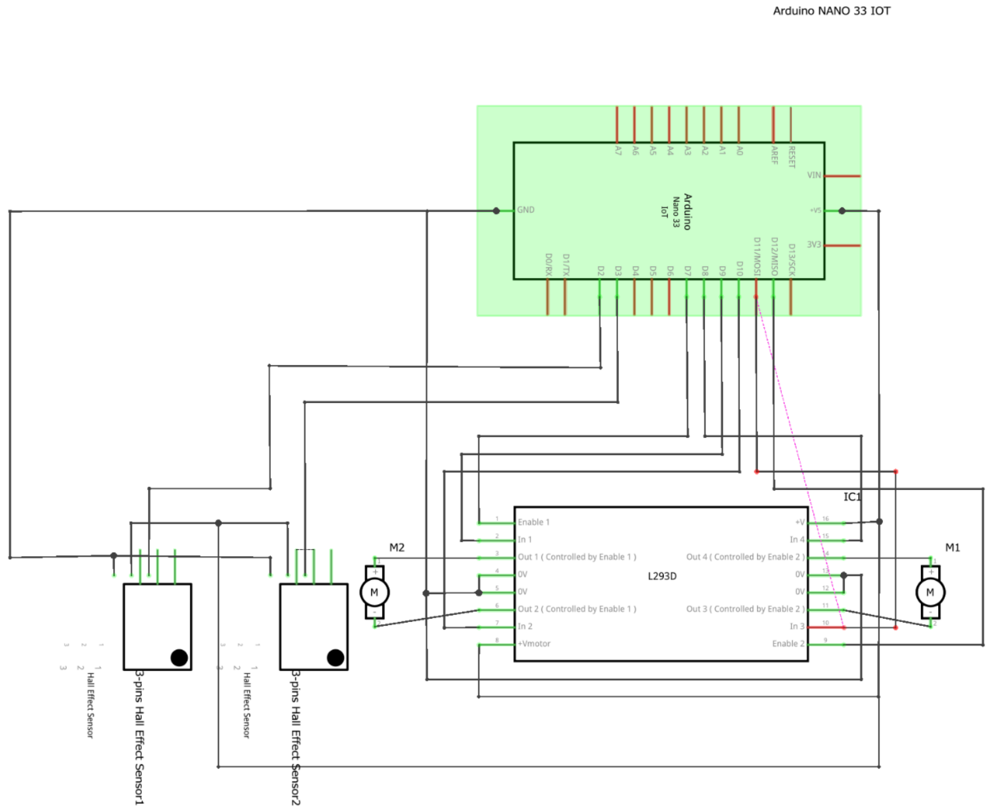
WHAT FUTURE PLANS DID WE HAVE?

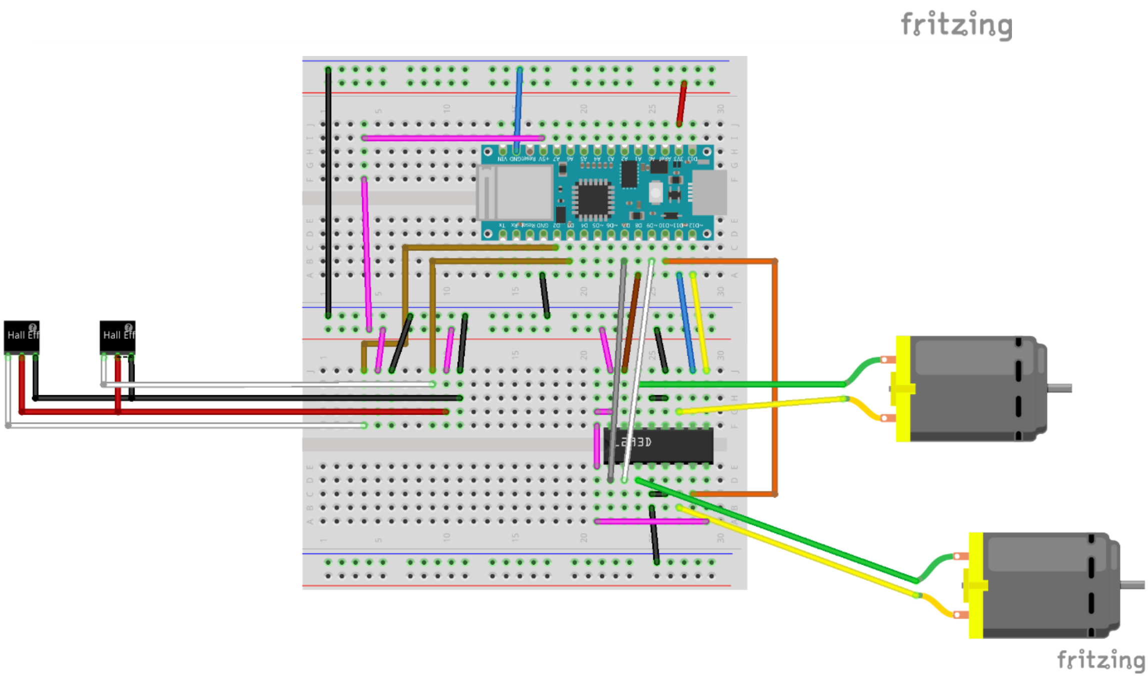
Once we had the PID tuned correctly, we believed we could've easily developed code to move the buggy forward and backward. By altering the target angle a small amount, the buggy would tip in a controlled manner and travel in a chosen direction. We also planned on having the ability to turn while upright. This could've been achieved by independently controlling the speed of each motor. This is similar to how the buggy turned in both the bronze and silver challenge.

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Hardware Documentation

For the Gold challenge we removed the Buggy’s IR and Ultrasonic sensors. We did this at the beginning of the attempt as we were not sure how much time we would have left over to implement other features of the Buggy. The updated Schematic and Breadboard diagrams are shown below.





Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Software Design Documentation

BUGGY PSEUDO CODE (GOLD)

Include IMU library
Include Encoder library
Include WifiNina library
Define buggy WAN **name** , **password**
Define WAN port

Define **Encoder1** & **Encoder2** pin numbers

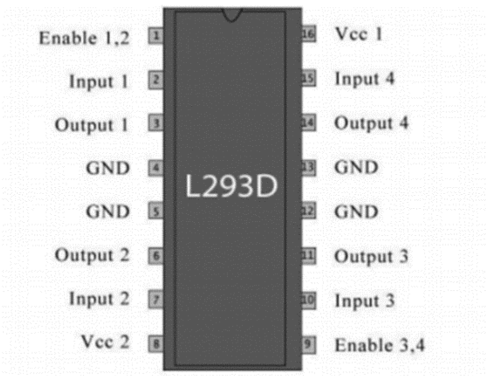
//Variables for PID
Initialise **currentTime**, **previousTime**, **Interval** as longs
Initialise **sumOfErrors** as double
Initialise & Set double values **Kp**, **Ki**, **Kd** to PID constants found

//Variables for Encoder
Initialise **oldPosition**, **speedStart**, **newPosition**, **distancePerStep** = 1.25 as long's
Initialise **difference** & **encoderSpeed** as ints.

//Boolean variable used to turn on/off the buggy
Initialise bool **drive** = false

//the angle the buggy balances at
Initialise **targetAngle** as float = 0.208

Define Motor A Connections (**enA**, **in1**, **in2**)
Define Motor B Connections (**enB**, **in3**, **in4**)
Initialise variable **Speed** as float



Initialise variable **motorPower** as double
Initialise & Set double values **Kp**, **Ki**, **Kd** to PID constants found
//variables for gyroscope
Initialise **accX**, **accY**, **accZ** as floats.

```
Void Setup(){
        Define Serial port as 9600
        Set PinMode for all pins
        Set motor initial state to off
        Start Wifi network using name and password defined previousl
        Print IP address to Serial Monitor
}

Void loop(){
        Set currentMillis to the number of milliseconds since the code started running

        IF difference between currentMillis & previousMillis is greater than the interval {
                Set previousMillis to currentMillis
                Set difference = difference between newPosition and speedStart
                Set speedStart = newPosition
                }
```

```
        }

        IF client is connected to WAN{
                Read character "C" received from client
                If C = "W" then set bool drive to true
                IF C = "S" then set bool drive to false
        }

        IF drive is true{
                IF the IMU is available{
                        Read the IMU accelerometer information
                }
                //This is the angle of the buggy
                Define variable Angle to polar coordinates between Z & Y acceleration
                Define speed as MotorPower function with Angle as input
                If the Speed is greater than 255, set it back equal to 255
                IF the Speed is a number{
                        Use SetMotorDirection Function
                        Set motor speeds to variable Speed
                        Use Print () function
                }
                Read encoder newPosition
                If position has changed, set oldPosition to newPosition
        } else if drive is false{
                Stop the buggy
        }
}

//Function to change the motor direction
Void setMotorDirection( currentAngle, targetAngle){
        if(currentAngle < targetAngle){
                Set motor direction in direction that robot needs to go in order to rebalance
        }
        if(currentAngle > targetAngle){
                Set motor direction opposite to above
        }
}


//Returns speed in PWM value, uses PID
Double MotorPower(currentAngle){
        Calculates the appropriate motor speed in order to rebalance the buggy
}

Void Print(){
        Prints the current Speed to processing
        Also was used for testing purposes
}
```

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

PROCESSING PSEUDO CODE (GOLD)

```
Import controlP5 library
Import processing.net library //used for wifi

Initialise variables

Void Setup(){

        Set canvas size
        Set background colour
        Set frame rate

        Connect to Arduino's WAN using its IP Address

        Set up Start & Stop buttons (Position, Size, Colour)
        Set up Speedometer/Knob (Position, Range, Size, Initial Value)
        Set up Text Display Area 1 (Position, Size, Colour)
}

Void draw(){
        Set speed as speed value read from client
        Display speed in speedometer
}

Function ReactOnEvent(event){
        If event is START button being pushed
                Send "W" to Arduino

        If event is STOP button being pressed
                Send "S" to Arduino
}
```