MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY OF UKRAINE

“IGOR SIKORSKYI KIEV POLYTECHNIC INSTITUTE”

THE DEPARTMENT OF DESIGN OF ELECTRONIC DIGITAL EQUIPMENT

**COURSEWORK**

in the discipline: Analog Electronics-2

on a topic: “DC motor PID controller”

student of the second course, DK-82 group

field of study: telecommunication and radiotechnics

Yegor Krapovnytskyi

Header:

\_\_\_\_\_associate professor, Korotkyi I.V.\_\_\_\_\_

(position, academic title, academic degree, surname and initials)

National assessment: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

number of points: \_\_\_\_ ECTS Assessment:\_\_\_\_\_

Member of a commission:\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_ associate professor, Korotkyi I.V.\_\_

(signature) (position, academic title, academic degree, surname and initials)

Kiev-2020

**TABLE OF CONTENTS**

1. **Introduction………………………………………………………………………2**
2. **List of acronyms………………………………………………………………….**
3. **Part 1 – Fundamentals of PID theory** 
   1. **Hardware setup and connection………………………………………………3**
   2. **Motor run and encoder read…………………………………………………..4**
   3. **Determine the RPM…………………………………………………………..6**
4. Part 2 – Designing a controller
   1. Plant model estimation……………………………………………………….9
   2. PID controller design………………………………………………………...10
   3. Implementing and testing PID controller on real hardware………………….16
5. Conclusion………………………………………………………………………...18

Reference list……………………………………………………………………...

**INTRODUCTION**

**Assume that we have a system we want to control. The system has an input, where it receives a command signal and the output, where we can measure a controller variable and it must repeat the command one. Also, we have a feedback path that goes from the output, then difference between the input and the output is calculated to see how far the system is from the desired point, so we have a closed loop. If the requirements are not that strict, the engineer can use an open-loop system. It is not as accurate as the open-loop one, because the amount of input force is not adjusted based on the real position of the system, so it may deviate from the required direction, or may overshoot. Feedback allows the engineer to make adjustments by sensing the output and comparing**

**it with the input. If to speak about a system with feedback loop and a controller, we imply a very sophisticated system, on which depend many prime things like human health, economic power of the country or even lives. For such a system requirement are of the highest level, so it must operate ideally. To make it to operate in such way, the corresponding controller is required. There are different types of controllers such as PI,PID,PD,etc. The goal of my course works is to implement a PID controller for the nonlinear system. In my case, I will use a DC Motor as a nonlinear system. It is a good variant, because of its low cost and convenience in use with motor drivers, and, of course it is simply enough to begin studying control theory. Also, the course project is actual, because PID control system is widely used in everyday life, for example a climate control or cruise control of the vehicle. In addition, studying this topic requires basic knowledge in control theory, that is very useful for further electronic projects, or automotive engineering.**

**In earlier times PID controllers were made of operational amplifiers and were exceptionally analog. Now they are implemented by software inside the PC, microcontroller or FPGA. In the project I will use MATLAB to build a motor model and design a PID controller for it. MATLAB allows to automate the process and make it more accurate. This approach is modern and the best for the engineers.**

**LIST OF ACRONYMS**

**PID Proportional, Integral, Derivative**

**PI Proportional, Integral**

**DC Direct current**

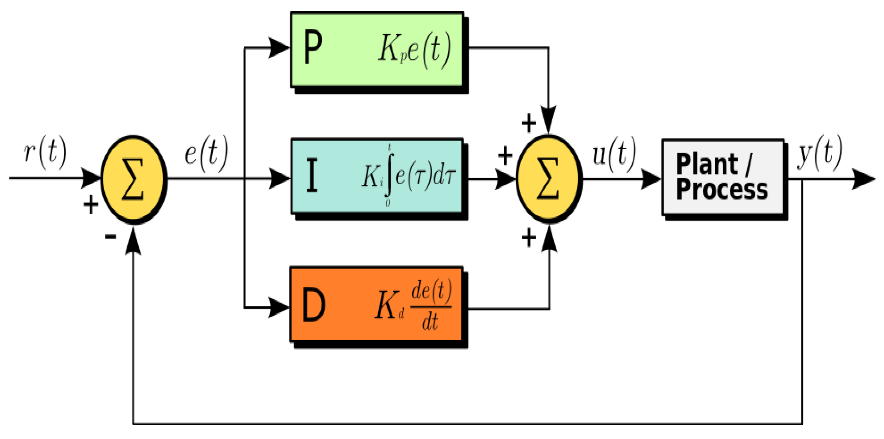
**PPR Pulses per revolution**

**RPM Rotations per minute**

**FPGA** Field-Programmable Gate Array

**PART 1**

**FUNDAMENTALS OF PID THEORY**

**During typing the introduction, I have mentioned a word “system” a lot of times.** In control theory this system is called a plant and our engineering purpose is to implement a controller for the plant, which provides the highest fidelity, lowest time response and cost. In my case it will be a PID controller. It uses the difference from the input and the output and then provides appropriate gains for all of P ,I and D terms to get a desired value on the output of the plant in minimum amount of time. A simple block diagram of the system with PID controller is showed in figure 1:

**Figure 1: plant with feedback path and PID controller**

Let’s skip over each of P, I and D terms in common words. Proportional part acts to reduce the error by simply amplifying or attenuating the error signal depending on its sign. The Derivative part is used to predict the system state and depends on the rate of the error signal change. If the error decreases too fast (that meant that P term is too high), then the derivative is negative and it is used to dampen the resulting signal to prevent overshooting. The Integral part of the controller is used to take down a steady state error. Assume that our system is a little out of desired point, but the error is too low to make the proportional part act. The small error accumulates in time and makes the Integral part to provide the finish touch to get 100% fidelity. After passing through each term the signal is summed and fed to the plant.

The main issue is how to adjust each of P, I and D terms to get the desired result. That’s why a PID Tuning plan exists and I have learning it via Brian Douglas videos on YouTube. It is shown in figure 2 as a very simple block-scheme. Obviously, I will use the simplest tuning methods because they cover the whole fundamentals and are very important if to speak about studying control theory.

**Figure 2: PID tuning guide block-scheme**

**The best approach for designing a PID controller is to represent a real model as a transfer function and test it on the software: in my case MATLAB Simulink. This will prevent a physical hardware to collapse during tests. Then, I will design a controller for the model and then test it on real hardware. This way is nice because it combines both efficiency and working with software to get accurate result.**

**By the way there are 3 ways to build a model:**

* **Derive from first principles**
* **Linearization**
* **System Identification Tool**

**I assume that my DC motor is well-behaved, and that’s wright of comparing with complex systems. As I said above, I have a physical hardware. I could derive a model from first principles, but as an electrical engineer I’m not really interested in DC motor mechanics and differential equations that describe it. I will measure input and the output of the plant and then use System Identification tool to represent a motor as a second-order transfer function. After I have done this, I will apply manual tuning and MATLAB PID Tuner to get acceptable P, I and D terms, that control the system.**

**For this course work my plan is:**

***Design a physical system*** *→ run input sequence →* ***model-based tuning*** *→ manual tuning/MATLAB Tuner*

**After doing this I will compare each of variants above and make a conclusion.**

**A plant with a PID controller can be represented as the next transfer function:**

Where: Gp — ­plant transfer function

Gc — PID controller transfer function

In this coursework I have skipped most of control theory maths, because it is sophisticated and doesn’t represent the whole real situation and may be useful only in theory. Of course, I can calculate appropriate terms and spend a lot of time for that, but it doesn’t guarantee the best result and in practice can be useless. A simply try and error method is more useful and convenient in this case. It requires less time and flexible enough to tune a PID controller.

**PART 2**

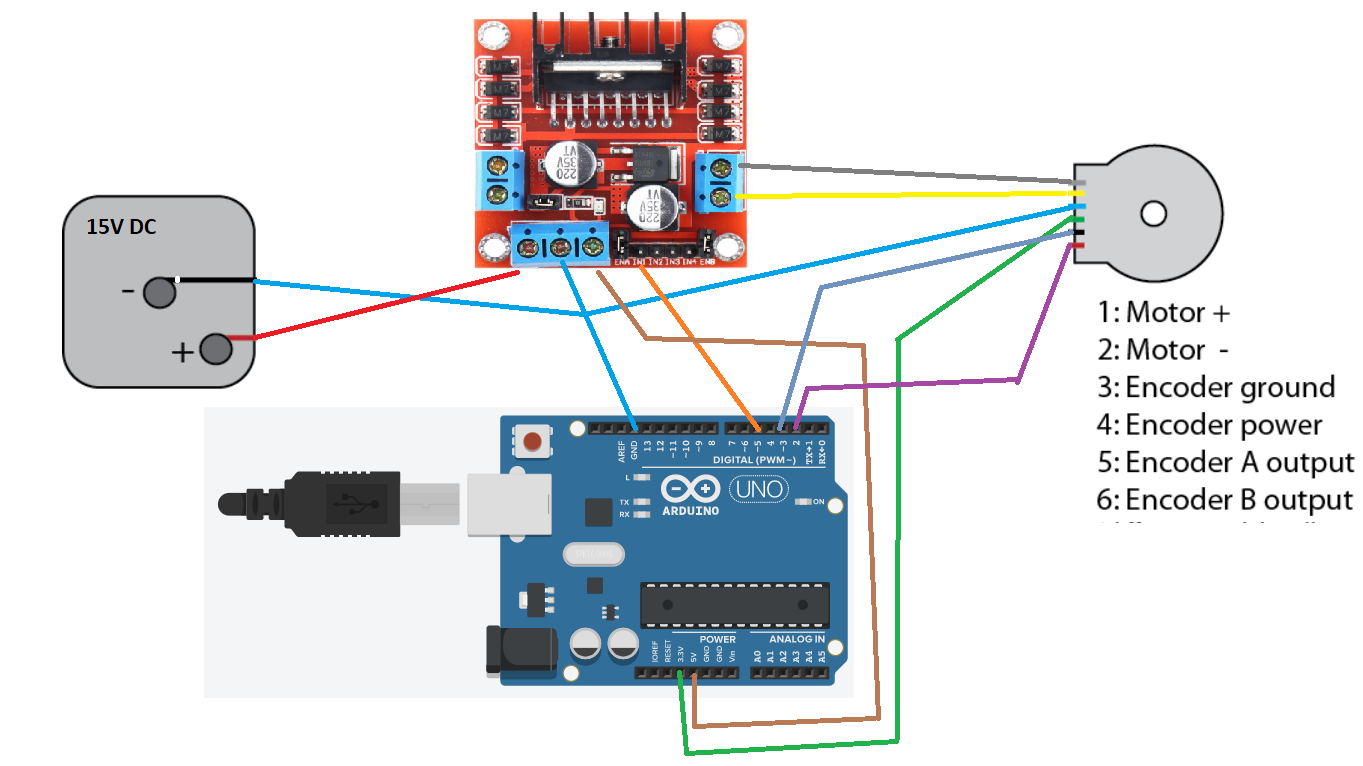
**WORKING PROTOTYPE DESCRIPTION**

**2.1 List of used components is shown in table 1:**

|  |  |  |
| --- | --- | --- |
| **№** | **Title** | **Description** |
| **1** | **L298N** | **Dual H-bridge motor driver. Technical parameters:**   * **Power-supply voltage: DC 5-35 V** * Peak current: 2 Amp * Operating current range: 0 ~ 36mA * Control signal input voltage range: * Low: -0.3V ≤ Vin ≤ 1.5V. * High: 2.3V ≤ Vin ≤ Vss. * Maximum power consumption: 20W (when the temperature T = 75 ℃). |
| **2** | JGY-370 12V DC Worm Gear Motor with Encoder | **Technical parameters:**   * Rated Voltage: DC 12V * No load speed: 30 RPM * No Load Current: 35mA * Load torque: 0.55kg.cm * Load Current: 180mA * Stall Current: 1A * Ratio: 37.3 * Stall torque: 2.2kg.cm * **Encoder power: 3.3V** |
| **3** | **Arduino Uno R3** | **Famous Arduino microcontroller, but its Chinese equivalent.**  **Technical parameters:**   * Microcontroller: ATmega328 * Operating Voltage: 5V * Input Voltage (recommended) 7-12V * Digital I/O Pins 14 (of which 6 provide PWM output) * Flash Memory 32 KB (ATmega328) of which 0.5 KB used by bootloader * SRAM 2 KB (ATmega328) * EEPROM 1 KB (ATmega328) * Clock Speed 16 MHz |
| **4** | **15V Case** | **15V battery case for 10 AA batteries** |
| **5** | **BLS DuPont cables** | **Cables to connect all the parts** |

**Table 1: List of components**

**2.2 *Hardware connection***



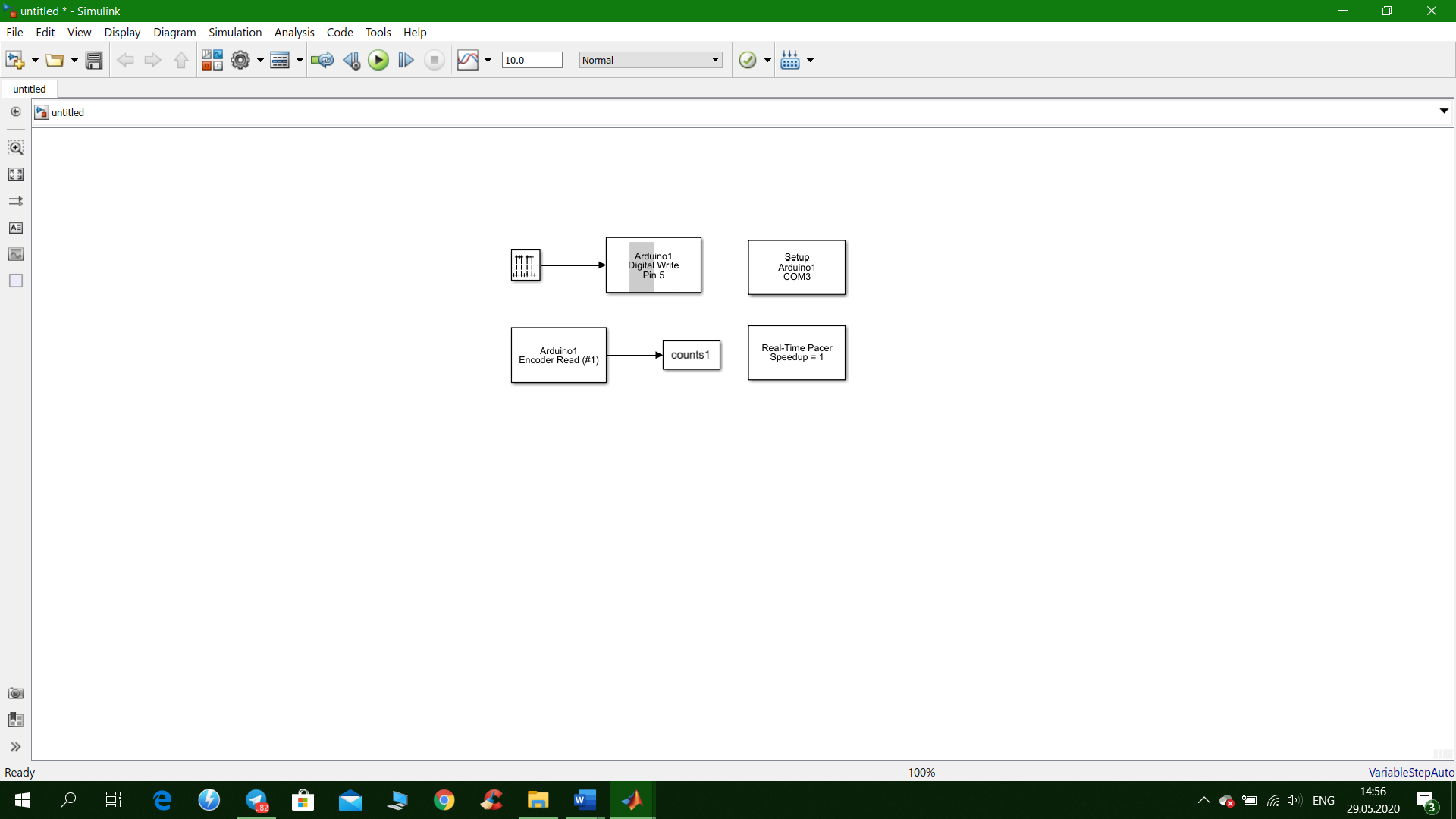
**Figure 3: Hardware connection**

I control the DC motor with the help of L298N driver. As a power supply I use 15V instead of 12V, because, in practice on the output pins of the motor driver voltage is lower than the supply voltage on 2-3 volts. Also, the motor is just 30 RPM, so it’s quite slow and I try to get maximum efficiency and speed. The Hall sensor is connected directly to Arduino and sends pulses to it, while the motor is moving, so it performs a function of feedback. It is connected to 2 and 3 pins. Arduino controls the motor due to 5 pin. It is connected with the host computer via USB cable.

2.3 Software setup

Firstly, I install Arduino IO support package for MATLAB. To make all the blocks work, I need to upload a server program on the Arduino board via Arduino IDE. This program works in parallel with Simulink and will allow to control the board directly from MATLAB. This program uses a serial port to communicate with host computer and executes received commands. Such a program is presented in Addition 1.

Now work in Simulink begins. Figure 4 shows the first block diagram, that is used to run a motor and read data from encoder. Real-Time Pacer block is used to synchronize time of simulation with real time. I set sample-based pulse to Arduino digital output pin 5 with sample time 0.02 second. Period is 28 seconds and pulse width is 26 seconds.



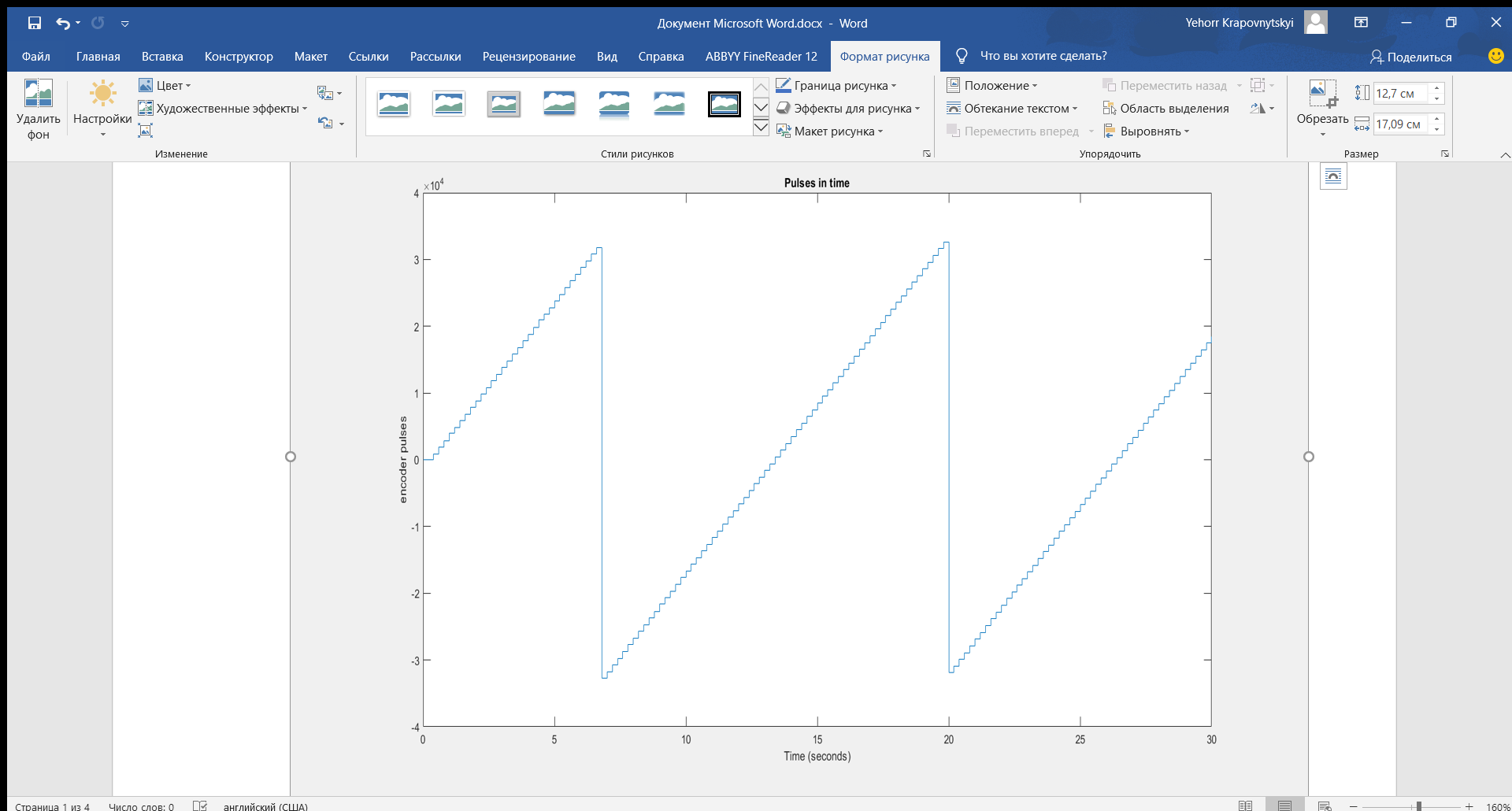
**Figure 4: Opening block-diagram to make the motor rotate through Simulink**

**PART 3**

**BULDING-UP A MODEL**

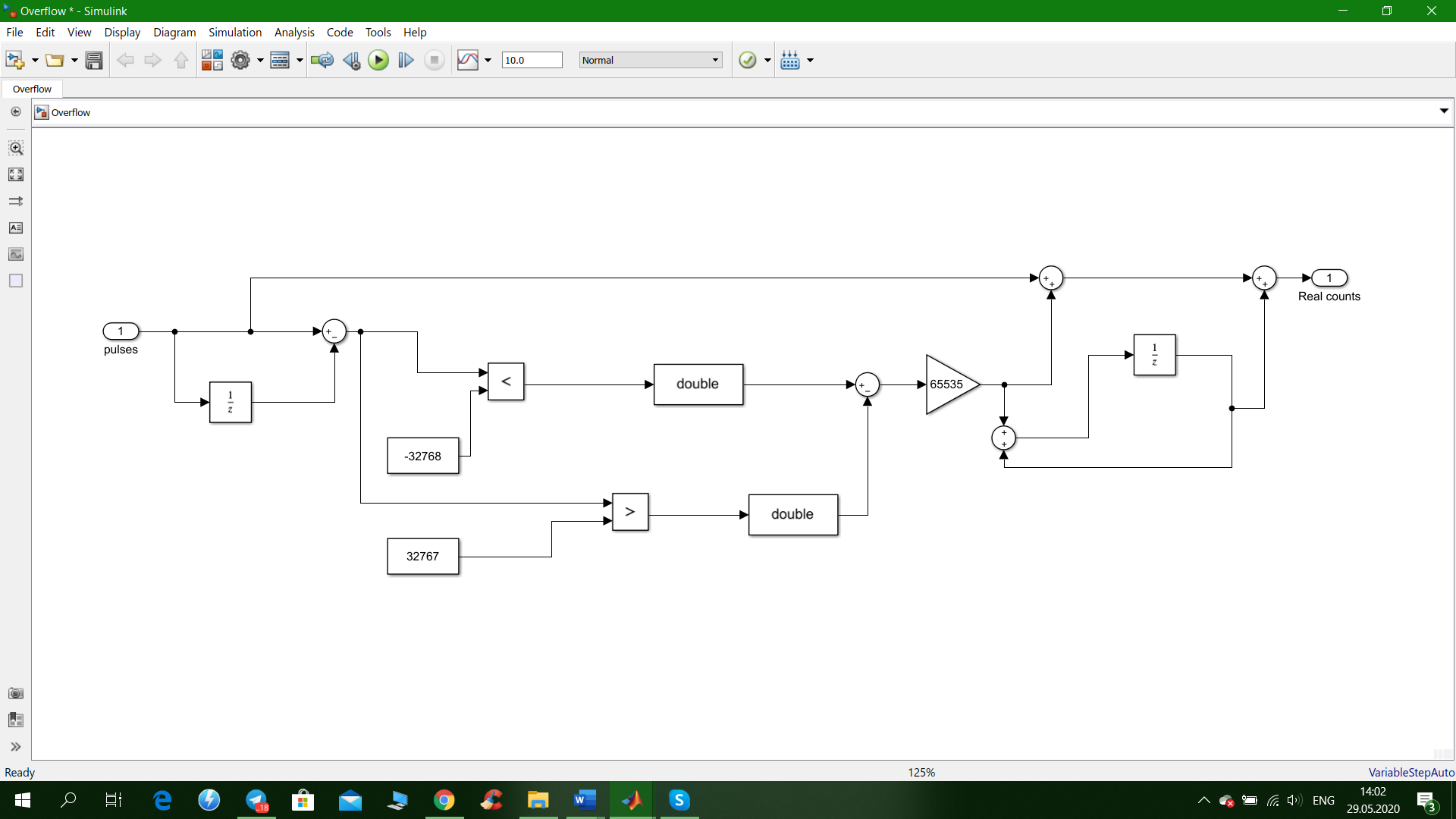
**3.1 Motor run and encoder read**

**When everything is ready, I can launch the motor. In this section I will obtain the output of the motor in RPM. Then, based on output and input I will be able to determine the transfer function of the motor, and so design a PID controller for it.** When I begin the simulation and plot count versus time, I get such a picture (fig. 5):

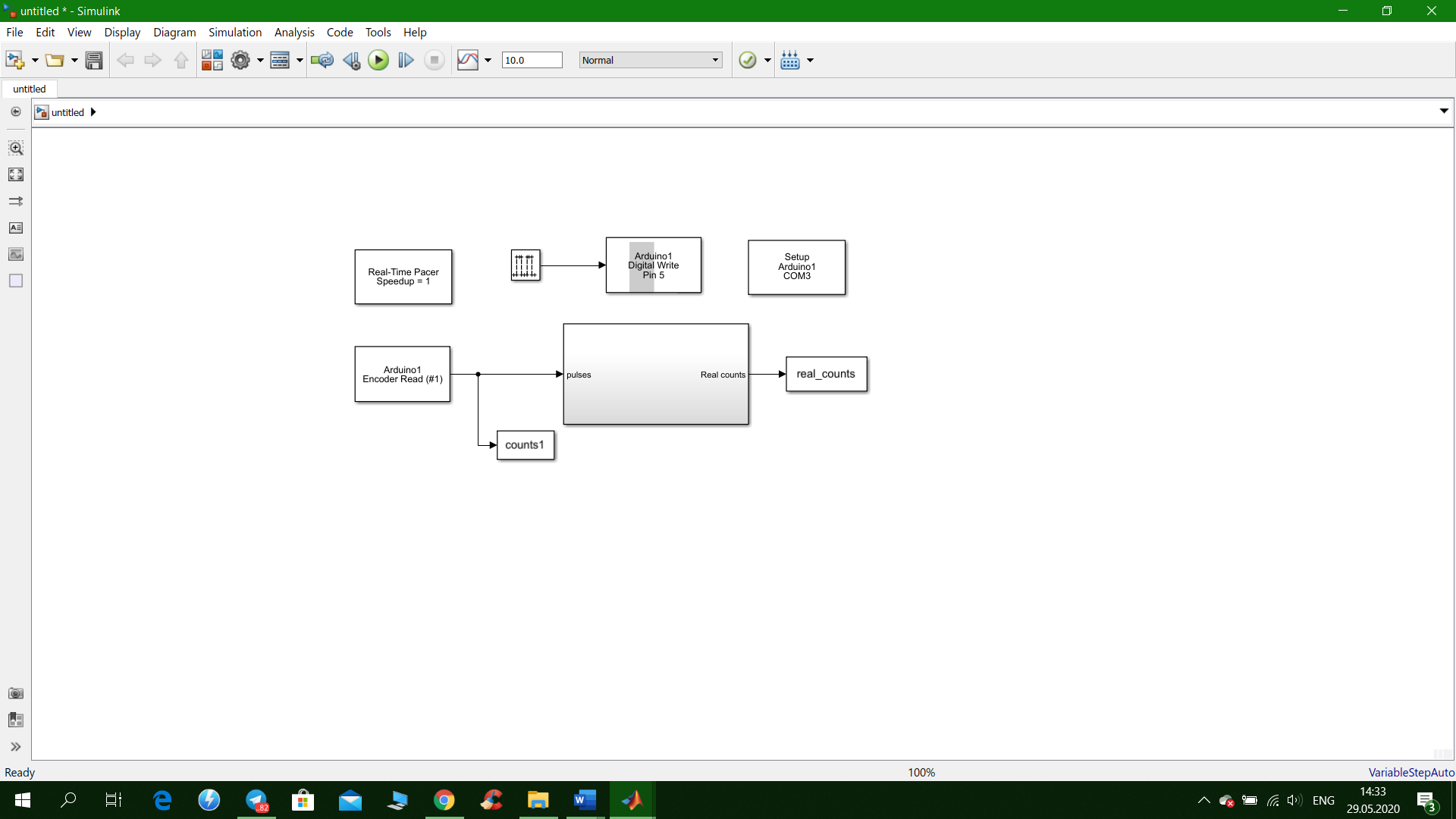


**Figure 5: Buffer overflow after receiving pulses from encoder**

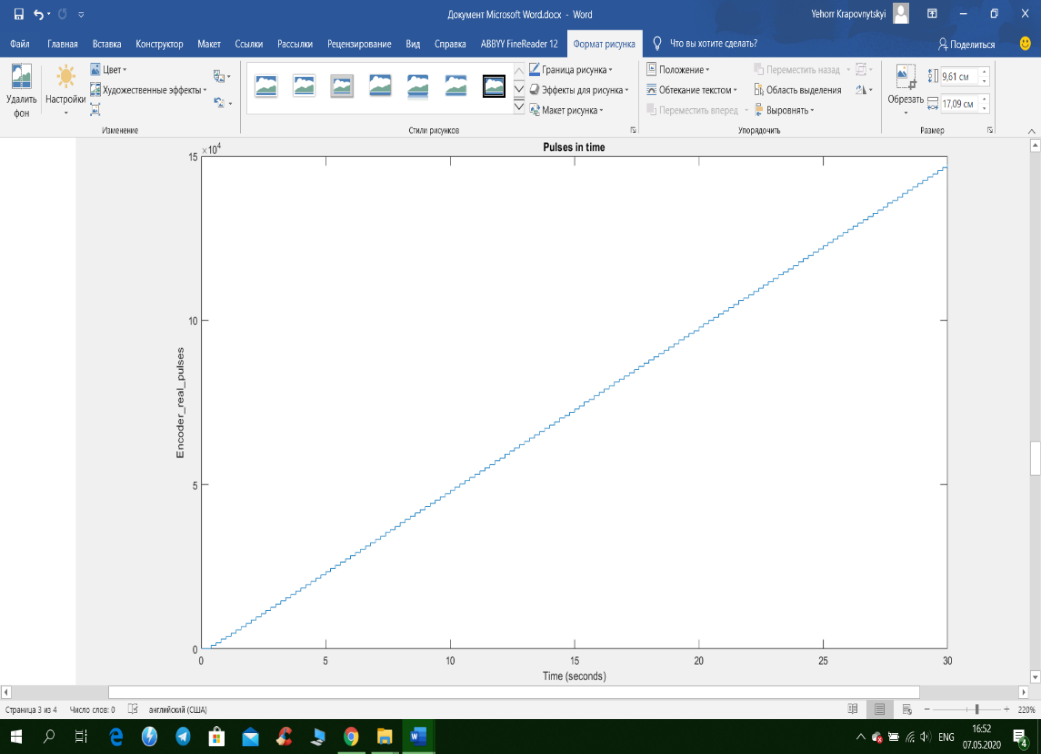
As we can see, there is an overflow of data, because the buffer in which it is collected is just 16 bit (1 for sign and 15 for value), so I implemented a function to get rid of the overflow. Its block diagram is shown in figure 6.

**Figure 6: A function that takes the overflow away to further data processing**

I compare counts of present and previous sample, to determine whether the overflows occurred in the -32768 pulse or 32767, when I accumulate the value and continue to count impulses. After this correction I get this block diagram and plot result (figure 6 and 7):



**Figure 7: Block diagram after applying anti-overflow function**

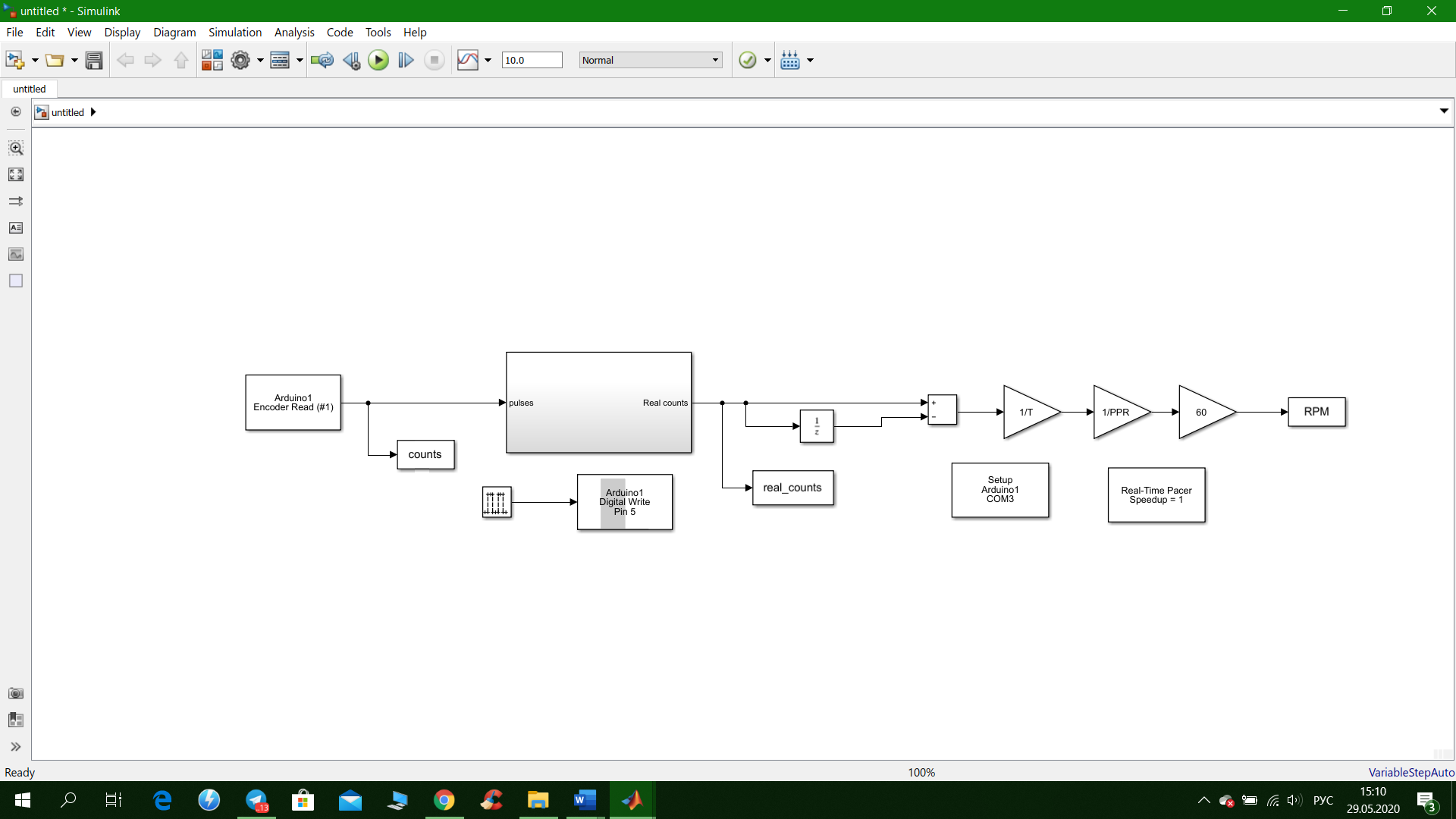


**Figure 8: Motor counts without overflow**

**3.2 Determine the RPM**

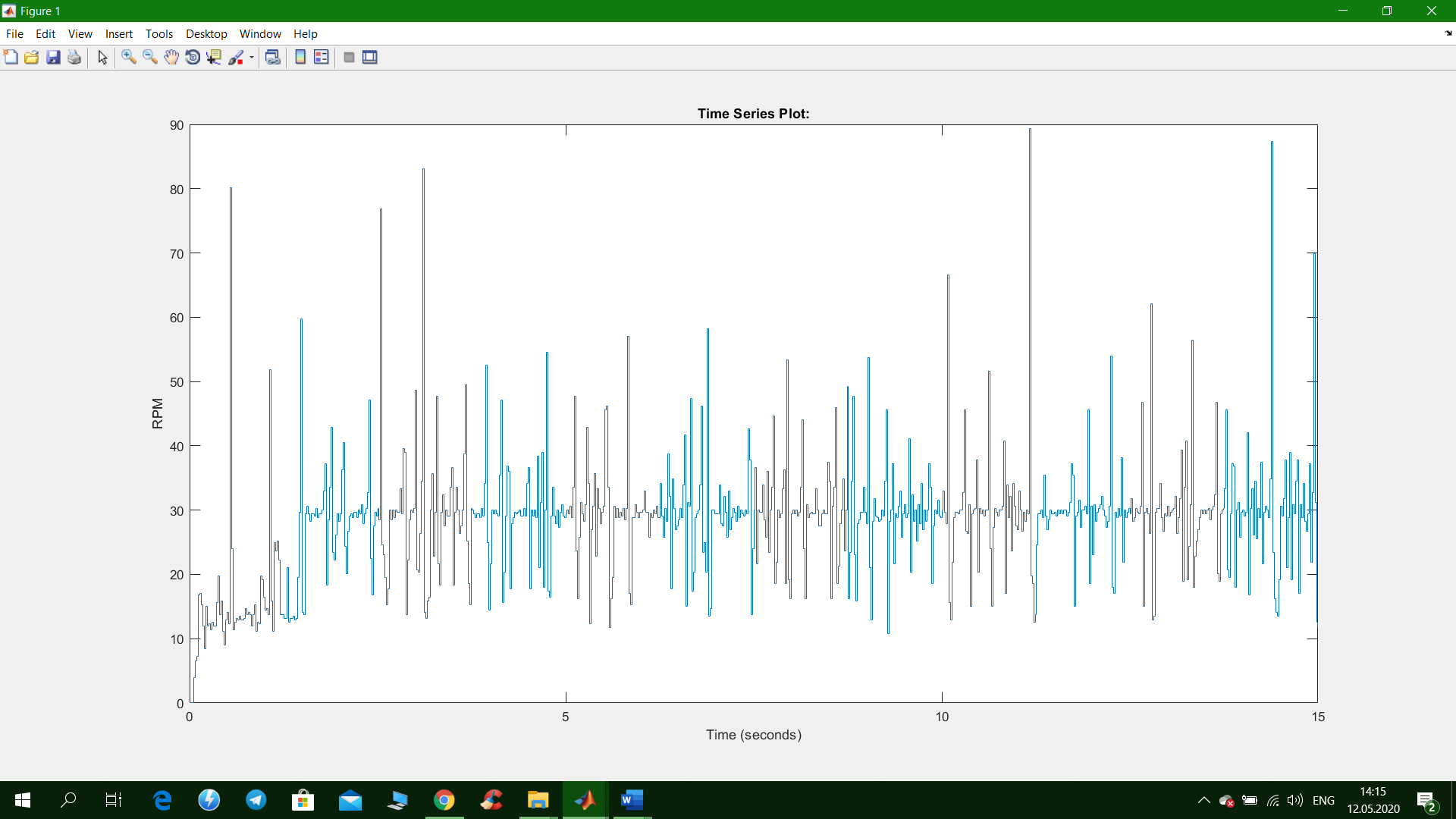
Now I need to determine motor speed practically and plot it versus time. Firstly, I calculate difference in counts (in position) and then divide by sample time(T), so I get counts/seconds. Secondly, I need to know encoder counts per revolution. I had, recently, calculated pulses during 30 seconds when the motor is constantly rotating. I got almost 150000 counts. At the same time, I know that, theoretically motor has 30 RPM, so 15 rotations per half a minute. Now it’s possible to determine pulses per revolution (PPR):

Now I have PPR and counts/seconds. To determine RPM, I have to take counts/seconds then divide by PPR (I get revolution per second) and multiply by 60 to convert into revolutions/minute. The following block diagram represents all the steps above (fig. 9):



**Figure 9: Block diagram with RPM as final output**

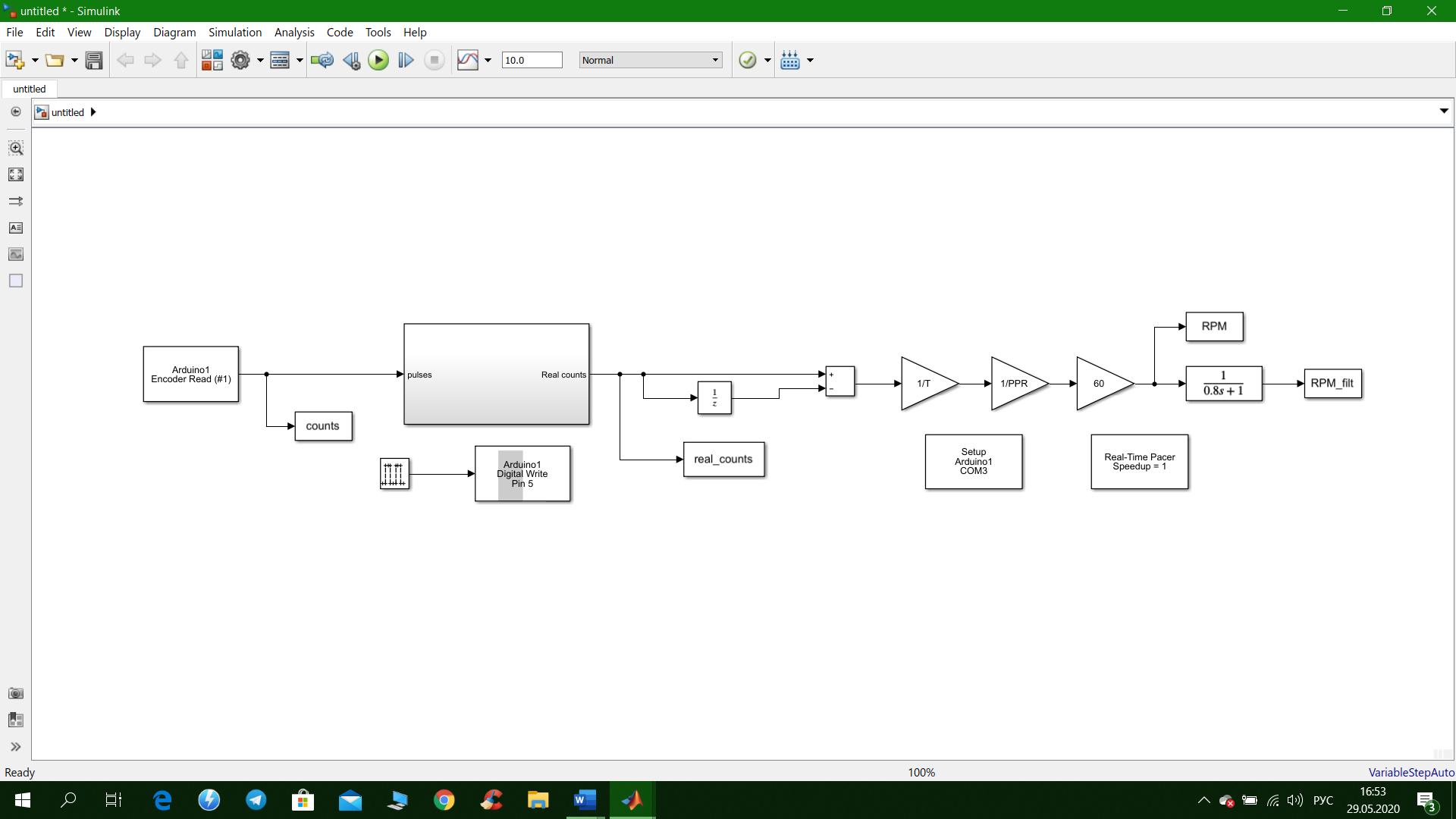
DC Motor speed plot looks like that (fig. 10):

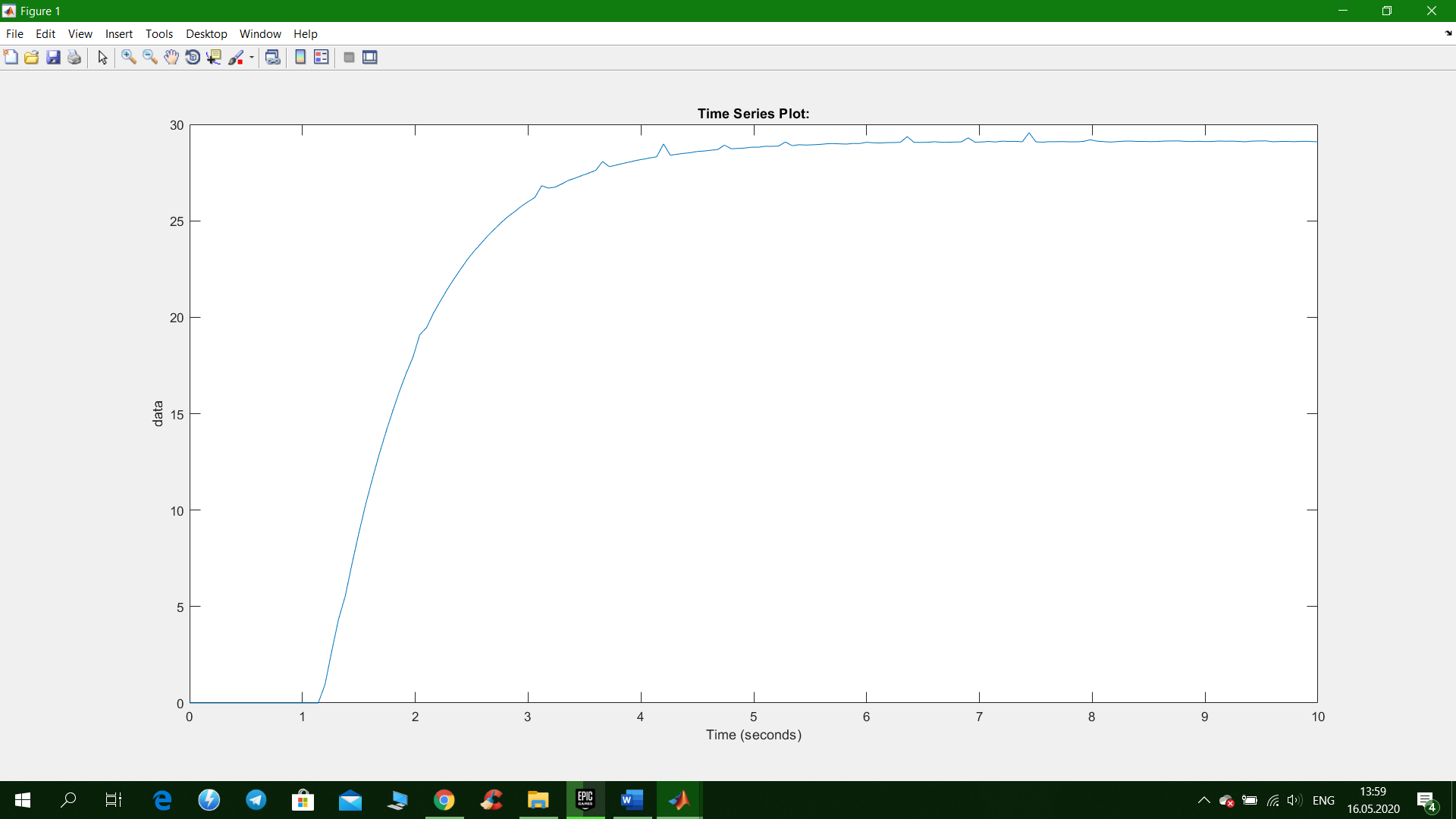


**Figure 10: Unfiltered RPM of DC motor**

As we can see, the plot looks noisy. It’s because of defects in wire connection, so periodically impulses are missed. Also, motor speed isn’t accurately 30 RPM, it may vary. In order to reduce this noise, I apply a common low-pass filter that can be represented as this transfer function:

Block diagram and RPM plot now look like that (fig. 11 and 12):

**Figure 10: Block diagram of filtered RPM as final output**



**Figure 12: Filtered RPM plot**

Now It’s much better: plot is quite smooth and we can see that RPM in not exactly 30, approximately 28-29 max. This result was accomplished by increasing sample time to 0.06s and using low-pass filter with 0.8 time constant. Using this time constant gives low response time: speed stabilizes after 5 seconds, but the system becomes more robust. I preferred robustness to fast response with lots of noise.