УДК 004.93’1 http://doi.org/10.15407/pp2023.02.010

S.O. Bezpalko, V.M. Shymkovysh, P.I. Kravets, A.O. Novatskyi, L.L. Shymkovysh, A.Yu. Doroshenko

SOFTWARE SYSTEM FOR LASER TARGETING DROPPED AMMUNITION

For last few years usage of small commercial unmanned aerial vehicles (UAV) on battlefield was drastically increased. In addition to use UAVs as spotter, they more and more frequently applied to drop small ammunition on enemy forces. In this paper laser targeting system that suitable for use on lightweight, mobile drones that drop freefalling ammunition was developed. Laser targeting systems are used for detecting and tracking the targets in the battlefield by obtaining the position information of these targets which is being designated and illuminated by UAV’s laser designator then the laser light reflected from these targets and come to the input of the optical systems of the dropped ammunition. An optical system for collimating the reflected beam from the target is one of the main components of the laser targeting system (LTS). The Quad Detector (QD) as a part of this optical system is simply consists of four photodiodes capable of detecting light spot projected on its surfaces and determine the deviation position of the laser spot from its center. It converts the incident laser spot to its corresponding photocurrent, the readout circuit that filter and convert the photocurrent to its corresponding voltage and the tracking system that is controlling the laser seeker movement to track the intended target based on the feedback information of the QD depending on the real position of the tracking platform. Developed system will ensure that munitions are accurately hit the target, reducing unnecessary casualties and damage, as well as reducing the number of sorties required and the time spent in hazardous airspace for the unmanned aerial vehicle.

Key words: laser targeting, microcontroller, unmanned aerial vehicles, regulators, hashtags, pulse width modulation, analog digital converter.

Introduction

In the modern world, one of the most pressing issues is ensuring the security of national borders and territories. Developing effective means of defense is a necessary step in ensuring national security [1-3]. One of the means of defense that is gaining popularity at present is the dropping of ordnance that can destroy a target from a great distance using lightweight and mobile unmanned aerial vehicles (UAVs). However, the accuracy of dropping ordnance and hitting the target remains a major challenge and is highly dependent on the skills and abilities of the operator of the device and weather conditions. Under these circumstances, the development of new technologies that can ensure the accuracy of dropping ordnance is particularly important.

In the civilian sector, freefalling guided objects control systems can be used in the delivery industry as a guidance system for postal drones. Additionally, these systems can be used in construction for the precise placement of large building elements or for assisting in the unloading of goods in warehouses.

This system will provide accurate dropping of ordnance and hitting the target, reducing the number of unnecessary casualties and damages, as well as reducing the number of necessary flights and time spent in hazardous airspace for unmanned aerial vehicles. The research results can be useful for military, law enforcement, and rescue services. In addition, the work can be used for further improvement of border and territory protection and safety systems.

Semi-Active Laser Homing (SALH) was chosen as the targeting method[4-8]. With SALH technique, a laser is directed at the target, and the laser radiation is reflected off the target and scattered in all directions (this is known as «painting the target» or «laser painting»). The missile, bomb, or other projectiles are launched or released somewhere close to the target. When they are close enough for a portion of the reflected laser radiation to reach them, a laser receiver determines the direction from which this radiation is coming and adjusts the trajectory of the projectile toward the source. As long as the projectile remains in the beam, its

trajectory will be corrected towards the target, providing a high degree of accuracy [9].

For this project infrared (IR) spectrum was chosen for laser and quad detector as it is commonly used for such applications. ATmega328p used as control device for its characteristics and availability. ATmega328p has six PWM channels and 6-channel 10-bit ADC, 2KBytes Internal SRAM, low power drain(only 0.2mA in active mode and 0.1μA in power-down mode)[10-15].

The aim of this work is to create laser targeting system suitable for usage on lightweight mobile UAVs.

1. Laser control

Laser technology, specifically light amplification by stimulated emission of radiation, has revolutionized the accuracy and effectiveness of modern weaponry. Laser systems have enabled military forces to target a wider range of objects with greater precision and with fewer munitions than previously possible. However, laser designators are susceptible to degradation caused by atmospheric scatter and various target reflections. To ensure effective use of laser designators and seekers, a pulse coding system is employed based on Pulse Repetition Frequency (PRF)[16].

Example [17] of Pulse Repetition Frequency signal is shown in Figure 1.

To generate pulse repetition frequency signal on IR laser we will use pulse-width modulation (PWM). To create PWM, the Atmega328 microcontroller has 3 built-in timers [10, 14, 15]: Timer0, Timer1, and Timer2. In this project, Timer 0 was used.

The Timer/Counter can operate on an internal clock pulse, through a frequency divider, or by using an external clock source on pin T0. The clock source selection logic block controls the source and edge that the Timer/Counter uses to increase (or decrease) its value. The Timer/Counter is inactive when no clock signal source is selected. The output of the clock source selection block is called the timer clock signal (clkT0) [14].

The dual buffer output compare registers (OCR0A and OCR0B) are continuously compared with the value of the Timer/Counter. The result of the comparison can be used by the waveform generator to create PWM or a variable output signal on the output compare pins (OC0A and OC0B). The output compare match event also sets the output compare flag (OCF0A or OCF0B), which can be used to generate an output compare interrupt request.

For our application frequency of PWM signal should be low enough so analog-to-digital converter can get some samples to be able to filter signal (with 8 prescaler its 153.8/4 kHz as we will read data from 4 sensors in QD). But high enough to be detectable and to not influence speed of reaction.

On Figure 2 we can see the setup of PWM for IR laser. This code is configuring PWM (Pulse Width Modulation) for an IR emitter on the Atmega328P microcontroller [18]. Note that _BV(bit) is defined as (1 << (bit)).

The code starts by resetting both timer/counters TCCR0A and TCCR0B to their default values of zero.

Then, it sets Pin PD5 to output mode by setting the corresponding bit in the DDRD register.

After that, it configures TCCR0A and TCCR0B registers to set the desired PWM mode and frequency. The TCCR0A register

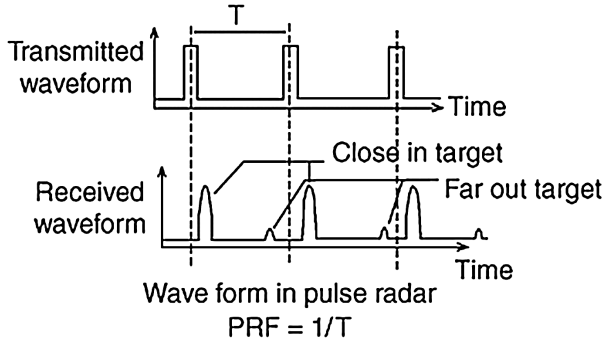

Figure 1. Pulse Repetition Frequency signal

```
// setup PWM for IR Emitter
// reset both timer/counters
TCCR0A = 0;
TCCR0B = 0;
// set PD5 to output
DDRD |= _BV(DDD5);

// TCCR0A [ COM0A1 COM0A0 COM0B1 COM0B0 0 0 WGM01 WGM00 ] = 0b10110011
TCCR0A = _BV(COM0A1) | _BV(COM0B1) | _BV(COM0B0) | _BV(WGM01) | _BV(WGM00);
// TCCR0B [ FOC2A FOC2B 0 0 WGM02 CS02 CS01 CS00 ] = 0b00000100
TCCR0B = _BV(CS02);

OCR0B = 127;
```

Figure 2. listing of PWM setup code

is set to _BV(COM0A1) | _BV(COM0B1) | _BV(COM0B0) | _BV(WGM01) | _BV(WGM00), which sets the waveform generation mode to fast PWM with top value OCR0A, and sets the Compare Output Mode to set OC0A/OC0B on Compare Match, clear OC0A/OC0B at BOTTOM (non-inverting mode). The default clock source for Timer 0 is the system clock. We can set the prescaler by modifying CS00, CS01, and CS02 in TCCR0B [10, 15, 19]. The TCCR0B register is set to _BV (CS02), which sets the clock prescaler of 256.

The PWM frequency for the output can be calculated by the following equation:

$$freq = f(clkIO) / (N * 256) \quad (1)$$

where:

- freq: the PWM frequency in Hz;
- f(clkIO): the clock frequency of the microcontroller in Hz, which is 16MHz in this case;
- N: the prescaler value used for the timer, which is 256 in this case

$$freq = \frac{16MHz}{256 * 256} = 244{,}140625\,Hz \quad (2)$$

Finally, the OCR0B register is set to 127, which sets the duty cycle of the PWM signal to 50% (half of the maximum value). This means the IR emitter will turn on for half of the PWM cycle and off for the other half.

Overall, this code sets up the Atmega328P to generate a ~0.244 kHz PWM signal with a 50% duty cycle on Pin PD5, which is connected to an IR emitter.

2. Reading data from photodiodes

While UAV emits infrared laser at target, dropped ammunition should correct its trajectory into target. Photodiode sensors located on ammunition receive reflection of IR laser from target we need to use analog-digital-transformation (ADC) to process this data and calculate vector to target.

To successfully process data, we should have fast ADC to get a lot of samples for filtering. Let's look at ADC in Atmel microcontrollers. Atmega328p features 10 bit ADC and provides 6 analog pins [10, 14, 15]; meaning you can connect up to 6 different analog devices to microcontroller, as we will connect IR quad sensor it will use 4 ADC pins and leave 2 pins for some other use.

On figure 3 we can see ADC setup on atmega328 for the project. For testing only one photodiode was used. To process data, a measurement was stored in 1024 byte array and then was send via serial port to computer. It was done this way to allow ADC work at full speed. To save space only 8 bits for each measurement was stored as it has enough precision to work with data and saves a lot of memory. This is important because Atmega328p has only 2kB internal memory.

First two lines clear the ADCSRA and ADCSRB registers to ensure that all bits are initialized to zero before setting any of them. The ADCSRA register controls the operation of the ADC, while the ADCSRB register controls some additional settings that are not used in this code.

```
// setup ADC for IR Sensor
ADCSRA = 0;             // clear ADCSRA register
ADCSRB = 0;             // clear ADCSRB register
ADMUX |= (0 & 0x07);    // set ADC0 analog input pin
ADMUX |= (1 << REFS0);  // set reference voltage
ADMUX |= (1 << ADLAR);  // left align ADC value to 8 bits from ADCH register
// sampling rate is [ADC clock] / [prescaler] / [conversion clock cycles]
// for Atmega328 ADC clock is 16 MHz and a normal conversion takes 13 clock cycles
ADCSRA |= (1 << ADPS1) | (1 << ADPS0);    // 8 prescaler for 153.8 KHz

ADCSRA |= (1 << ADATE); // enable auto trigger
ADCSRA |= (1 << ADIE);  // enable interrupts when measurement complete
ADCSRA |= (1 << ADEN);  // enable ADC
ADCSRA |= (1 << ADSC);  // start ADC measurements
```

Figure 3. listing of ADC setup code

The 3th line sets the input channel for the ADC to the ADC0 analog input pin of the microcontroller. The ADMUX register has several bits that select the input channel for the ADC, and this line sets those bits to 0x00 (which is equivalent to setting them to zero in this case). In the future more inputs will be used but for initial tests its good enough.

Next line sets the reference voltage for the ADC to the internal 5V reference voltage. The REFS0 bit of the ADMUX register is set to 1 to select the internal reference voltage. This means that the ADC will measure the input voltage relative to this 5V reference voltage. Therefore, using this formula, we can get voltage value on analog pin:

$$V_{in} = \frac{ADC * V_{REF}}{1024} \qquad (3)$$

Where VIN is the voltage on the selected input pin and VREF the selected voltage reference. In our case this formula will look like this, as VREF = 5V and we use only first 8 bits of ADC:

$$V_{in} = \frac{ADC * 5V}{255} \qquad (4)$$

To conveniently use only 8 bit data, next line sets the ADC result to be left-aligned in the ADCH register [20]. The ADLAR bit of the ADMUX register is set to 1 to left-align the result. This means that the 8 most significant bits of the ADC result will be stored in the ADCH register, and the 2 least significant bits will be stored in the 2 least significant bits of the ADCL register. In this case all we need to do to get data is read value in ADCH register.

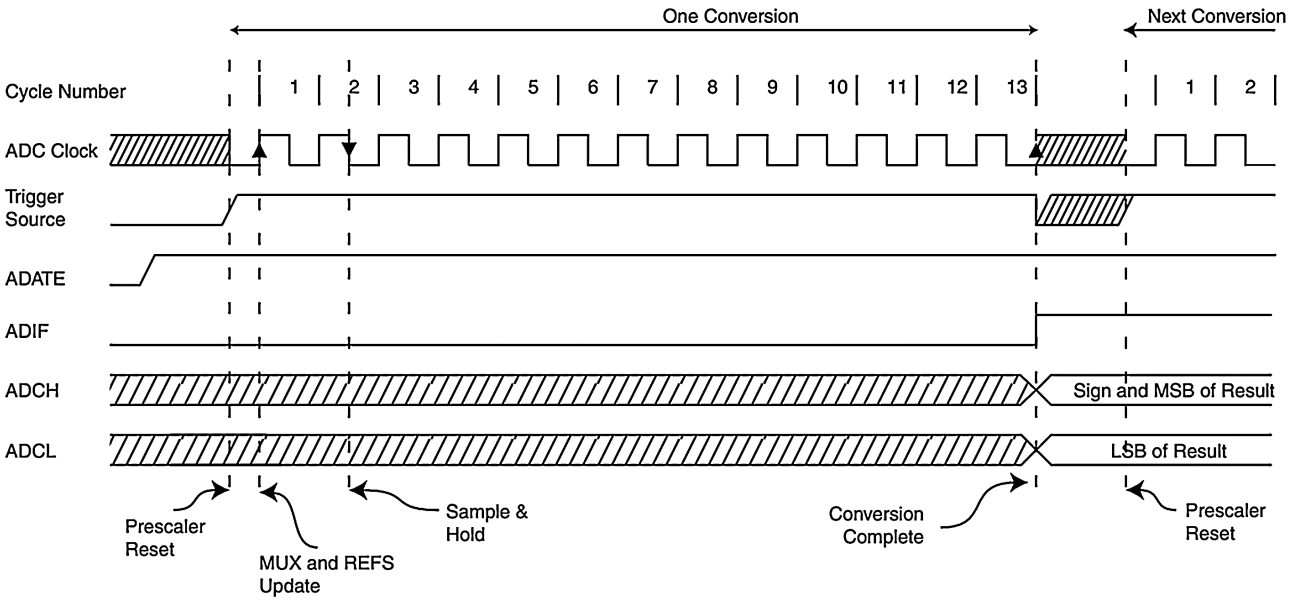

Figure 4. ADC Timing Diagram, Auto Triggered Conversion Figure 4. ADC Timing Diagram, Auto Triggered Conversion

As was said before, we need to get enough samples, so 6th line sets the ADC prescalervalue to 8, which means that the ADC clock frequency is divided by 8 before the analog signal is sampled. This line sets the ADPS1 and ADPS0 bits of the ADCSRA register to 1, which selects a prescaler value of 8. Unfortunately, we can't use lower prescaler as it would result in unreliable results. From datasheet [14] we know that one conversation takes 13 clock cycles (figure 4).

We can calculate sample rate of ADC with this formula:

$$SR = \frac{Cl_{ADC}}{\text{presc}}/13 = \frac{16\ MHz}{8 * 13} \approx 153.8\ KHz \quad (5)$$

The resulting sampling rate is 153.8 kHz, which is fast enough to capture the signal from an IR sensor. Next line enables auto-triggering of the ADC. The ADATE bit of the ADCSRA register is set to 1 to enable auto-triggering, which means that the ADC will start a new conversion automatically whenever a trigger event occurs. The trigger event can be a timer overflow, an external interrupt, or other events that are specified in the ADCSRB register (which is not used in this code).

To get data we use interrupts when a measurement is complete. The ADIE bit of the ADCSRA register is set to 1 to enable the ADC interrupt, which will trigger whenever a conversion is complete. The last 2 lines enables the ADC module and starts the ADC conversion process. The ADEN bit of the ADCSRA register is set to 1 to turn on the ADC module. The ADSC bit of the ADCSRA register is set to 1 to initiate a single conversion. Once the conversion is complete, the ADC interrupt will trigger to indicate that the result is ready. The next conversion will be triggered automatically by the auto-triggering feature, since ADATE was set to 1 earlier in the code.

Overall, this code sets up the ADC module of the ATmega328P microcontroller to read analog signals from the A0 input pin, with a reference voltage of 5V, a sampling rate of 153.8 kHz, and auto-triggering enabled. Interrupts are also enabled to handle the completion of each conversion. Once the setup is complete, the ADC module is enabled and a single conversion is initiated. Subsequent conversions will be triggered automatically by the auto-triggering feature. This code is typically used in applications that require high-speed analog-to-digital conversion, such as signal processing or sensing applications.

To verify that ADC is working correctly we can compare oscillogram of photodiode signal and output of Atmega328p. To filter high frequency noise from sensor data a simple resistor-capacitor (RC) filter was used.

To get data that microcontroller sends via serial port to computer PuTTY was used. It is a free and open-source terminal emulator, serial console and network file transfer application. It supports several network protocols, including SCP, SSH, Telnet, rlogin, and raw socket connection. It can also connect to a serial port [21]. After that data was proceed with Python script and plot was build using «matplotlib» [22]. We can see result on figure 5. Overall plot is similar to oscillogram, but we got a "cuts" every so often. They accrued because of using buffer to store data on microcontroller. Every time buffer is filled up we send data via serial port and in this time frame ADC turned off. This would not be a problem on final model as it won't use serial to send data. Otherwise, this problem can be reduced with more optimized serial code and with using 2 smaller buffers.

3. Data filter implementation

As was noted in chapter "Laser control" the UAV's microcontroller uses modulation on laser to make signal more recognizable. Therefore, on controllable ammunition side we need to read this signal and decode it. To implement this digital filter was used. A digital filter is a system that modifies or analyzes digital signals, such as audio or images, by removing or amplifying certain frequencies or components. It is implemented using digital signal processing techniques, where the input signal is sampled at discrete time intervals and the filter algorithm processes these samples to produce an output signal. They can be classified based on their transfer function, which defines how the filter modifies the

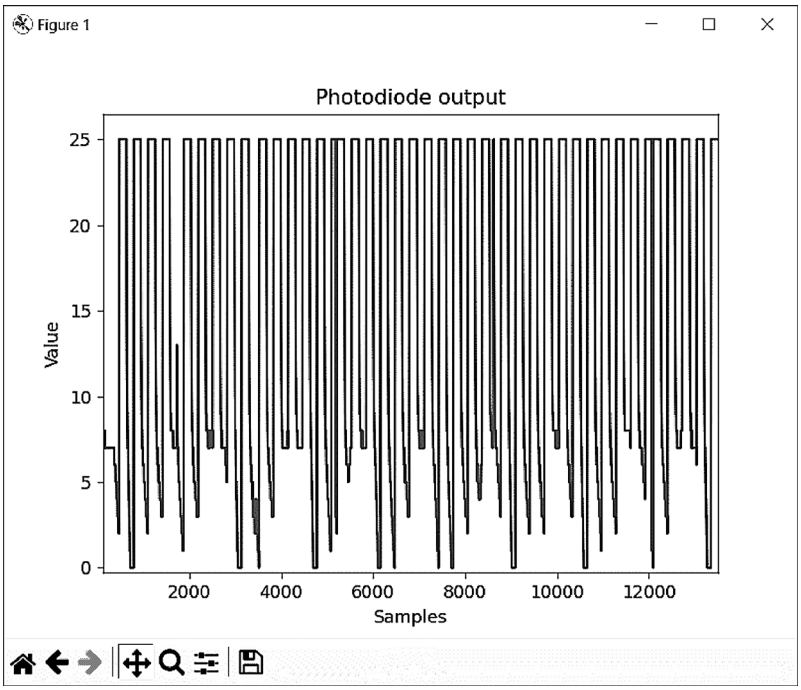

Figure 5. ADC output plot

input signal. Some common types of digital filters include finite impulse response (FIR) filters, infinite impulse response (IIR) filters, and digital notch filters. Digital filters have many advantages over their analog counterparts, including the ability to easily adjust filter parameters, reduced sensitivity to environmental noise, and the ability to implement complex filtering algorithms using software.

In this project passband Butterworth filter was used. Butterworth filter is a type of digital filter that is designed to have a flat frequency response in the passband and a monotonic attenuation in the stopband. The Butterworth filter belongs to the family of infinite impulse response (IIR) filters and is characterized by its smoothness and absence of ripples in the passband. It achieves [23] this by using a maximally flat magnitude response (figure 6), which means that the passband has a uniform gain and the cutoff frequency is defined as the frequency at which the magnitude response drops by 3 dB.

Butterworth filters can be designed for various filter orders, which determines the steepness of the transition between the passband and stopband. Higher order filters have steeper transitions but can suffer from passband ripple and phase distortion. In contrast, lower order filters have smoother responses but may not provide enough attenuation in the stopband.

The algorithm for finding the coefficients of a Butterworth filter involves several steps:

1. Determine the filter order: The filter order, denoted as n, determines the number of poles or zeros in the filter. It is determined based on the required cutoff frequency and the desired transition bandwidth. A higher-order filter provides a steeper transition but requires more complex calculations.

2. Calculate the analog prototype filter: The analog prototype filter is a normalized low-pass filter that serves as a reference for designing the Butterworth filter. The transfer function of the analog prototype filter is given by:

$$H(s) = \frac{1}{\sqrt{1 + \varepsilon\left(\frac{s}{\omega_c}\right)^{2n}}} \qquad (6)$$

where s is the complex frequency variable, ωc is the cutoff frequency, n is the filter order, and ε is a constant that determines the filter's ripple. For Butterworth filter, ε is set to 1.

3. Convert the analog prototype filter to a digital filter: The analog prototype filter is transformed into a digital filter using a bilinear transformation, which maps the s-plane of the analog filter to the z-plane of

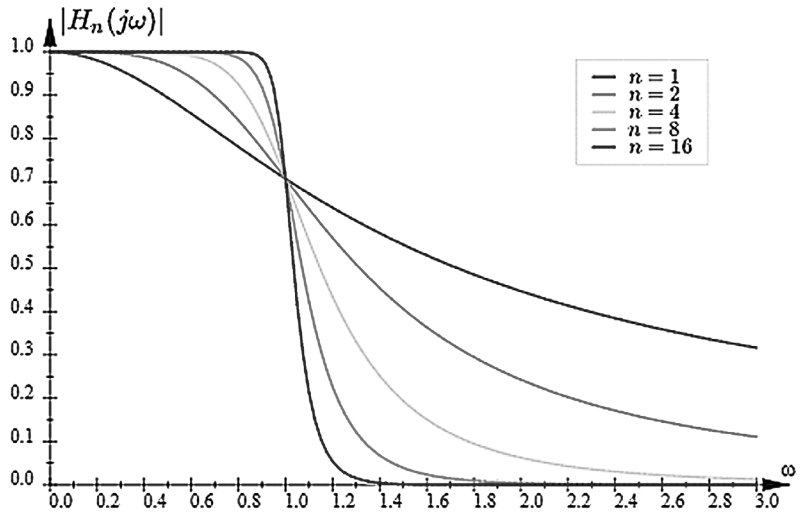

Figure 6. The frequency response plot for Butterworth filter with different order

the digital filter. The transfer function of the digital filter is given by:

$$H(z) = \frac{B(z)}{A(z)} \qquad (7)$$

where B(z) and A(z) are the numerator and denominator polynomials of the digital filter, respectively. These polynomials can be calculated using the following equations:

$$B(z) = \frac{(\omega_c)^n}{1 + \beta_1 z^{-1} + \beta_2 z^{-2} + \ldots + \beta_n z^{-n}} \qquad (8)$$

$$A(z) = 1 + \alpha_1 z^{-1} + \alpha_2 z^{-2} + \ldots + \alpha_n z^{-n} \qquad (9)$$

where βi and αi are coefficients that depend on the filter order and the bilinear transformation.

Normalize the digital filter coefficients: The coefficients of the digital filter are normalized by dividing them by A(0), which ensures that the DC gain of the filter is unity. The resulting coefficients are then used to implement the Butterworth filter.

To implement Butterworth filter, we need to calculate filter coefficients. For this task MATLAB [24] tools will be used. To calculate filter firstly we need to do spectral analysis of the input signal. For this purpose, was used fast Fourier transform [25] to perform frequency and power spectrum analysis of time-domain signals.

For this, firstly, we need to get data sample from microcontroller. As was mentioned before just straight forward dumping of ADC data is not ideal as it creates artifacts in output data. New highly efficient code was developed to get around this issue. To get data from Atmega328p to computer where it can be processed serial port was used [26].

This code is for an Atmega328P microcontroller, and it defines a data buffer, a counter, two offsets, and a flag to start serial data transfer. The program reads data from an analog-to-digital converter (ADC) and stores it in the buffer. When the buffer is full, the program swaps the offsets, so that the serial data transfer starts sending data from the just-populated buffer. The program then sets the start flag to true and sends the data in parts using the Serial.write() function. If the ADC filled the second buffer while the data transfer is in progress, the program swaps the offsets and starts sending data from the beginning of the buffer again.

The buffer is a continuous array of bytes, with a length of 2 * buffL, where buffL is a constant unsigned integer, which can be changed by user to any value that can be divided by 8. The program uses two separate buffers within this array, one for storing data from the ADC and another for sending data via serial. The two buffers are distinguished by their respective offsets, offsetW and offsetR. The program switches between the two buffers simply by changing the offset. If offset is 0, then data[offset + i] uses the first buffer, and if offset is buffL, then data[offset + i] uses the second buffer. This allows to do a great optimization of a code. By switching between

```
//Buffer size
const unsigned int buffL = 512;   //this value could be chosen larger or lover if desired
//here the data will be stored
//we need 2 separate buffer
//in first one we save data from ADC, and from second one we send data via Serial
//creating 1 continious array allows us to switch between first and second buffer just by changing offset
//if offset == 0, then data[offset + i] uses 1st buffer, if offset == buffL - 2nd
//therefore we can use add operator instead of if whick save us from branching and should work much faster
byte data[2 * buffL];
//Offset for Serial buffer
uint16_t offsetR = 0;
//Offset for ADC data buffer
uint16_t offsetW = buffL;
//Counter for ADC to populate data array
uint16_t counter = 0;
//Flag to start Serial data transfer
bool start = false;
//1/8 buffer used to send data via serial by parts
const uint16_t part = buffL / 8;
```

Figure 7. Global variables of data transfer code

offsets, we use simple numeric operations and avoid branching in code that influence program running speed very much. The part constant defines the size of the data sent in parts and is set to one-eighth of the buffer size, it used to transfer data via serial port in smaller parts what increase stability of output data.

The program uses an interrupt service routine (ISR) to read data from the ADC(figure 8). The ISR increments the counter and writes the data to the buffer. When the counter reaches the buffer size, the ISR swaps the offsets, sets the start flag to true, and triggers the serial data transfer. Tag [[unlikely]] used here to hint compiler that this branch is unlikely to happen, as it will occur only once at buffL times.

The main program loop (figure 9) checks whether the start flag is true. If so, it sends the data in parts using the Serial. write() function. Before sending the data, it resets the start flag to false, so we avoid sending same data more than one time. Start flag will be set by ADC interrupt when it will fill its buffer. If the ADC has filled the second buffer while the data transfer is in progress, the loop swaps the offsets and starts sending data from the beginning of the buffer again.

Overall, this part of the code reads data from an ADC and sends it via serial in parts using a continuous buffer that can be switched between the ADC buffer and the serial buffer. The program uses an interrupt to read data from the ADC and a loop to send data via serial. The program also handles buffer overflow and switching between buffers in a simple and efficient manner.

Using Putty [21] we can receive data from microcontroller and parse it into array of discrete data, figure 10. Let's use MATLAB to read data from log file and build graph, figure 11.

```
// ADC complete ISR
ISR(ADC_vect)
{
    //Write data from ADC to data array
    data[offsetW + counter++] = ADCH;
    //If ADC buffer is full
    if (counter == buffL) [[unlikely]]
    {
        //Clean counter
        counter = 0;
        //Swap offsets, so Serial now sends data from just populated buffer
        uint16_t temp = offsetW;
        //And ADC populates another buffer
        offsetW = offsetR;
        offsetR = temp;
        //Start Serial data transfer
        start = true;
    }
}
```

Figure 8. Interrupt service routine of ADC

```
void loop() {
    if (start)
    {
        //Set start to false to not send same data over and over
        start = false;
        //Send data in parts
        for (int i = 0; i < 8; i++)
        {
            uint16_t serOffset = offsetR + part * i;
            Serial.write(data + serOffset, part);
            //If ADC filled second buffer we need to swap and send data from the start
            if (start = true)
                break;
        }
    }
}
```

Figure 9. Main loop of the program

After getting signal in discrete array form we can analyze it using Fast Fourier Transformation (FFT). It is a mathematical algorithm used to transform time-domain signals into frequency-domain signals. It is a widely used technique in signal processing, image processing, and many other fields. The FFT algorithm is an efficient implementation of the Discrete Fourier Transform (DFT), which is a mathematical technique that converts a sequence of N complex numbers into another sequence of N complex numbers. The DFT is defined by a set of complex exponential functions, and it provides a way to analyze the frequency components of a signal, figure 12. Using MATLAB, we can easily perform FFT and build the power spectrum plot, figure 13. As frequency range we will use values from 0 to 10000. To make power spectrum plot more readable maximum value if power was limited by 1000.

We can see 2 main spikes on the start and end of the spectrum. A Fast Fourier Transformation (FFT) graph with two spikes, one at the start of the spectrum and the other at the end, typically indicates that there is a periodic component in the input signal. The spike at the start of the spectrum (i.e., at frequency 0) indicates the presence of a DC (direct current) component in the signal. A DC component represents the average value of the signal over time, and it is often caused by a bias or offset in the measurement system. The spike at the end of the spectrum (i.e., at the Nyquist frequency) indicates the presence of a high-frequency periodic component in the signal. The Nyquist frequency is equal to half of the sampling rate, and it represents the maximum frequency that can be accurately represented in the signal.

To get modulated signal we need to filter middle frequencies. To generate bandpass Butterworth filter, we can use MATLAB's "butter" function [24]. It takes as parameters needed order of filter, and relative cutoff frequencies.

Looking at received graph we can see that it nicely filters out data on carrier frequency. Its has longest spikes where background noise is almost zero and lover spikes where its high. To make this signal more readable we can take absolute value from it, multiply so it more relatable to input. To make filter more precise we also can increase order of a filter from 10 to 22.

Butter function returns a and b matrix that contain coefficients for filter. For an infinite impulse response (IIR) filter, the transfer function is not a polynomial, but a rational function. The Z-transforms of the input and output signals are related by

$$Y(z) = H(z)X(z) = \\ = \frac{b(1) + b(2)z^{-1} + \ldots + b(n+1)z^{-n}}{a(1) + a(2)z^{-1} + \ldots + a(m+1)z^{-m}} X(z) \quad (10)$$

where b(i) and a(i) are the filter coefficients. C++ code for Atmega328 was developed to implement digital filter that can have any order and be flexible for any coefficients. On figure 13 we can see implementation of filter.

This implementation (figure 16) defines a digital filter class called «Filter» which implements a finite impulse response (FIR) filter using an array of coefficients for

```
>> file = fopen("putty.log");
variable = fread(file);
fclose(file);
n = length(variable);
x = [1:n];
plot(x,variable)
>>
```

Figure 10. MATLAB command to read data from file and build graph

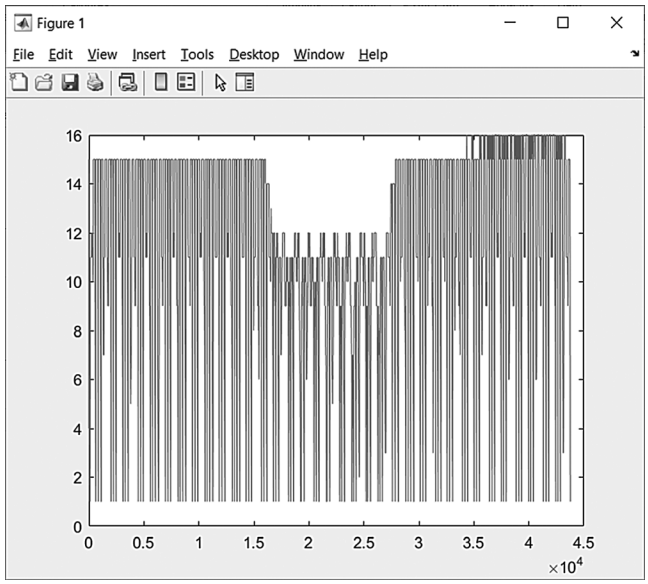

Figure 11. Plot of signal received from Atmega328

```
>> fu = fft(variable);
fs = 10000;
f = (0:n-1)*(fs/n);     % frequency range
power = abs(fu).^2/n;
power(power>1000) = 1000;
plot(f,power)
```

Figure 12. Performing FFT in MATLAB

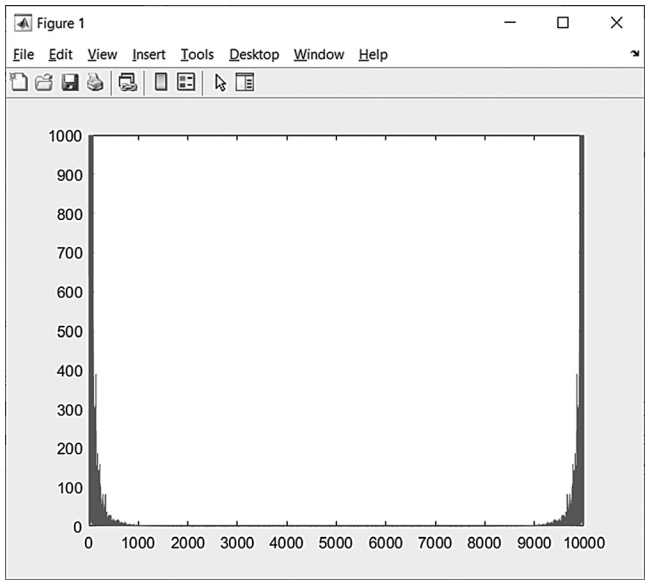

Figure 13. Power spectrum plot of received signal

```
>> [b,a] = butter(5,[300 400]/(1000));
y = filter(b,a,variable);
plot(x,y,x,variable)
```

Figure 14. Generating 10th order Butterworth bandpass filter

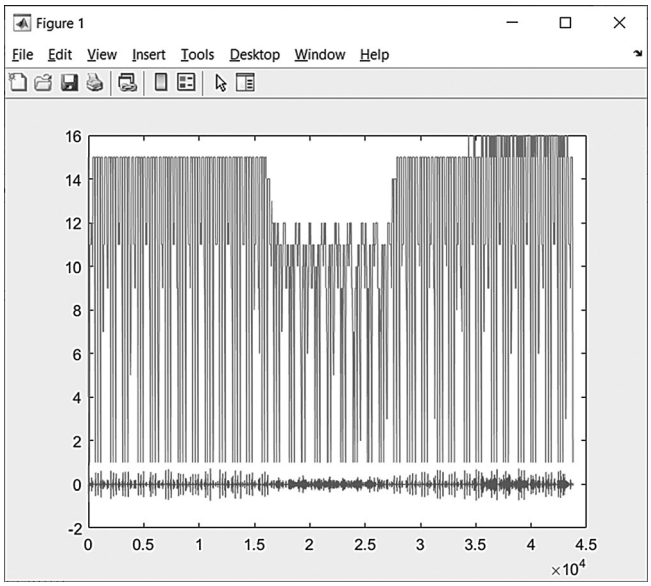

Figure 15. Filtering data result (bottom graph)

both the numerator (b) and denominator (a) of the transfer function. The filter has a constant order of 5, which means it can only process 5 samples at a time, but this parameter can be easily changed by user to any needed value.

The implementation uses an input array (x) to store the previous input samples and an output array (y) to store the previous output samples. The process() function takes in a new input sample, shifts all the existing samples in the input and output arrays by one position, and computes the output sample using the filter's transfer function. The output sample is then stored in the output array and returned as the output of the filter.

The filter's transfer function is implemented using a difference equation in the time domain. The output sample (yn) is computed as a weighted sum of the input samples (x) and previous output samples (y) using the filter's coefficients for both numerator (b) and denominator (a). This implementation assumes that all the coefficients have already been computed and assigned to the filter's coefficient arrays.

Note that the implementation uses the «memmove()» function to shift the input and output arrays, it is intended for use in a memory-constrained environment, but also it's a lot faster than manually do shift for each pair of inputs or outputs. Such saves allow microcontroller to proceed more data much faster.

4. System characteristics

Compiled code stats:

1. Program size: 3406 bytes (used 11% of a 30720 byte maximum)

2. Minimum Memory Usage: 1337 bytes (65% of a 2048 byte maximum).

Developed system characteristics:

1. ADC sampling frequency: 153.8 kHz for 1 sensor, 38.45 kHz for quad sensor;

2. Developed filter speed: 1000 samples processed in 60200 nanoseconds.

The results satisfy the technical requirements that was set before developing the system.

Conclusions

A In conclusion, the development of a laser targeting system for dropped ammunition using the Atmega328p microcontroller, PWM for IR signal modulation, fast ADC

```
class Filter {
public:
    static const byte order = 5;
    double a[order]; // a1 = a2 ... an = 0
    double b[order];
    byte x[order] = {};
    byte y[order] = {};

    byte process(byte sample)
    {
        memmove(&x[1], &x[0], (order - 1) * sizeof(byte));
        x[0] = sample;

        double yn;
        for (byte i = 0; i < order; i++) {
            yn += b[i] * x[i] - a[i] * y[i];
        }

        memmove(&y[1], &y[0], (order - 1) * sizeof(byte));
        y[0] = yn;
        return(yn);
    }
};
```

Figure 16. C++ implementation of FIR filter

for signal read from photodiode, and digital filtering techniques has been successfully implemented. The system demonstrated reliable and accurate targeting capabilities for dropped ammunition, which could be used in various military and civilian applications.

The use of the Atmega328p microcontroller provided a low-cost and efficient solution for controlling the system, while the PWM modulation allowed for precise control of the IR signal. The fast ADC was crucial for accurate signal reading from the photodiode, and the implemented digital filter improved the accuracy of the results.

Furthermore, the calculated Butterworth filter in MATLAB allowed for the optimization of the filter parameters and improved the overall performance of the system. The laser targeting system has the potential to be further improved and optimized for specific applications.

Overall, the laser targeting system for dropped ammunition presented in this paper provides a promising solution for accurate and reliable targeting, demonstrating the potential for future advancements in this area.

References

1. Bezpalko, S. O., Shymkovysh, V. M., and Doroshenko, A. Y. (2022) “A model and software for the inertial measurement unit”. PROBLEMS IN PROGRAMMING, (2), pp.3-12. http://doi.org/10.15407/pp2022.02.003
2. А.Ю. Дорошенко, В.М. Шимкович, В.О. Федоренко. (2018) “Програмні засоби моделювання системи управління векторною тягою реактивного двигуна” Проблеми програмування. № 2-3. с. 296-304.http://dspace.nbuv.gov.ua/handle/123456789/144641
3. Shymkovych, V., Telenyk, S. andKravets, P. (2021) “Hardwareimplementationofradial-basisneuralnetworkswithgaussianactivationfunctionson FPGA,” Neural Computing and Applications, 33(15), pp. 9467–9479. https://doi.org/10.1007/s00521-021-05706-3.
4. Syed Affan Ahmed, Mujahid Mohsin, Syed Muhammad Zubair Ali. (2021) “Survey and technological analysis of laser and its defense applications”. Defence Technology, 17(2), pp. 583-592. https://doi.org/10.1016/j.dt.2020.02.012.
5. H. Kaushal and G. Kaddoum, (2017) “Applications of Lasers for Tactical Military Operations”IEEE Access, vol. 5, pp. 20736-20753 https://doi.org/10.1109/ACCESS.2017.2755678
6. Zhichao Wu, Le Liu, Xiuli Zhang, (2020) ”Study on large area laser reflective precision target”,Optik, vol.224,165730. https://doi.org/10.1016/j.ijleo.2020.165730.
7. Padarev, Nikolai. (2022) “Guide lines for improving laser targeting device inmilitary”,

Technology transfer: fundamental principles and innovative technical solutions. pp. 38-40. https://doi.org/10.21303/2585-6847.2022.002681.
8. H. Kaushal and G. Kaddoum, (2017) "Applications of Lasers for Tactical Military Operations",IEEE Access, vol. 5, pp. 20736-20753,.https://doi.org/10.1109/ACCESS.2017.2755678
9. «Spot Leading Target Laser Guidance for Engaging Moving Targets».U.S. Patent 8,237,095.JohnPardue(2005)
10. Інформаційно-керуючі системи. Локальні інформаційно-керуючі системи. Лабораторний практикум [Електронний ресурс] : навчальний посібник для здобувачів ступеня бакалавра за освітньою програмою «Інтегровані інформаційні системи» спеціальності 126 «Інформаційні системи та технології» / П. І. Кравець, В. М. Шимкович, Ю. М. Бердник; – Київ : КПІ ім. Ігоря Сікорського, 2022. – 142 с. https://ela.kpi.ua/handle/123456789/47956
11. Shymkovych, Volodymyr, Anatoliy Doroshenko, Tural Mamedov, and Olena Yatsenko (2022)»Automated Design of an Artificial Neuron for Field-Programmable Gate Arrays Based on an Algebra-Algorithmic Approach", International Scientific Technical Journal «Problems of Control and Informatics»67(5), pp.61-72. https://doi.org/10.34229/2786-6505-2022-5-6.
12. P. Kravets, V. Nevolko, V. Shymkovych and L. Shymkovych (2020) «Synthesis of High-Speed Neuro-Fuzzy-ControllersBased on FPGA»,2020 IEEE 2nd International Conference on Advanced Trends in Information Theory (ATIT), Kyiv, Ukraine, pp. 291-295. https://doi.org/10.1109/ATIT50783.2020.9349299
13. Shymkovych, V., and Niechkina, V. (2020). "The criterion for determining the buffering time of the measuring channel for smoothing the variable changes of the sensor signal", In2020 IEEE 7th International Conference on Energy Smart Systems (ESS).pp. 343-346.
14. "ATmega48A/PA/88A/PA/168A/PA/328/PmegaAVR® Data Sheet". Microchip Technology Inc. (2020)
15. Новацький, А. О. Проектування вбудованих систем. Лабораторний практикум [Електронний ресурс] : навч. посіб. для студ., які навчаються за освітньою програмою «Інтегровані інформаційні системи» за спеціальністю 126 «Інформаційні системи та технології» / А. О. Новацький, В. М. Шимкович – Київ: КПІ ім. Ігоря Сікорського, 2022. – 463 с. https://ela.kpi.ua/handle/123456789/48130
16. V. E. Clark, Joint Tactics, Techniques, and Procedures for Laser Designation Operations,United States Joint Chiefs of Staff, May 1999, pp. 149
17. Aviation dictionary, pulse repetition frequency URL: https://aviation_dictionary.en-academic.com/5406/pulse_repetition_frequency(датазвернення: 01.04.2023)
18. Pandanom/IRGuidanceIR guiding system source code – URL: https://github.com/Pandanom/IRGuidance (датазвернення: 11.04.2023).
19. MEAM.Design : ATmega32U4 : Timers/Counters : Timer 0– URL: https://medesign.seas.upenn.edu/index.php/Guides/MaEvArM-timer0 (дата звернення: 08.04.2023).
20. Новацький, А. О. Мікропроцесорні та мікроконтролерні системи. Частина 2. Проектування мікропроцесорних систем [Електронний ресурс] : підручник для студентів освітньої програми «Інтегровані інформаційні системи» за спеціальністю 126 «Інформаційні системи та технології» / А. О. Новацький – Київ : КПІ ім. Ігоря Сікорського, 2021. – 462 с. https://ela.kpi.ua/handle/123456789/43051
21. PuTTYFAQ – URL: https://www.chiark.greenend.org.uk/~sgtatham/putty/faq.html#faq-meaning (дата звернення: 09.04.2023).
22. Matplotlib main website – URL: https://matplotlib.org/stable/devel/index (дата звернення: 09.04.2023).
23. Derivation of the transfer functions of low-pass and high-passButterworth Filters by Pieter P– URL:https://tttapa.github.io/Pages/Mathematics/Systems-and-Control-Theory/Analog-Filters/Butterworth-Filters.html (дата звернення: 10.04.2023).
24. MathWorks Butterworth filter design – URL: https://www.mathworks.com/help/signal/ref/butter.html (дата звернення: 10.04.2023)
25. MathWorks Basic Spectral Analysis– URL:

https://www.mathworks.com/help/matlab/math/basic-spectral-analysis.html (дата звернення: 10.04.2023)
26. Fast serial communication with Arduino– URL: https://curiousscientist.tech/blog/fast-serial-communication-with-arduino (дата звернення: 11.04.2023)

Received 25.04.2023

Про авторів:

Безпалько Станіслав Олегович,
студент 6 курсу
Національного Технічного Університету України «КПІ імені Ігоря Сікорського»
https://orcid.org/0000-0001-8550-1792

Шимкович Володимир Миколайович,
кандидат технічних наук,
доцент кафедри інформаційних систем та технологій
Національного технічного університету України «КПІ імені Ігоря Сікорського».
Кількість наукових публікацій в українських виданнях – понад 30.
Кількість наукових публікацій в зарубіжних виданнях – понад 10.
Індекс Хірша – 4. https://orcid.org/0000-0003-4014-2786

Новацький Анатолій Олександрович,
кандидат технічних наук,
доцент кафедри інформаційних систем та технологій
Національного технічного університету України «КПІ імені Ігоря Сікорського».
Кількість наукових публікацій в українських виданнях – понад 30.

Кравець Петро Іванович,
кандидат технічних наук,
доцент кафедри інформаційних систем та технологій
Національного технічного університету України «КПІ імені Ігоря Сікорського».
Кількість наукових публікацій в українських виданнях – понад 40.
Кількість наукових публікацій в зарубіжних виданнях – понад 10.
Індекс Хірша – 4. https://orcid.org/0000-0003-4632-9832

Шимкович Любов Леонідівна,
асистент кафедри інформаційних систем та технологій
Національного технічного університету України «КПІ імені Ігоря Сікорського».
Кількість наукових публікацій в українських виданнях – 2.
Кількість наукових публікацій в зарубіжних виданнях – 1.
https://orcid.org/0000-0002-1291-0373

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор, завідувач відділу теорії комп'ютерних обчислень,
професор кафедри інформаційних систем та технологій
Національного технічного університету України «КПІ імені Ігоря Сікорського».
Кількість наукових публікацій в українських виданнях – понад 200.
Кількість наукових публікацій в зарубіжних виданнях – понад 90.
Індекс Хірша – 6. http://orcid.org/0000-0002-8435-1451

Місце роботи авторів:

Національний технічний університет України «Київський політехнічний інститутімені Ігоря Сікорського», проспект Берестейський 37 та Інститут програмних систем НАН України, 03187, м. Київ, проспект Академіка Глушкова, 40.
Тел.: (044) 526 3559
E-mail:
stas110922@gmail.com,
v.shymkovych@kpi.ua,
a.novatskyi@kpi.ua,
peter_kravets@yahoo.com,
L.shymkovych@gmail.com,
doroshenkoanatoliy2@gmail.com