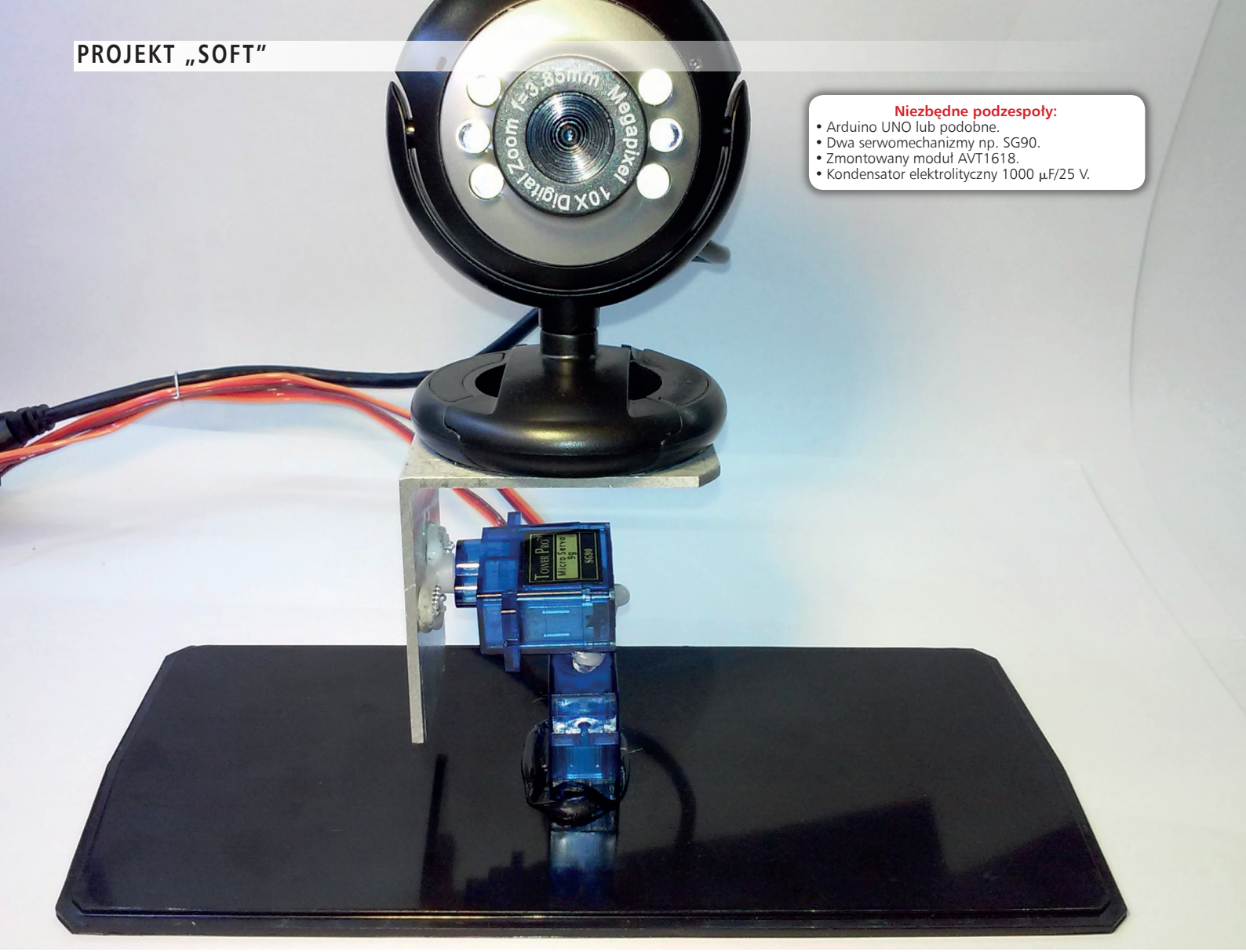



**Niezbędne podzespoły:**
- Arduino UNO lub podobne.
- Dwa serwomechanizmy np. SG90.
- Zmontowany moduł AVT1618.
- Kondensator elektrolityczny 1000 µF/25 V.

# Arduino w roli sterownika pozycjonera kamery

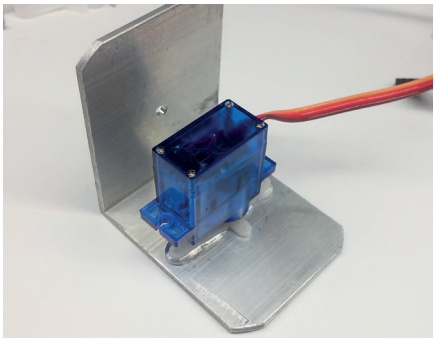
*Wiele drogich kamer do systemów monitorowania jest wyposażonych w zamontowany fabrycznie napęd realizujący funkcje pan/tilt/zoom. W podobny napęd, niestety bez zoomu optycznego, możemy wyposażyć dowolną kamerę. W artykule opisano manipulator do kamery lub aparatu wykonany na bazie gotowych komponentów. Jest to rodzaj statywu ustawianego w dwóch płaszczyznach, sterowanego joystickiem, mającego możliwość zaprogramowania sekwencji ruchów.*

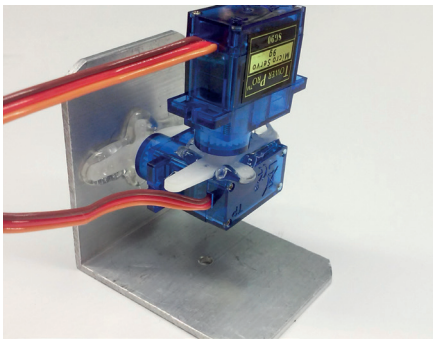
Część mechaniczną wykonano z dwóch serwomechanizmów połączonych ze sobą (sklejonych) jak na **fotografii 1**. Jeden odpowiada za ruch w płaszczyźnie poziomej, drugi w płaszczyźnie pionowej. W trakcie sklejania należy zwrócić uwagę czy serwomechanizmy mogą swobodnie poruszać się w całym zakresie, czy nie są blokowane przez pozostałe elementy mechaniczne oraz czy są równo ustawione. Do serwomechnizmów należy dołączyć duży, płaski element, który będzie pełnił rolę podstawki lub elementu mocującego oraz np. metalowy kątownik, na którym zostanie zamocowana sama kamera.

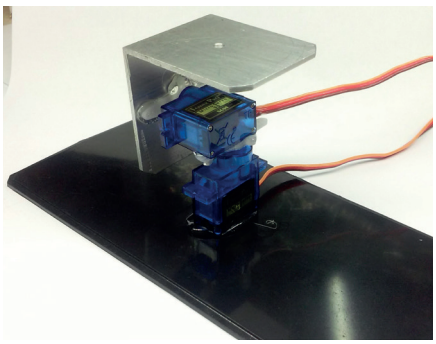
## Serwomechanizm

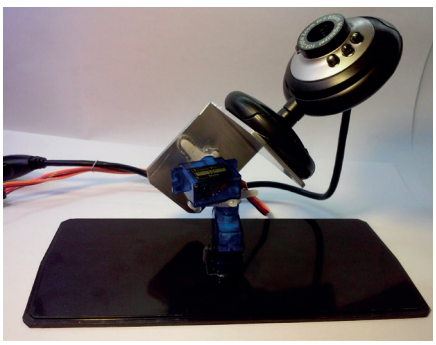
Serwomechanizm to silnik z przekładnią i układem sterującym skonfigurowanymi w taki sposób, aby umożliwić ruch osi w pewnym zakresie, najczęściej 180 stopni. Serwomechanizm jest kontrolowany za pomocą przebiegu impulsowego o ściśle określonych parametrach. Położenie osi jest proporcjonalne do czasu trwania impulsu sterującego, co pokazano na **rysunku 2**.

Większość serwomechanizmów wymaga zasilania napięciem z zakresu 4,5…6 V. W stanie spoczynku pobiera niewielki prąd rzędu kilkudziesięciu miliamperów. W czasie przemieszczania się lub gdy ramie jest obciążone (nawet nie musi być w ruchu), silnik napędu może pobierać znaczny prąd, nawet kilka amperów. Dlatego bardzo ważne jest

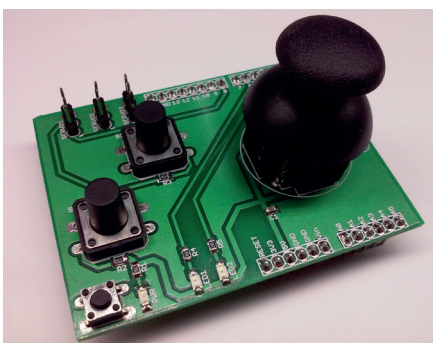




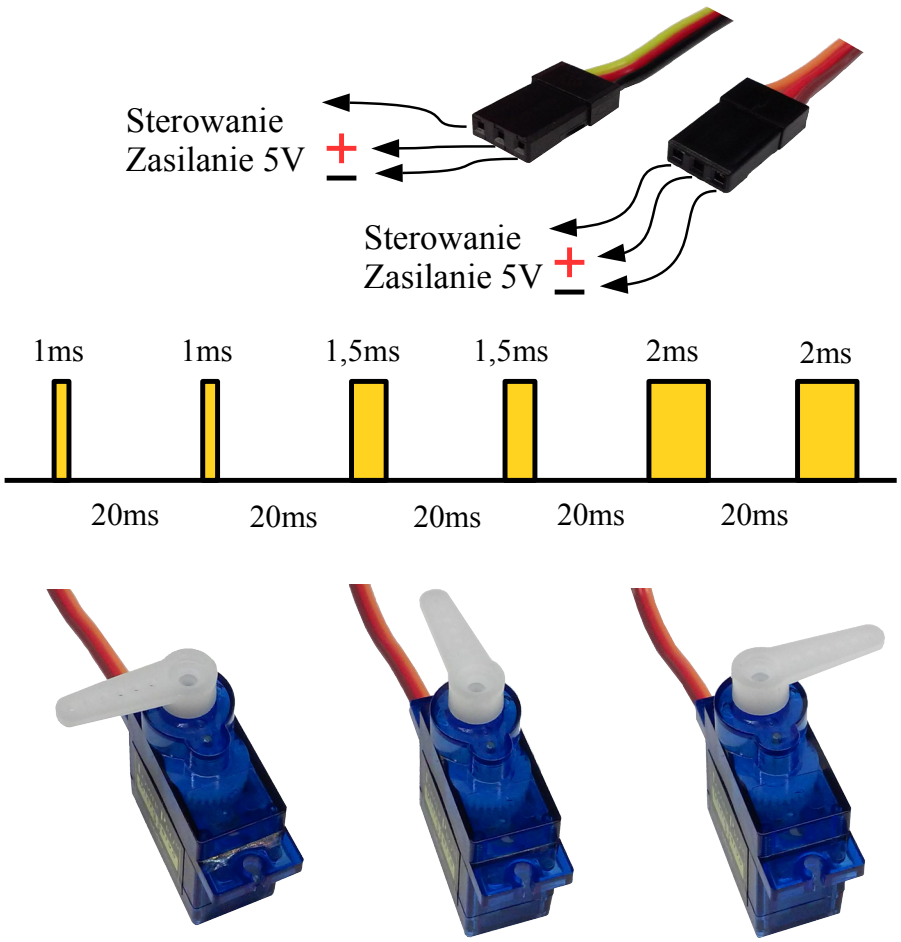




**Fotografia 1. Sposób przyklejenia serwomechanizmów: a) serwomechanizm przyklejony do uchwytu kamery, b) sposób sklejenia dwóch serwomechanizmów, c) przyklejenie zespołu serwomechanizmów do podstawy, d) sposób zamocowania kamery**



**Fotografia 3. Moduł ArduinoUno**




**Rysunek 2. Złącze oraz impulsy sterujące serwomechanizmem**

zapewnienie odpowiednio wydajnego zasilania, a najlepszym rozwiązaniem jest zastosowanie oddzielnego zasilacza specjalnie dla serwomechanizmu. W proponowanym rozwiązaniu zastosowano serwomechanizmy modelarskie klasy „micro", a współpracująca kamera jest lekka, więc nie użyto odrębnego zasilacza. Całość należy jednak zasilać z zasilacza, a nie z portu USB, który może mieć zbyt małą wydajność prądową oraz dołączyć dodatkowy kondensator filtrujący zasilanie o dużej pojemności.

## Panel sterujący

Jako panel sterujący zastosowano pokazany na **fotografii 3** moduł AVT1618, który zawiera następujące elementy umożliwiające wygodną manipulację kamerą: joystick, dwa przyciski, dwie diody LED oraz wyprowadzone złącza dla serwomechanizmów.

Głównym elementem panelu sterującego manipulatora jest joystick analogowy, zbliżonych budową do stosowanych w padach do konsoli PS. Drążek joysticka jest sprzęgnięty z osiami dwóch potencjometrów umieszczonych prostopadle, a ruch drążka zmienia położenie suwaków tych potencjometrów. Jeśli do potencjometrów doprowadzimy zasilanie, to na wyjściach otrzymamy dwa napięcia, jedno odpowiadające położeniu na umownej osi x, a drugie na umownej osi y. Zasadę działania oraz wygląd panelu pokazano na **rysunku 4**.

## Kontroler

„Mózgiem" urządzenia jest płytka Arduino Uno. Ma ona tę zaletę, że kupiona u sprawdzonego dostawcy, na pewno działa i nie trzeba marnować czasu na jej uruchomienie. Sposób połączenia poszczególnych komponentów pokazano na **fotografii 5**. Gdyby fotografie były niewystarczające, to można posiłkować się schematem zamieszczonym na **rysunku 6**.

## Oprogramowanie

Oprogramowanie sterujące manipulatorem zostało wykonane w środowisku Arduino IDE. Do wygenerowania impulsów sterujących serwomechanizmem należałoby użyć

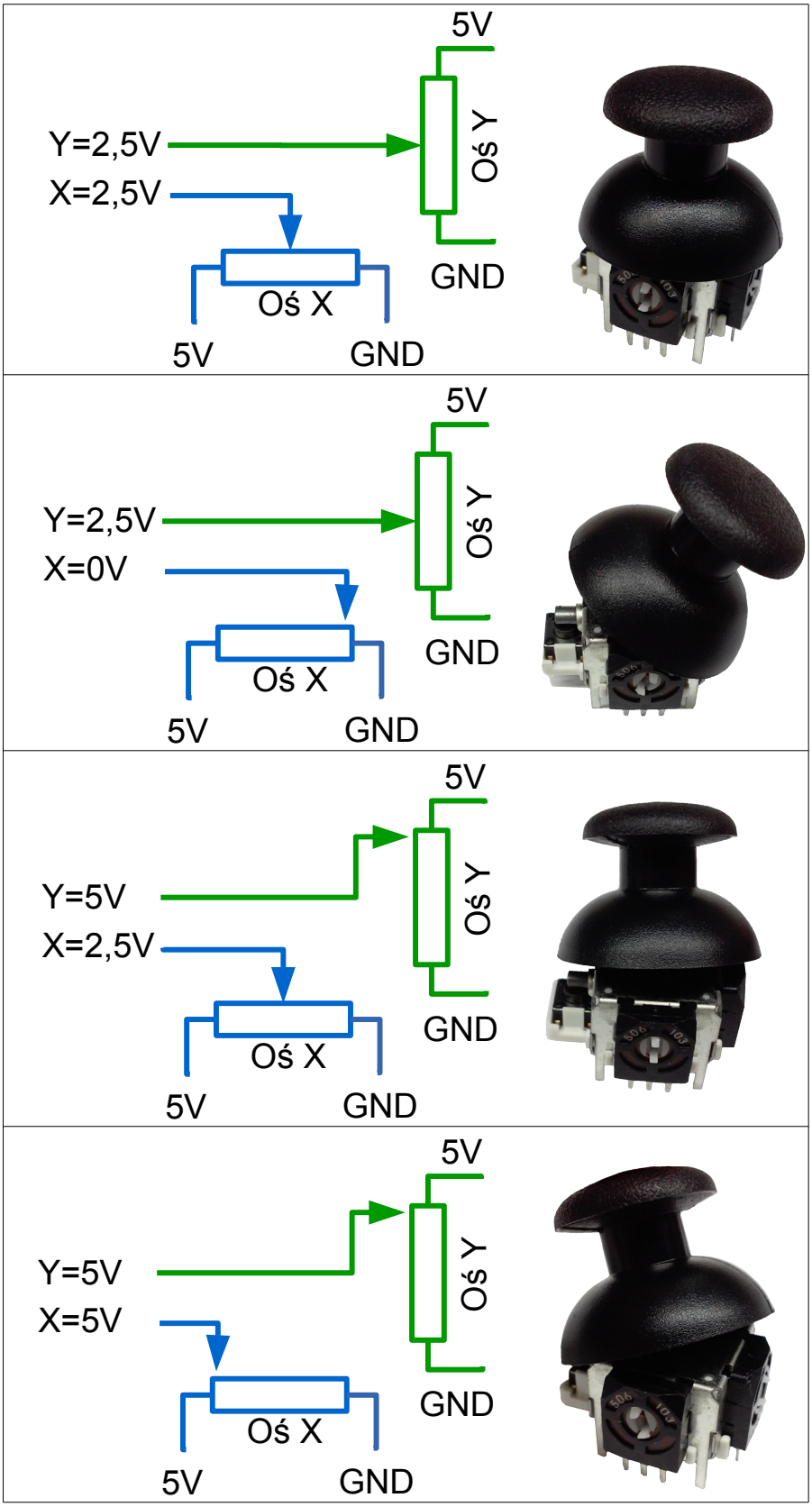
timera zawartego w peryferiach procesora i uruchomić odpowiednie przerwania, ale Adruino ma gotowe rozwiązanie – jest nim biblioteka *servo.h*, którą dołączymy do projektu. Odczyt położenia suwaków potencjometrów joysticka zostanie wykonany za pomocą funkcji *analogRead*.

W celu przetestowania pracy płytki tworzymy nieskomplikowany program, jak na **listingu 1**. Kompilujemy i przesyłamy do płytki Arduino.

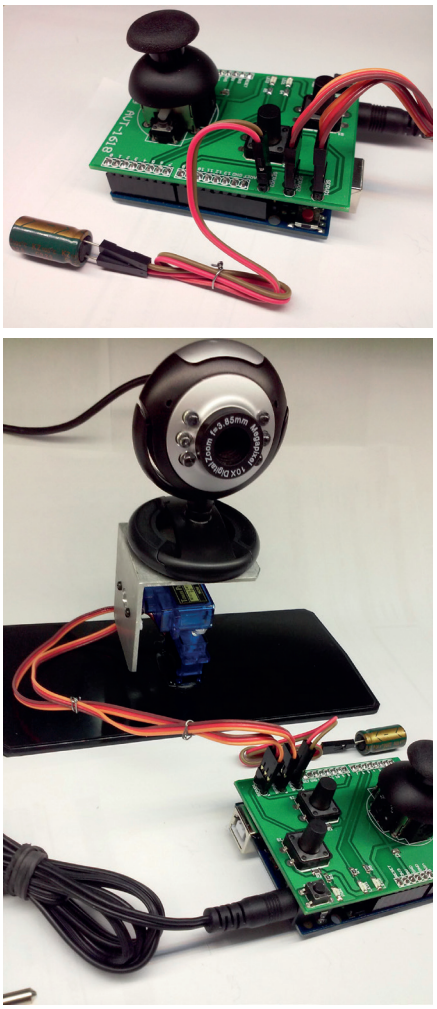
Po załadowaniu programu urządzenie działa, ale można zauważyć, że ruch serwomechanizmów jest gwałtowny i szarpany, zwłaszcza przy szybszym ruchu joysticka. Dzieje się tak, ponieważ serwomechanizmy dążą do jak najszybszego osiągnięcia zadanego położenia. Może się zdarzyć, że płytka „zawiesi się” – to właśnie efekt gwałtownego poboru prądu przez szybko pracujące serwomechanizmy. W kolejnym programie „zmusimy” je do nieco wolniejszej pracy, dzięki czemu ruch stanie się bardziej płynny i unikniemy zawieszania się płytki. Zmieniamy nieco główną pętlę programu, jak pokazano **listingu 2**.

Po załadowaniu programu z listingu 2, widzimy, że teraz serwomechanizmy pracują dużo wolniej, ich szybkość możemy regulować wartością opóźnienia na końcu głównej pętli programu. Ponadto, zamieniliśmy parametry polecenia *map*, dzięki czemu ruch w płaszczyźnie x jest zgodny z ruchem joysticka.

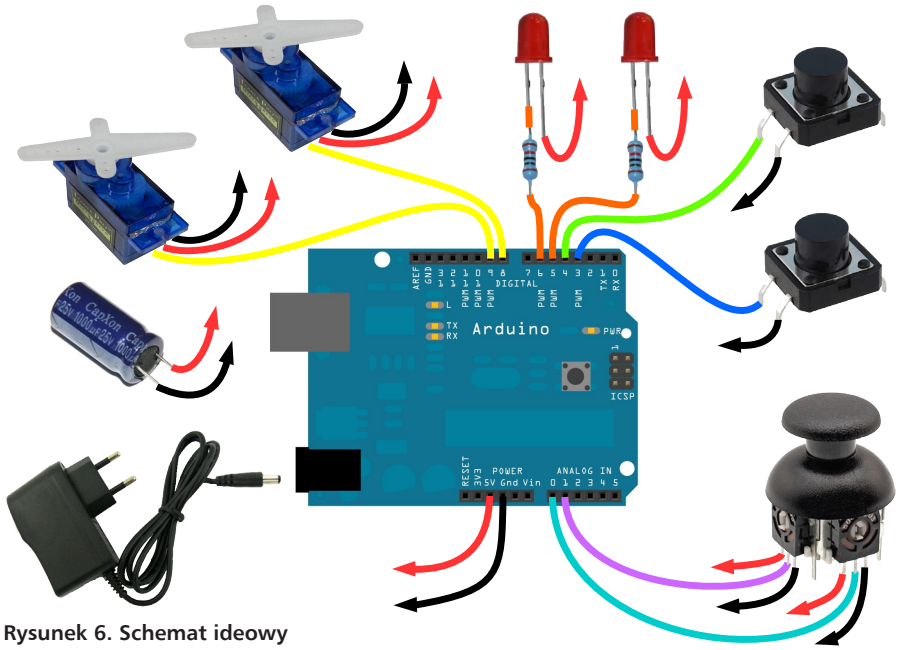
Na koniec sprawimy by nasz manipulator sam wykonywał zapamiętane sekwencje ruchów, co może przydać się do monitorowania obiektów. Do tego celu przede wszystkim będzie potrzebny bardziej rozbudowany interfejs użytkownika, do którego dodamy obsługę dwóch przycisków i diod LED.




**Rysunek 4. Działanie joysticka analogowego**



**Fotografia 5. Sposób przyłączenia płytki ArduinoUno: a) widok złącza zasilania i serwomechanizmów, b) widok mechanizmu i sterownika**




Rysunek 6. Schemat ideowy

**Listing 1. Odczyt wartości analogowych i ustawianie serwomechanizmów – wersja uproszczona**

```
#include <Servo.h>    //dołączenie biblioteki do obsługi serw
Servo servox;        //tworzymy obiekt kontrolujacy serwo 1
Servo servoy;        //tworzymy obiekt kontrolujacy serwo 2
int servox_pin = 8;  //sterowanie serwa 1 na wyprowadzeniu 8
int servoy_pin = 9;  //sterowanie serwa 2 na wyprowadzeniu 9
int joyx_pin = 0;    //sygnał z osi x joysticka na wyprowadzeniu 0
int joyy_pin = 1;    //sygnał z osi y joysticka na wyprowadzeniu 1
int analog_x;        //wychylenie osi x
int analog_y;        //wychylenie osi y
int degree_x;        //położenie serwa osi x
int degree_y;        //położenie serwa osi y

void setup()
{
    servox.attach(servox_pin);  //serwo x przypisane do wyprowadzenia 8
    servoy.attach(servoy_pin);  //serwo y przypisane do wyprowadzenia 9
    degree_x = 90;     //położenie początkowe serva osi x
    degree_y = 90;     //położenie początkowe serva osi y
}

void loop()
{
    //odczyt położenia joystika,
    //odczyt wartosci analogowych
    analog_x = analogRead(joyx_pin);
    analog_y = analogRead(joyy_pin);
    //przeskalowanie wartosci analogowych
    //na wartość kąta odchylenia serwomechanizmów
    degree_x = map(analog_x, 0, 1023, 0, 180);
    degree_y = map(analog_y, 0, 1023, 0, 180);
    //ustawienie serwomechanizmów
    servox.write(degree_x);
    servoy.write(degree_y);
    //przerwa
    delay(15);
}
```

**Listing 2. Zmodyfikowany program – spowolnienie pracy serwomechanizmów**

```
void loop()
{
    int temp;  //zmienna pomocnicza
    //odczyt położenia joystika,
    //odczyt wartosci analogowych
    analog_x = analogRead(joyx_pin);
    analog_y = analogRead(joyy_pin);
    //przeskalowanie wartosci analogowej
    //na wartość kąta odchylenia serwomechanizmów
    //oraz odwrócenie ruchu - teraz podąża za joystickiem
    temp = map(analog_x, 0, 1023, 180, 0);
    //powolne dochodzenie do zadanej wartości kąta x
    //o 1 jeden stopień przy każdym przejściu pętli
    if (degree_x < temp) degree_x++;
    if (degree_x > temp) degree_x--;
    //przeskalowanie wartosci analogowej
    //na wartość kąta odchylenia serwomechanizmów
    temp = map(analog_y, 0, 1023, 0, 180);
    //powolne dochodzenie do zadanej wartości kata y
    //o 1 jeden stopień przy każdym przejściu pętli
    if (degree_y < temp) degree_y++;
    if (degree_y > temp) degree_y--;
    //ustawienie serwomechanizmów
    servox.write(degree_x);
    servoy.write(degree_y);
    //przerwa, ustala szybkość ruchu serw
    delay(30);
}
```

**Listing 3. Funkcje *ButtonCheck()* oraz *LedState()***

```
//sprawdza stan przycisków,
//zwraca wartość tylko w momencie przyciśnięcia
int ButtonCheck(void)
{
    //zmienne statyczne,
    //zapamiętują ostatni stan przycisków
    static int button1_state;
    static int button2_state;
    int result = 0;

    if (digitalRead(button1_pin) == LOW)
    {
        //jeśli przycisk 1 był zwolniony to zwraca wartość
1
        if (button1_state == 0) result = 1;
        button1_state = 1;
    }
    else button1_state = 0;

    if (digitalRead(button2_pin) == LOW)
    {
        //jeśli przycisk 2 był zwolniony to zwraca wartość
2
        if (button2_state == 0) result = 2;
        button2_state = 1;
    }
    else button2_state = 0;
    return result;
}

//zaświeca diody LED w zależności od stanu urządzenia
void LedState(int pos, int go)
{
    if (pos > 0)  //w czasie programowania pozycji świeci
LED1
          digitalWrite(led1_pin, LOW);
    else
        digitalWrite(led1_pin, HIGH);
    if (go > 0)  //w czasie odtwarzania sekwencji świeci
LED2
        digitalWrite(led2_pin, LOW);
  else
        digitalWrite(led2_pin, HIGH);
}
```

**Listing 4. Pętla główna programu po modyfikacjach**

```
void loop()
{
    int temp;  //zmienna pomocnicza
    //odczyt położenia joystika,
    //odczyt wartosci analogowych
    analog_x = analogRead(joyx_pin);
    analog_y = analogRead(joyy_pin);
    //przeskalowanie wartosci analogowej
    //na wartość kąta odchylenia serwomechanizmów
    //oraz odwrócenie ruchu - teraz podąża za joystickiem
    temp = map(analog_x, 0, 1023, 180, 0);
    //gdy funkcja replay to odtwazra zapisane położenia
    if (pos_go > 0) temp = pos_x[pos_step];
    //powolne dochodzenie do zadanej wartości kąta x
    //o 1 jeden stopień przy każdym przejściu pętli
    if (degree_x < temp) degree_x++;
    if (degree_x > temp) degree_x--;
    //przeskalowanie wartosci analogowej
    //na wartość kąta odchylenia serwomechanizmów
    temp = map(analog_y, 0, 1023, 0, 180);
    //gdy funkcja replay to odtwazra zapisane położenia
    if (pos_go > 0) temp = pos_y[pos_step];
    //powolne dochodzenie do zadanej wartości kata y
    //o 1 jeden stopień przy każdym przejściu pętli
    if (degree_y < temp) degree_y++;
    if (degree_y > temp) degree_y--;
    //ustawienie serwomechanizmów
    servox.write(degree_x);
    servoy.write(degree_y);
    //przerwa, ustala szybkość ruchu serw
    delay(30);
    //sprawdzenie przycisków
    temp = ButtonCheck();
    //przycisk 1 - zapamietywanie położenia
    if (temp == 1)
    {
        PositionSet(pos_step, degree_x, degree_y);
        pos_step++;
        if (pos_step >= 4) pos_step = 0;
    }
    //przycisk 2 - włączanie funkcji replay
    if (temp == 2)
    {
        if (pos_go == 0) pos_go = 1;
        else pos_go = 0;
    }
    //steruje diodami obrazując stan urządzenia
    LedState(pos_step, pos_go);
    //automatycznie przełącza zapisane pozycje
    // po określonej liczbie cykli
    PositionNext(150);
}
```

Do programu dopisujemy linie kodu definiujące zmienne określające, do których wyprowadzeń są dołączone komponenty:

```
int led1_pin = 6;
int led2_pin = 5;
int button1_pin = 3;
int button2_pin = 4;
```

W sekcji *setup()* dopisujemy konfigurację nowych wyprowadzeń:

```
pinMode(led1_pin, OUTPUT);
pinMode(led2_pin, OUTPUT);
pinMode(button1_pin,
INPUT_PULLUP);
pinMode(button2_pin,
INPUT_PULLUP);
```

Tworzymy funkcje *ButtonCheck()* oraz *LedState()* obsługujące nowe komponenty – zamieszczono je na **listingu 3**.

Teraz potrzebujemy dodatkowych zmiennych, które zapamiętają sekwencje. Wykorzystamy do tego tablice oraz dwie zmienne pomocnicze:

```
int pos_go;      //funkcja replay
- gdy 1 to odtwarza sekwencje
int pos_step;    //numer położenia
int pos_x[4];    //tablica położeń
na płaszczyźnie poziomej
int pos_y[4];    //tablica położeń
na płaszczyźnie pionowej
```

Ostatecznie główna pętla programu będzie wyglądała jak na **listingu 4**.

Procedury odpowiedzialne za wywoływanie sekwencji ruchów pokazano na **listingu 5**. *PositionSet()* służy do zapamiętania aktualnych wartości położenia x i y w tablicach sekwencji, natomiast *PositionNext()* służy do przełączania kolejnych sekwencji po określonym czasie, po wykonaniu ostatniej sekwencji powraca do pierwszej.

Po załadowaniu programu za pomocą joysticka sterujemy położeniem statywu. Gdy naciśniemy przycisk „1”, to zostaje zapamiętane aktualne położenie, jako pierwsze położenie w sekwencji oraz zaświeci się dioda LED1 sygnalizując, że należy wprowadzić kolejne pozycje. Po wprowadzeniu czterech pozycji dioda LED1 zgaśnie. Teraz, przyciskając przycisk 2, uruchamiamy odtwarzanie sekwencji. Statyw będzie ustawiał się po kolei w czterech wcześniej ustawionych pozycjach.

Pełne źródło programu dostępne jest w materiałach dodatkowych do tego artykułu.

**KS**

**Listing 5. Procedury *PositionSet()* oraz *PositionNext()***

```
//zapamiętuje położenie x i y w tablicach
void PositionSet(int pos_num, int x, int y)
{
    if (pos_step >= 4) pos_step = 0;
    pos_x[pos_num] = degree_x;
    pos_y[pos_num] = degree_y;
}

//przełącza numer sekwencji po określonym czasie trwan
void PositionNext(int how_long)
{
    static int counter;
    if (pos_go > 0)
    {
        counter++;
        if (counter >= how_long)   //następna pozycja
        {
            counter = 0;
            pos_step++;
            //jeśli to ostatnia to zaczyna od początku
            if (pos_step >= 4) pos_step = 0;
        }
    }
    else
        counter = 0;
}
```