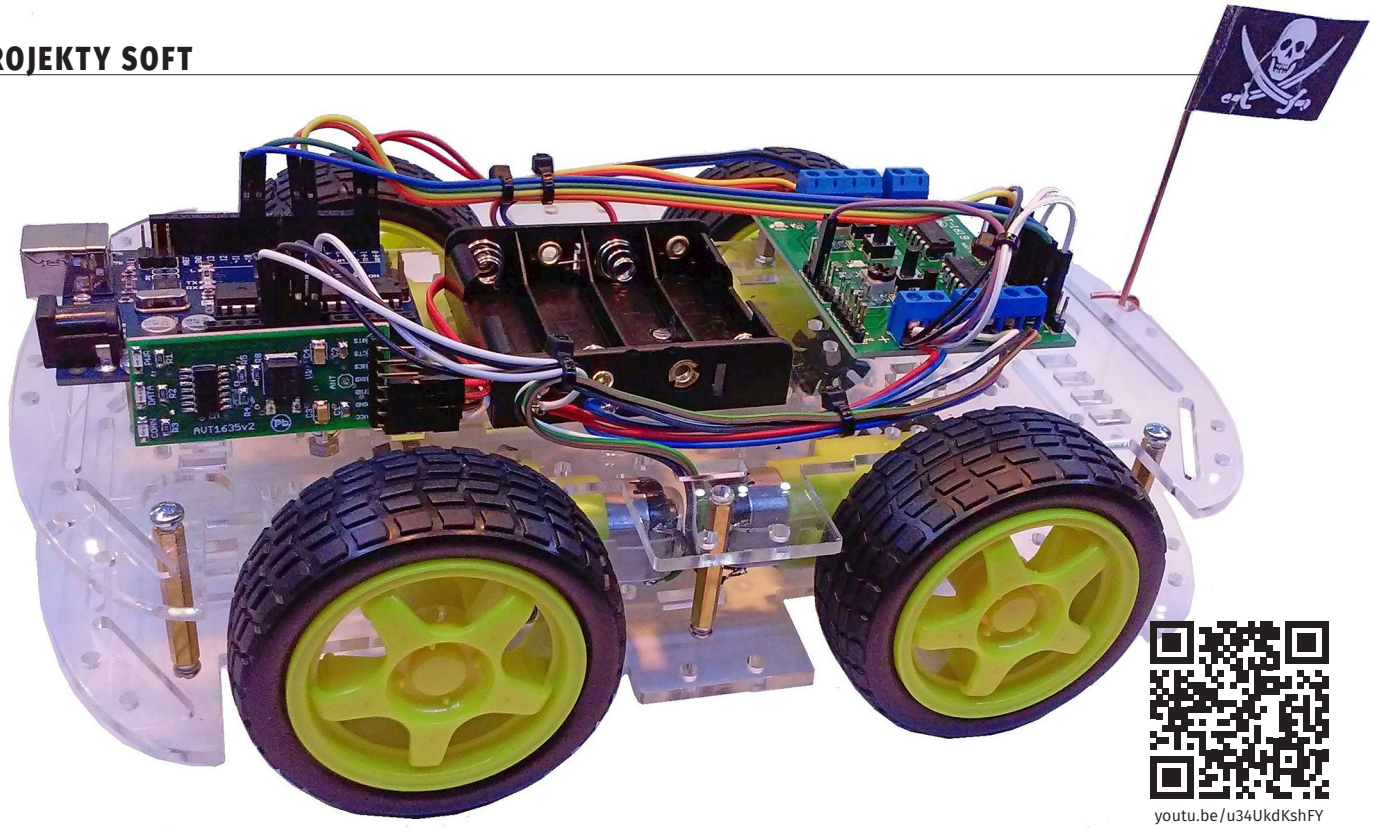


youtu.be/u34UkdKshFY

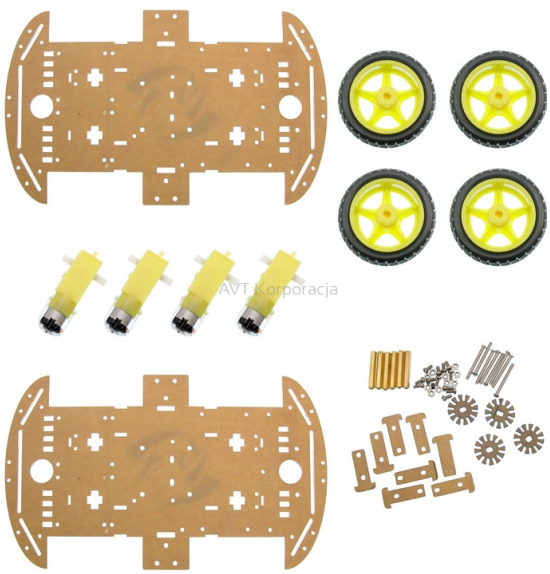
Pojazd zdalnie sterowany

Smartfon często bywa używany w roli platformy do gier. Sterowanie wirtualnym pojazdem (np. w grze) poprzez wychylanie i obracanie urządzeniem, jak kierownicą jest bardzo intuicyjne i stało się pewnym standardem. Jednak nieporównywalnie więcej zabawy przynosi sterowanie pojazdem w rzeczywistości. Ten łatwy w budowie pojazd pozwoli przekonać się o tym na własnej skórze.

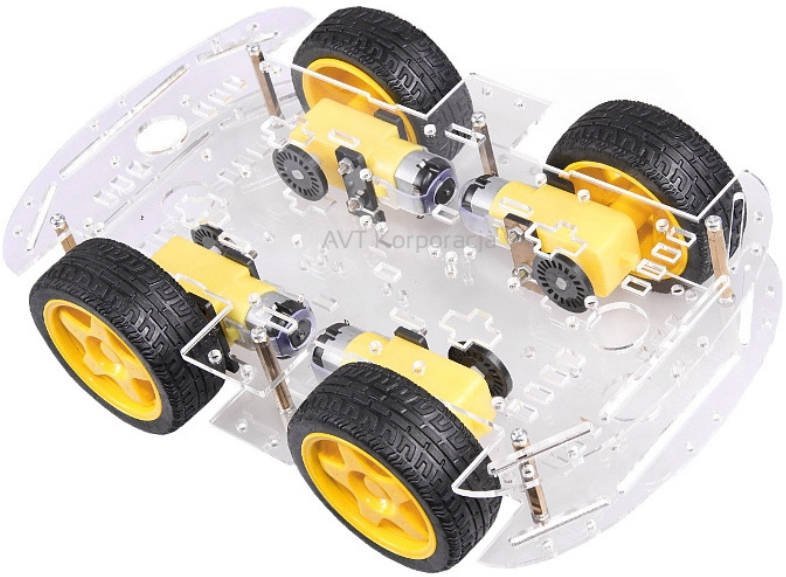
Niezbędne komponenty:
- Płytka Arduino Uno.
- Moduł sterownika silników dla Arduino – płytka AVT1619.
- Moduł Bluetooth AVT1635.
- Koszyk na baterie 4×R6.
- Dioda prostownicza np. 1N4007.
- Kondensator 2200 µF/10 V.
- Przewody połączeniowe.

Do budowy pojazdu użyto gotową platformę jezdną, dzięki czemu część mechaniczna jest prosta w wykonaniu, nie będziemy tracili czasu na wykonywanie elementów pojazdu, a montaż sprowadza się do zamocowania za pomocą śrub gotowych elementów. Platformę przed i po zmontowaniu pokazano na **fotografiach 1** i **2**.

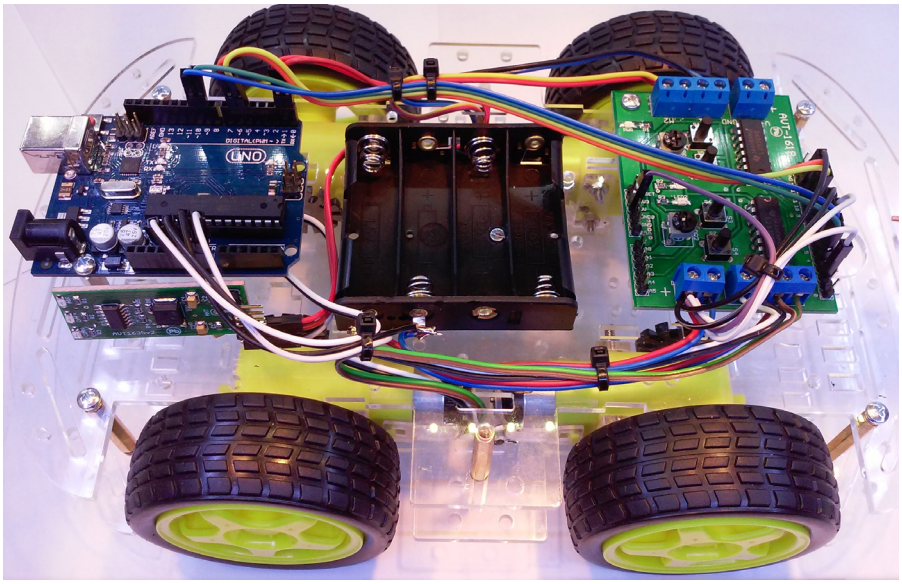
Platforma ma wiele otworów montażowych, dzięki czemu można łatwo zamontować dodatkowy osprzęt używając śrub i gwintowane tulejki dystansujące.



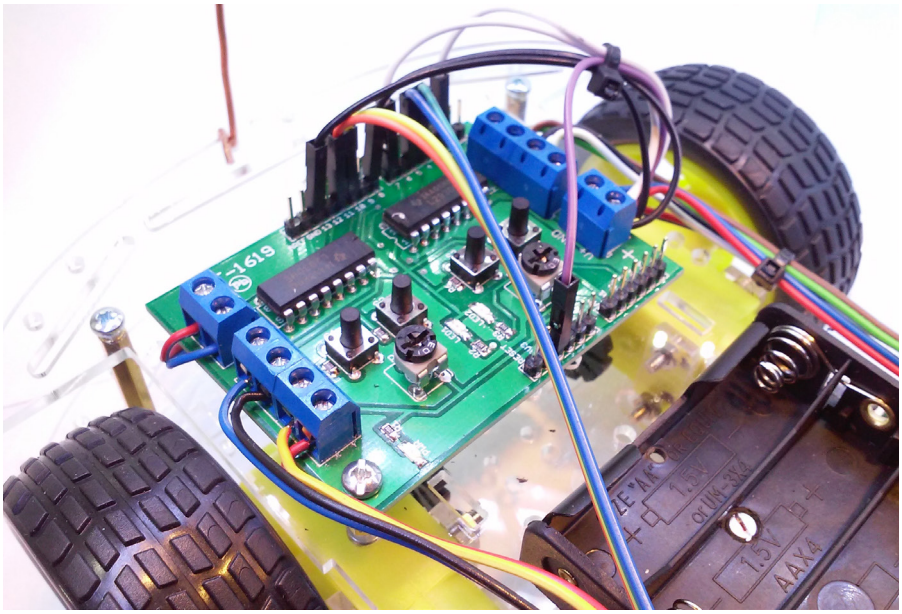
Fotografia 1. Części składowe pojazdu – platformy



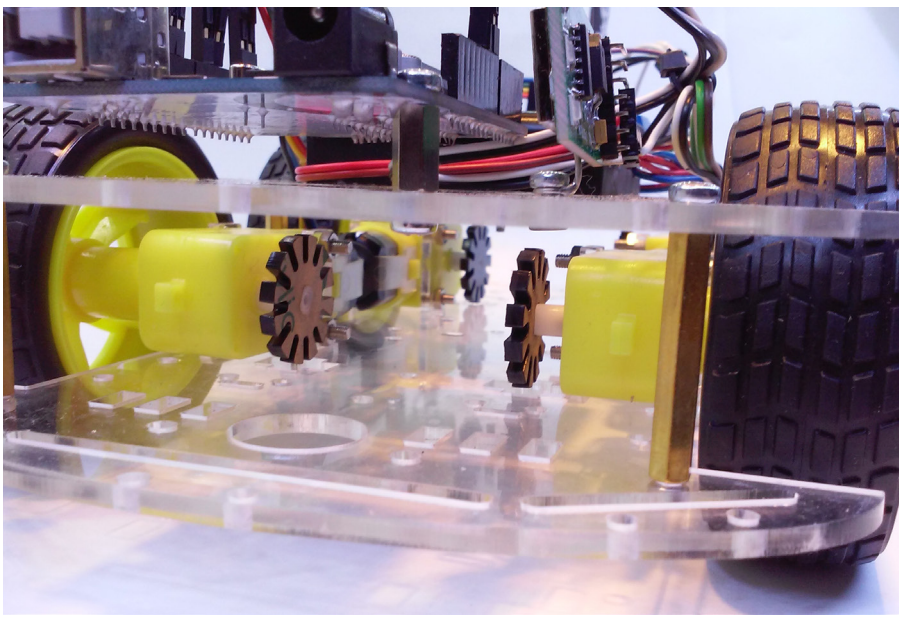
Fotografia 2. Pojazd po zmontowaniu



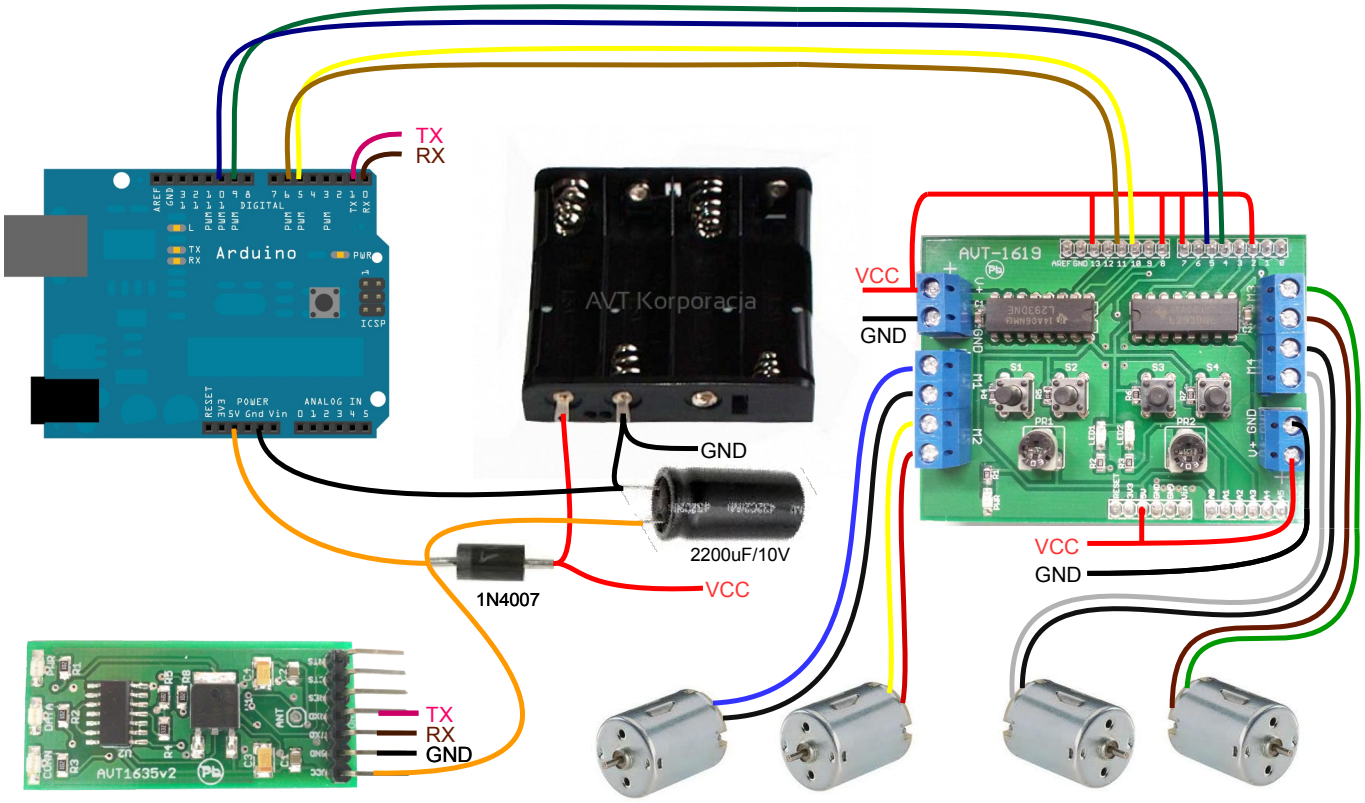
Fotografia 3. Szczegóły montażu pojazdu – widok od góry



Fotografia 4. Szczegóły montażu pojazdu – widok zamontowanej płytki Arduino



Fotografia 5. Szczegóły montażu pojazdu – widok napędu i sposobu montażu mechanicznego



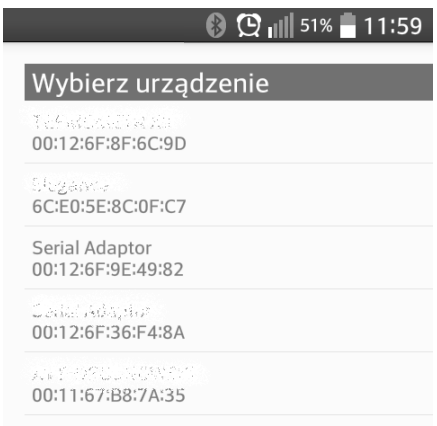
Rysunek 6. Schemat montażowy zdalnie sterowanego pojazdu

Listing 1. Procedura parsująca komendy

```
//-------------------------------------------------
//czekamy aż będą 2 znaki nowej linii
if (blueComplete >= 2) {
    if (blueBuffer.charAt(1) == 'x') //sprawdza czy na poz. 1 jest x
    {
        blueStatus = 1;  //prawidłowa komenda - status = 1
        statusDelay = 0;  //zeruje timer braku komendy
        i = 2;  //odczyt wartości od poz. 2
        vectX = 0;  //na początek zeruje wartość X
        data = blueBuffer.charAt(i);
        if (data == '-') {  //sprzawdza czy wartość ujemna
            minus = -1;
            i++;
        } else minus = 1;
        data = blueBuffer.charAt(i);
        if ((data >= '0') && (data <= '9'))  //czy znak to cyfra?
        {
            vectX += (data - '0');
            i++;
            data = blueBuffer.charAt(i);  //kolejny znak
            if ((data >= '0') && (data <= '9'))  //czy cyfra?
            {
                vectX *= 10;  //oblicza wartość dwucefrową
                vectX += (data - '0');
                i++;
            }
        }
        vectX *= minus;  //uwzglednia minus
        vectY = 0;  //na początek zeruje wartość Y
    .
    .
    .
    //wartość vectY oblicza analogicznie jak vectX//
    blueBuffer = "";  //czyści bufor
    blueComplete = 0;  //zeruje znacznk odebranych danych
    vect2pwm(vectY, vectX);  //oblicza pwm
}
```

Listing 2. Procedura powodująca automatyczne zatrzymanie pojazdu

```
//-----------------------------------
if (statusDelay <= 100) statusDelay++;
else blueStatus = 0;
if (statusDelay == 100) {
    vect2pwm(0, 0);
    blueBuffer = "";
    blueComplete = 0;
}
```

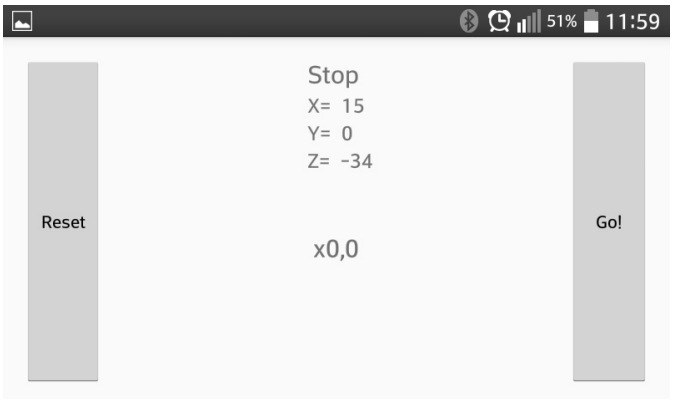


Rysunek 7. Ekran aplikacji sterującej pojazdem – nawiązywanie połączenia

Na **fotografiach 3…5** widoczne są szczegóły montażu i rozmieszczenie osprzętu. Wszystkie komponenty połączono zgodnie ze schematem montażowym zamieszczonym na **rysunku 6**.

Do sterowania pojazdem służy smartfon lub inne urządzenie z systemem Android i zainstalowaną aplikacją **Vehicle_v1**. Ekran aplikacji pokazano na **rysunku 7**. Na początek, na ekranie startowym należy wybrać urządzenie Bluetooth, z którym aplikacja ma się połączyć – chodzi o Serial Adapter, ponieważ tak jest rozpoznawany układ BTM222 zamontowany w module AVT1635. Po nawiązaniu połączenia z pojazdem zostanie wyświetlone ekran, jak na **rysunku 8**, a na płytce Arduino powinna zaświecić się dioda LED informująca o poprawnym połączeniu. W górnej linii okna aplikacji jest wyświetlony status pojazdu: „Stop” oznacza pojazd zatrzymany, „GO!!!” to pojazd w ruchu. Kolejne trzy linie „X=”, „Y=” oraz „Z=” zawierają dane odczytane z akcelerometru. Ostatnia linia przedstawia treść komendy wysyłanej do pojazdu – ma ona postać „\n *x wartość Y, wartość Z* \n”.

Przycisk „Reset” służy do wyzerowania wartości współrzędnych *x*, *y*, *z* w położeniu neutralnym – zerowanie należy wykonać przed każdą jazdą. Dopóki przycisk „Go!” Jest zwolniony, do pojazdu dociera komenda „x0,0” (pojazd zatrzymany). Dopiero, gdy przycisk „Go!” jest wciśnięty i trzymany,



Rysunek 8. Ekran aplikacji sterującej pojazdem – tryb zdalnego sterowania

Rysunek 9. Ekran aplikacji sterującej pojazdem – pojazd w ruchu

Listing 3. Obliczanie wartości współczynnika PWM

```
//-----------------------------------------------------
//Oblicza PWM
void vect2pwm (int vect1, int vect2) {
    int pwm_Rgo;
    int pwm_Rback;
    int pwm_Lgo;
    int pwm_Lback;
//najpierw wybieramy ruch do przodu lub do tyłu
    if (vect1 >= 0) {   //kierunek do tyłu
        if (vect1 > 0) {
            pwm_Rback = (vect1 * 2) + 57;  //pwm rozpoczyna się od współ. 57/255
            pwm_Lback = (vect1 * 2) + 57;   //nie ma mniejszych wartości
        } else {
            pwm_Rback = 0;
            pwm_Lback = 0;
        }
        pwm_Rgo = 0;
        pwm_Lgo = 0;
    } else {  //kierunek do tyłu
        pwm_Rgo = (vect1 * -2) + 57;  //pwm rozoczyna sie od współ. 57/255
        pwm_Lgo = (vect1 * -2) + 57;  //nie ma mniejszych wartości
        pwm_Rback = 0;
        pwm_Lback = 0;
    }
//teraz określamy kierunek lewo lub prawo
    if (vect2 >= 0) {
        vect2 *= 2;
        //jeśli ruch do przodu to
        if (pwm_Rgo > 0) pwm_Rgo -= vect2;  //spowalniamy prawą stronę
        //jeśli ruch do tyłu to
        if (pwm_Lback > 0) pwm_Lback -= vect2; //lub spowalniamy lewą stronę
    } else {
        vect2 *= -2;
        if (pwm_Lgo > 0) pwm_Lgo -= vect2;  //analogicznie
        if (pwm_Rback > 0) pwm_Rback -= vect2;
     }
    if (pwm_Rgo < 0) pwm_Rgo = 0;  //usuwa wartości ujemne
    if (pwm_Lgo < 0) pwm_Lgo = 0;
    if (pwm_Rback < 0) pwm_Rback = 0;
    if (pwm_Lback < 0) pwm_Lback = 0;
    analogWrite(motor_Rback, pwm_Rback);  //wpisuje wyniki do pwm
    analogWrite(motor_Lback, pwm_Lback);
    analogWrite(motor_Rgo, pwm_Rgo);
    analogWrite(motor_Lgo, pwm_Lgo);
    //Serial.println(pwm_Rgo);
    //Serial.println(pwm_Lgo);
    //Serial.println(pwm_Rback);
    //Serial.println(pwm_Lback);
}
```

parametry komendy przybierają wartości niezerowe, co pokazano na **rysunku 9**.

Program sterujący pojazdem napisano w środowisku Arduino. Program wykonuje dwa ważne zadania. Pierwsze, to odczytywanie danych z portu szeregowego UART, które są odbierane za pomocą modułu Bluetooth. Wśród nich wyszukiwane są ciągi w postaci „\n x Y , Z \n”. Symbol „\n” to znak nowej linii; „x” – umowny znacznik początku komendy; „Y”, „Z” – odczytane i przetworzone dane z akcelerometru w zakresie -99…99 oraz przecinek oddzielający parametry.

Procedurę odpowiedzialną za wyszukiwanie komend zamieszczono na **listingu 1**. Ponadto, program sprawdza czas pomiędzy komendami. Jeśli przez długi czas nie zostanie odebrana prawidłowa komenda, to pojazd automatycznie zatrzyma się. Odpowiada za to procedura z **listingu 2**. Kolejnym ważnym zadaniem programu jest obliczanie wartości współczynników PWM dla czterech kanałów na podstawie odebranych dwóch wartości *vectX* i *vectY*. Sposób wyznaczania współczynnika wypełnienia ilustruje **listing 3**.

Program aplikacji sterującej dla smartfona został napisany w środowisku Android Studio. Pełne źródła obu programów dostępne są w materiałach dodatkowych do projektu.

KS