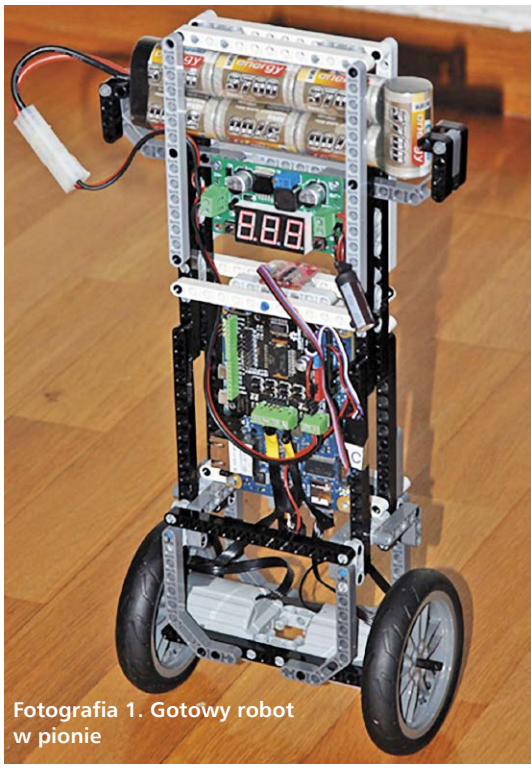
Samojezdny, dwukołowy robot z Intel Galileo

Od czasu pojawienia się na rynku mikroelektromechanicznych akcelerometrów, dużą popularnością wśród pokazowych aplikacji firm oraz projektów studenckich cieszą się maszyny, które są w stanie utrzymać w pionie, wbrew grawitacji, długie, pionowe konstrukcje. Ten bardzo widowiskowy pokaz możliwości inżynierskich wcale nie jest trudny do realizacji, a przy zastosowaniu gotowych komputerków jednopłytkowych, jego przygotowanie sprowadza się do połączenia ze sobą komponentów i dostosowania oprogramowania. Elementem wyróżniającym niniejszy projekt od pozostałych jest użycie w nim płytki Intel Galileo, która pozwala na uruchomienie na niej systemu operacyjnego Windows. Jest to także ciekawy przykład zastosowania algorytmu PID.



Fotografia 1. Gotowy robot w pionie

Omawiany projekt został wykorzystany z użyciem komponentów Lego, które ułatwiają montaż elementów. Ustawienie samojezdnego robota monitorowane jest za pomocą żyroskopu i akcelerometru – obu monitorujących ruch w trzech wymiarach. Całość pracuje pod kontrolą systemu operacyjnego Windows 8.1.

Dodatkowe informacje:
Autorem projektu jest Victor Cazacov, a opis projektu oraz pełny kod programów można znaleźć pod adresem: *https://github.com/cazacov/IntelGalileo/tree/master/BalancingRobot*

Komponenty

Sercem urządzenia jest płytka Intel Galileo. Model ten wyposażony jest w procesor Intel Quark X1000, taktowany zegarem 400 MHz i w 256 MB pamięci RAM. Galileo jest podłączone do płytki DFRduino L298P motor shield, będącej elementem napędzającym silniki kół robota. DFRduino jest płytką kompatybilną z Arduino i pozwala dostarczać do 2 A prądu do dwóch silników. Maksymalny pobór mocy płytki to 25 W, a jej wymiary to 55×55 mm. Jako napędy kółek autor zastosował dwa stałoprądowe silniczki LEGO 8882 Power function XL, zasilane napięciem 9 V. Użycie innych silniczków nie powinno stanowić problemu, a modele z LEGO są o tyle wygodne w użyciu, że dobrze pasują do reszty elementów konstrukcyjnych.

Zamiast samodzielnie lutować miniaturowe czujniki, dostępne w obudowach SMD i LGA oraz układy ich zasilania, autor skorzystał z gotowej płytki SparkFun IMU Fusion Board, na której zainstalowane są właśnie 3-osiowy akcelerometr ADXL345 firmy Analog Devices i 3-osiowy żyroskop IMU 3000 firmy Invensense. Sygnały z płytki SparkFun IMU Fusion Board są wyprowadzone na goldpiny, które łatwo podłączyć do innych elementów konstrukcji poprzez interfejs I2C.

Ponieważ robot jest mobilny, konieczne było wyposażenie go w źródło energii. Wybór padł na akumulator 8,4 V, złożony z 6 ogniw, które łącznie mają pojemność 3700 mAh. Źródło to ma na tyle dużą rezystancję wewnętrzną, że jego napięcie wyjściowe istotnie się zmienia, w zależności od poboru prądu. W związku z tym autor użył płytkę z obniżającą przetwornicą napięcia, uzyskując w ten sposób stabilne 5 V do zasilania całego zestawu. Przetwornica bazuje na układzie LM2596 firmy Texas Instruments.

Pozostałe elementy, potrzebne do wykonania projektu to przewody, wtyczki (m.in. do zasilania płytki Galileo) oraz klocki LEGO Technics.

Montaż i połączenia

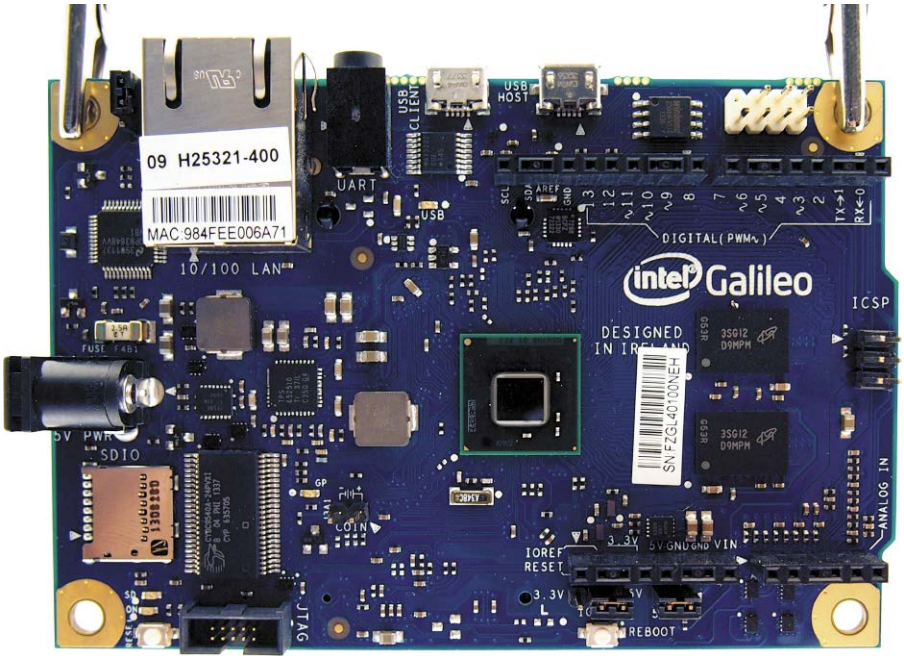
Realizując projekt, wykonawca staje przed wyborem lokalizacji poszczególnych elementów, z których wszystkie – ze względu na stosowanie modułów – zajmują pewną niemałą przestrzeń i trochę ważą. A ponieważ urządzenie w istotnej mierze opiera się na mechanice, czynniki te mają znaczenie. Najcięższym elementem jest akumulator i mogłoby się wydawać, że by utrzymać jak największą stabilność robota, powinien być on umieszczona jak najniżej – zaraz przy kołach, by obniżyć środek ciężkości. Jednakże w praktyce, jeśli robot z założenia ma być niestabilny i dopiero zastosowanie odpowiednich algorytmów ma powodować utrzymanie pionowej pozycji maszyny, okazuje się, że korzystniej jest umieścić „baterię" wyżej – o ile tylko użyte silniczki będą mieć odpowiednio duży moment obrotowy. Taka konstrukcja sprawia, że układ zachowuje się bardziej tak, jak odwrotne wahadło, które całkiem łatwo można kontrolować algorytmem PID. Wątpliwości może budzić też

DODATKOWE MATERIAŁY NA FTP:
ftp://ep.com.pl
USER: 87542, PASS: o8v5gec9

Potrzebne komponenty:
- Płytka Intel Galileo.
- 2×silniczki LEGO 8882 Power function XL.
- Płytka DFRduino L298P motor shield.
- Przetwornica obniżająca z układem LM2596.
- Płytka z akcelerometrem i żyroskopem SparkFun IMU Fusion board.
- Akumulator 8,4 V.
- Przewody i wtyczki.
- Klocki Lego Technics oraz śrubki do montażu.
- Opcjonalna karta sieciowa Wi-Fi lub modem Bluetooth na mini PCI Express.

kwestia lokalizacji sensorów. W wielu projektach tego typu są one montowane na dole robota, blisko kół. Jednakże i w tym przypadku okazuje się, że lepiej jest je umieścić wyżej, gdyż przy niższej pozycji, ruch w poziomie zbytnio zakłóca odczyty, powodując niemożność poprawnego reagowania na przechylenie robota. W praktyce umieszczenie czujników mniej więcej na wysokości 60% konstrukcji od kół, czyli w środku masy urządzenia, dało znacznie lepsze rezultaty.

Płytkę Galileo podłączono do DFRduino L298P Shield z użyciem pinów 4, 5 dla silniczka pierwszego oraz 6 i 7 dla drugiego silniczka. Piny 4 i 7 zostały wykorzystane do przesyłania informacji o kierunku obrotu, a piny 5 i 6 o prędkości (z użyciem sygnału PWM). Płytkę DFRduino podłączono do silniczków za pomocą konektorów śrubowych oraz do dodatkowego zasilania (terminal PWRIN). Płytkę z czujnikami podłączono do wyprowadzeń A4 i A5 Galileo, przez które (interfejsem I2C) przesyłane są odczyty z sensorów (piny SCL i SDA płytki

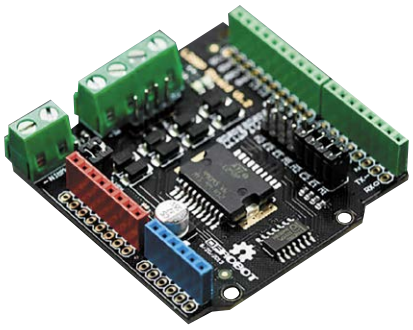


Fotografia 2. Płytka Intel Galileo

Listing 1. Przykładowy kod programu sterującego silnikami kół

```
int pin_motor_a_dir = 4;
int pin_motor_b_dir = 7;

int pin_motor_a_speed = 5;
int pin_motor_b_speed = 6;

void setup()
{
    pinMode(pin_motor_a_dir, OUTPUT);
    pinMode(pin_motor_b_dir, OUTPUT);
    pinMode(pin_motor_a_speed, OUTPUT);
    pinMode(pin_motor_b_speed, OUTPUT);
}

void loop()
{
    // Silnik 1
    digitalWrite(pin_motor_a_dir, HIGH);     // do przodu
    analogWrite(pin_motor_a_speed, 255);     // 100%

    // Silnik 2
    digitalWrite(pin_motor_b_dir, LOW);      // do tyłu
    analogWrite(pin_motor_b_speed, 128);     // 50%
}
```

Fotografia 3. Silniczek LEGO 8882 Power XL



Fotografia 4. Płytka DFRduino L298P motor shield

firmy SparkFun). W nowszej wersji oprogramowania Windows, zamiast wyprowadzeń A4 i A5 można użyć pinów SCL i SDA, dostępnych na Galileo, do podłączenia ich do analogicznie oznaczonych wyprowadzeń modułu IMU Fusion Board. Zasilanie modułu z czujnikami zostało pociągnięte bezpośrednio z Galileo.

Wyprowadzenie akumulatora podłączone zostało na wejście płytki stabilizatora, a ustabilizowane napięcie 5 V podłączono do gniazda zasilania Galileo oraz – jak wcześniej wspomniano – do terminalu PWRIN płytki DFRduino.

Polecenia i sygnały

Program dla robota został napisany w C#, ale wymagana jest znajomość tego języka tylko na podstawowym poziomie. Nie ma tu złożonych konstrukcji, które korzystałyby z dobrodziejstw C#, a jedynie proste, obiektowe komendy, takie jak w C++. Dopiero próba realizacji komunikacji z użytkownikiem wymaga użycia poleceń i bibliotek typowych dla C#.

Sterowanie napędem kół poprzez DFRduino jest dosyć proste – przykład został przedstawiony na **listingu 1**. Wystarczy określić piny, którymi przesyłane są informacje sterujące, ustawić je do trybu pracy jako wyjścia, a następnie w pętli przesyłać odpowiednie wartości. Sygnał wysoki na pinach sterujących kierunkiem oznacza jazdę do przodu, a niski – jazdę wstecz. Wypełnienie sygnału PWM równe 255 (100%) oznacza maksymalną szybkość obrotu, a 0 (0%) zatrzymanie koła. Wartości pośrednie odpowiadają liniowo szybkościom obrotów.

Odczyt z czujników jest nieco bardziej skomplikowany – został przedstawiony na **listingu 2**. Autor użył biblioteki *Wire.h* do komunikacji z użyciem interfejsu I²C. Ponadto przygotował funkcję *writeTo()*, która wywołuje wybrane urządzenie oraz pod wskazany adres przesyła dane i kończy transmisję. Posługuje się tą funkcją do wprowadzenia ustawień początkowych w funkcji *setup()*. Ustawia parametry pracy żyroskopu, przekazuje do żyroskopu informacje o akcelerometrze (żyroskop pośredniczy w komunikacji z akcelerometrem) oraz konfiguruje akcelerometr. Po wprowadzeniu wszystkich ustawień jest uruchamiana funkcja *loop()*, w której nawiązywane jest połączenie z żyroskopem i cyklicznie prowadzone są odczyty 12 bajtów danych, które umieszczane są w buforze.

Algorytm

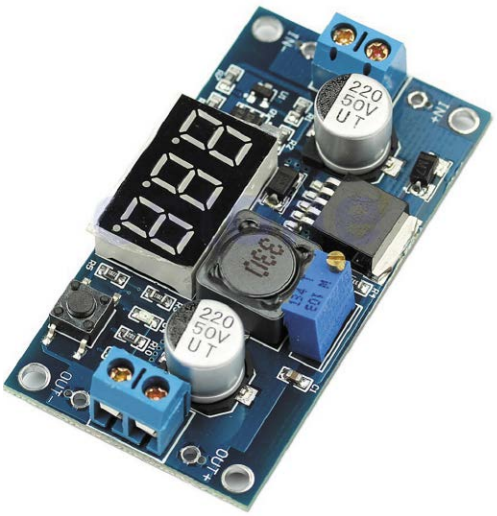
Opisany dotąd fragment projektu obejmuje wszystko, co konieczne do sterowania pracą dwóch silników i odczytywania informacji z sensorów, ale nie mówi nic na temat tego, jak

Listing 2. Przykład odczytu danych z czujników robota

```
#include <Wire.h>
#define GYRO 0x68
#define REG_GYRO_X 0x1D
#define ACCEL 0x53
#define ADXL345_POWER_CTL 0x2D

void setup()
{
    Wire.begin();
    // Próbkowanie 1kHz, Pasmo filtru 42Hz, Zakres 500 d/s
    writeTo(GYRO, 0x16, 0x0B);
    // adres rejestru danych akcelerometru
    writeTo(GYRO, 0x18, 0x32);
    // adres I2C akcelerometru dla żyroskopu
    writeTo(GYRO, 0x14, ACCEL);
    // żyroskop przekazuje polecenia do akcelerometru
    writeTo(GYRO, 0x3D, 0x08);
    // konfiguracja akcelerometru
    writeTo(ACCEL, ADXL345_POWER_CTL, 8);
    // powrót żyroskopu do normalnej pracy
    writeTo(GYRO, 0x3D, 0x28);
}

byte buffer[12];   // tablica z wczytywanymi danymi
void loop()
{
    Wire.beginTransmission(GYRO);
    Wire.write(REG_GYRO_X);
    Wire.endTransmission();

    Wire.requestFrom(GYRO, 12); // Wczytanie 12 bajtów danych
    i = 0;
    while (Wire.available())
    {
        buffer[i] = Wire.read();
        i++;
    }
}

void writeTo(int device, byte address, byte val) {
    Wire.beginTransmission(device); // inicjacja transmisji
    Wire.write(address);            // przesłanie adresu rejestru
    Wire.write(val);                // przesłanie wartości danej
    Wire.endTransmission();         // zakończenie transmisji
}
```

właściwie należy reagować na przechylenia i o ile obracać koła. Za te obliczenia odpowiada algorytm PID (proporcjonalno-całkująco-różniczkujący), znany wszystkim inżynierom, którzy zajmują się automatyką. Nie będziemy tu wyjaśniać jego zawiłości – były one już zgrubnie opisywane przy okazji innych projektów, np. w numerze 10/2007 Elektroniki Praktycznej. W skrócie mówiąc, algorytm PID pozwala sterowanemu systemowi na osiąganie zadanych wartości w oparciu o zmieniające się warunki. Algorytm tego typu monitoruje stan czujników, określa odchylenia wartości aktualnych od zadanych oraz szybkość ich narastania lub spadania, po czym ustala siłę oddziaływań, mających doprowadzić monitorowane wartości do pożądanego stanu.

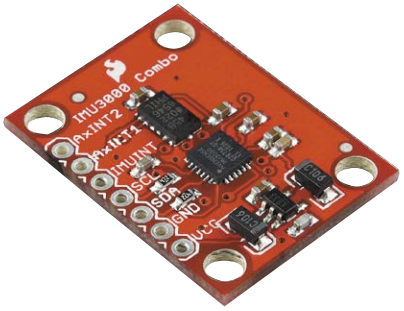
W naszym przypadku pożądanym stanem jest sytuacja, gdy robot stoi pionowo. Akcelerometry pozwalają określić, jak szybko kąt ustawienia robota się zmienia, a żyroskop sprawdza, na ile odległy jest od zadanego. Regulowana jest też prędkość ruchu robota, by ten mógł jeździć zgodnie z poleceniami.



Fotografia 5. Przetwornica obniżająca z układem LM2596 i wyświetlaczem diodowym

Implementacja algorytmu PID wymaga dobrania odpowiednich wartości współczynników do realizowanego zadania (po trzech na każdą monitorowaną wartość). Optymalny dobór wartości tych współczynników jest sztuką, której można się nauczyć z czasem, metodą prób i błędów oraz korzystając z powszechnie dostępnych poradników odnośnie strategii ich określania.

Warto jeszcze rzucić okiem na główny fragment kodu gotowego programu. Został on przedstawiony na **listingu 3**. Po załadowaniu bibliotek i zdefiniowaniu wszystkich potrzebnych obiektów, konfigurowane są poszczególne z nich, a w tym – podawane są parametry regulatorów kąta i prędkości. Pierwsza z wartości parametru (P – proporcjonalny) dla regulatora kąta jest ujemna, by od razu uwzględnić bezwładność podczas przyspieszenia robota w wybranym kierunku. Następnie, w głównej pętli, odbywa się kalibracja żyroskopu oraz



Fotografia 6. Płytka z sensorami Sparkfun IMU Fusion board

określenie długości pojedynczego cyklu pracy. Ta druga wartość jest potrzebna po to, by regulatory pracowały jak najbardziej w czasie rzeczywistym, tj. by kolejne zadawane przez algorytm wartości były podawane w równych odstępach czasu. Ponieważ Intel Galileo jest w stanie wykonywać program szybciej niż jest to potrzebne, autor wprowadził dodatkową pętlę opóźniającą, w której sprawdza czy nadszedł już czas kolejnej iteracji algorytmu; pętla ta jest pusta i przerywana dopiero, gdy ten czas nadszedł.

Za każdym razem, gdy konieczne jest wykonanie kolejnej iteracji algorytmu PID, odczytywane są wartości kąta z żyroskopu. Na ich podstawie określana jest chwilowa, docelowa wartość kąta oraz docelowa wartość prędkości kół. Ta ostatnia jest przekazywana do obiektu sterującego pracą silników i zapisywana na stosie na potrzeby kolejnej iteracji.

Uzupełnieniem kodu jest obsługa klawiatury, za której pomocą przekazywane są polecenia – czy to zmiany kierunku ruchu, czy zakończenia programu. Polecenie wczytujące komendy z klawiatury również znajduje się w głównej pętli programu.

Listing 3. Główny kod gotowego programu robota

```
#include „stdafx.h”
#include „arduino.h”
#include „Gyroscope.h”
#include „Motors.h”
#include „Regulator.h”
#include „KeyboardController.h”

#define TICK_PER_SECOND 75
#define TICK_LENGTH_MICROSECONDS 1000000L / TICK_PER_SECOND

int _tmain(int argc, _TCHAR* argv[])
{
    return RunArduinoSketch();
}

Gyroscope gyroscope;
Motors motors;
Regulator angleRegulator(TICK_PER_SECOND);
Regulator speedRegulator(TICK_PER_SECOND / 2);
KeyboardController keyboard;
Integrator speedIntegrator(TICK_PER_SECOND);

float targetSpeed = 0;
float targetAngle = 0;

void setup()
{
    motors.init();
    gyroscope.init();
    keyboard.init();
    angleRegulator.init(-0.02, 0, 0);
    speedRegulator.init(35, 0.2, 0);
    wprintf(L”Press Esc to exit\n”);
}

void loop()
{
    wprintf(L”Calibration in 3 seconds\n”);
    delay(3000);
    wprintf(L”Calibrating...”);
    gyroscope.calibrate();
    wprintf(L”Cycle tyme in microseconds: %ld \n”, TICK_LENGTH_MICROSECONDS);
    // główny cykl pracy
    delay(100);
    unsigned long lastIteration = micros();
    unsigned long nextIteration = lastIteration + TICK_LENGTH_MICROSECONDS;
    int n = 0;
    do {
        float actualAngle, deltaAngle;
        while (micros() < nextIteration)
        {
            // pusta pętla
        }
        unsigned long now = micros();
        gyroscope.getAngleFiltered(actualAngle, deltaAngle, now);
        targetAngle = angleRegulator.getResult(speedIntegrator.getSum(), 0,
now - lastIteration);
        float motorSpeed = 0;
        motorSpeed = speedRegulator.getResult(actualAngle - targetAngle,
deltaAngle, now - lastIteration);

        n++;
        if (n % 20 == 0) // Komunikat co 20 cykl
        {
            wprintf(L”%d\t%f\t%f\t%f\t%f\n”, n, targetAngle, actualAngle,
deltaAngle, motorSpeed);
        }
        motors.setSpeedBoth(motorSpeed);
        speedIntegrator.pushValue(motorSpeed);
        lastIteration = now;
    nextIteration = now + TICK_LENGTH_MICROSECONDS;
        keyboard.processEvents();
        if (keyboard.shouldExit())
        {
            break;
        }
    } while (true);
    motors.stopAll();
    wprintf(L”Exiting...\n”);
    exit(0);
}
```

Podsumowanie

Zademonstrowany projekt nie jest szczególnie innowacyjny, ale pokazuje, że tego typu aplikację można wykonać nie tylko na popularnym Raspberry Pi czy Arduino, ale także na platformach intelowskich, a co najciekawsze – z użyciem systemu operacyjnego Windows. Tu niestety pojawia się problem, którego autorowi projektu nie udało się rozwiązać pod Windowsem. Robot w opisanej wersji jest sterowany przez podłączoną do niego klawiaturę przewodową, co niewątpliwie silnie ogranicza jego ruchy. Oczywiście system Windows można dosyć łatwo skonfigurować do pracy z klawiaturą bezprzewodową, ale znacznie wygodniejszym rozwiązaniem byłoby sterowanie robota przez sieć. Autor podjął takie próby, korzystając z połączenia przewodowego, co wcale nie rozwiązywało problemu „smyczy”, choć praktyka pokazała, że zastosowanie niestandardowego, płaskiego przewodu ethernetowego ułatwia ruchy robota. W takiej konfiguracji polecenia z klawiatury są przesyłane przez sieć do konsoli systemu i z niej odczytywane.

Najbardziej problematyczne okazało się używanie bezprzewodowej karty sieciowej Wi-Fi pod systemem Windows 8.1. To duża szkoda, gdyż płytka Galileo ma na sobie złącze mini PCI Express, które pozwoliłoby zainstalować niedrogi, szybki moduł Wi-Fi czy Bluetooth. Niestety autorowi nie udało się skonfigurować takiej karty pod Windows, mimo że pod linuksem (dystrybucja Yocto Linux) funkcjonowała bez problemu. Być może w Windows 10, który jest obecnie promowany przez Microsoft do wszelkich aplikacji Internetu Przedmiotów nie sprawiałby problemu.

Marcin Karbowniczek, EP

Fotografia 7. Intel Galileo z kartą Wi-Fi