**Rafał Nazarko**

Koło Naukowe Elektroniki i Technologii Informacyjnych

**Sterowanie zestawem LEGO Mindstorms za pomocą komputera Raspberry Pi 4 przy użyciu języka Python**

**Streszczenie**

Artykuł opisuje zasadę działania komunikacji mikrokomputera Raspberry Pi 4 z jednostką sterującą zestawu LEGO Mindstorms NXT lub EV3 przy użyciu bardzo popularnego języka programowania Python.

**Słowa kluczowe:** układ elektroniczny, programowanie, zdalne sterowanie.

1. Wprowadzenie

Głównym celem przyświecającym stworzeniu oprogramowania była potrzeba zwiększenia elastyczności sterowania w robotach tworzonych przez członków koła naukowego. Podstawowe narzędzie do oprogramowania, dostarczone przez firmę LEGO, ma określony zestaw poleceń, który często okazuje się niewystarczający przy bardziej wymagających projektach.

Użycie zewnętrznego kontrolera pozwala zwiększyć moc obliczeniową, którą dysponuje tworzona maszyna – a co za tym idzie również prędkość działania, wykorzystać złożone algorytmy lub rozszerzyć komunikację o dodatkowe porty USB, połączenie internetowe lub urządzenia wejścia/wyjścia.

1. Komunikacja mikrokomputera Raspberry Pi 4 z jednostką sterującą LEGO Mindstorms

Jednostka sterująca LEGO Mindstorms zapewnia trzy rodzaje komunikacji: Bluetooth, Wi-Fi i USB. Wszystkie z nich pozwalają tworzyć aplikacje działające na dowolnym komputerze i komunikować się z jednostką sterującą LEGO Mindstorms. Aby zapewnić maksymalną prędkość transmisji danych, powinno się stosować połączenie za pomocą kabla USB.

Jednostka sterująca LEGO Mindstorms, podobnie jak inne współczesne komputery grupuje 8 bitów w jeden bajt i adresuje swoją pamięć po jednym bajcie. Sąsiadujące ze sobą bajty graficznie oddzielane są dwukropkami „ : ” lub pionowymi kreskami „ | ”. Z racji tego, że zapis binarny jest bardzo długi i zazwyczaj nieczytelny, przyjęło się stosować notację szesnastkową. Zapis w tej postaci jest zwarty a konwersja do i z systemu binarnego prosta, ponieważ jedna cyfra szesnastkowa oznacza pół bajtu.

Aby to dobrze zrozumieć, poniżej został przedstawiony przykład komendy, która pinguje jednostkę sterującą (zmusza ją do odpowiedzi):

|  |
| --- |
| 0x|06:00|2A:00|00|00:00|01| |

Listing 1. Konstrukcja komendy ping

Jak widać komenda składa się ośmiu bajtów i znaku określającego liczbę szesnastkową *0x*. Dwa pierwsze bajty są częścią protokołów komunikacyjnych, którymi mogą być – jak wspomniano wcześniej – Bluetooth, USB lub Wi-Fi. Kolejne dwa bajty są licznikiem komunikatów, które pozwoli dopasować polecenie bezpośrednie oraz odpowiedź na nie. Piąty bajt określa czy nadawca tego polecenia oczekuje na odpowiedź czy też nie (|00| oznacza oczekiwanie na odpowiedź zaś |80| jej nie potrzebuje). Bajty szósty i siódmy stanowią nagłówek komendy, który jest kombinacją dwóch liczb, które określają rozmiary polecenia bezpośredniego. Od bajtu ósmego rozpoczynają się operacje, czyli dokładne informacje o działaniu, jakich oczekuje od jednostki sterującej mikrokomputer.

1. Pisanie i wysyłanie bezpośrednich poleceń

Aby otworzyć komunikację z jednostką sterującą z poziomu kodu Python na mikrokomputerze Raspberry Pi 4, należy zaimportować bibliotekę *ev3\_dc*, która dostarcza pakiet narzędzi ułatwiających rozwiązanie podstawowych zagadnień. Konfigurację należy rozpocząć od zdefiniowania obiektu hosta, którym jest jednostka sterująca. W tym celu można użyć poniższego fragmentu kodu:

|  |
| --- |
| #!/usr/bin/env python3  import ev3\_dc as ev3  my\_ev3 = ev3.EV3(  protocol=ev3.USB,  host='00:16:53:5F:19:24'  )  my\_ev3.verbosity = 1  my\_ev3.sync\_mode = ev3.SYNC |

Listing 2. Konfiguracja obiektu hosta

Argumenty podane pod nazwą *protocol* oraz *host* definiują odpowiednio, który z trzech protokołów (wspomnianych wcześniej) będzie używany do transmisji danych oraz adres Mac jednostki sterującej (te informację można znaleźć w ustawieniach). Ustawienie *my\_ev3.verbosity* na wartość 1 pozwala na podgląd komunikacji pomiędzy mikrokomputerem a jednostką sterującą. Gdy powyższa operacja zostanie wykonana, można przejść bezpośrednio do wysyłania komend. Konstrukcja pojedynczej operacji powinna mieć postać jak poniżej:

|  |
| --- |
| cmd = b ''. join ((  ev3.opSound, # Moduł  ev3.TONE, # Czynność  ev3.LCX (440), # Częstotliwość  ev3.LCX (1000), # Czas trwania  ))  my\_ev3.send\_direct\_cmd(cmd) |

Listing 3. Konstrukcja komendy odtwarzającej dźwięk

W powyższym przykładzie cała komenda jest przypisana do zmiennej *cmd*. Zostaje ona wywołana przy pomocy funkcji *send\_direct\_cmd(cmd)* znajdującej się w instancji obiektu jednostki sterującej. Pierwszą linią zawierającą się w funkcji *join* kreującą daną komendę, powinna być informacja o module, do którego użytkownik chce się zwrócić. W tym przypadku jest to moduł dźwiękowy dlatego użyto wartości *ev3.opSound.* Kolejna linia definiuje w jaki sposób ma być wykorzystany wybrany moduł. Użyte w powyższym przykładzie *ev3.TONE* zainicjuje granie dźwięku o częstotliwości 440Hz przez 1000ms. Odpowiadają za to linie trzecia oraz czwarta. Moduł dźwiękowy w jednostkach sterujących LEGO Mindstorms nie jest rozbudowany posiada bowiem jeszcze tylko jeden przypadek użycia, w którym można odegrać muzykę zapisaną na karcie SD. Istnieją jednak takie moduły, które oferują szeroki wachlarz możliwości użycia.

Przed rozpoczęciem korzystania z przygotowanej biblioteki, warto poznać metody statyczne jakie zostały w niej wykorzystane. W głównej mierze konwertują one liczby lub znaki zrozumiałe dla ludzi na język czytelny dla jednostki sterującej LEGO Mindsotrms. Składają się na nie:

* *LCX(int)* – konwertuje liczbę całkowitą na polecenie bezpośrednie z bajtem identyfikacyjnym. W zależności od wartości przekazanej w argumencie tej funkcji będzie ciągiem bajtów o długości jednego (LC0), dwóch (LC1), trzech (LC2) lub pięciu bajtów (LC4).
* *LCS(string)* – dodaje do podanego w argumencie funkcji łańcucha znaków bajt identyfikacyjny oraz terminator początkowy oraz końcowy ciągu bajtów.
* *LVX(int) –* konwertuje adres pamięci lokalnej na format zgodny z poleceniem bezpośrednim z bajtem identyfikacyjnym. W zależności od wartości przekazanej w argumencie tej funkcji będzie ciągiem bajtów o długości jednego (LV0), dwóch (LV1), trzech (LV2) lub pięciu bajtów (LV4).
* *GVX(int) –* konwertuje adres pamięci globalnej na format zgodny z poleceniem bezpośrednim z bajtem identyfikacyjnym. W zależności od wartości przekazanej w argumencie tej funkcji będzie ciągiem bajtów o długości jednego (GV0), dwóch (GV1), trzech (GV2) lub pięciu bajtów (GV4).
  1. Zarządzanie czujnikami

Podstawowym źródłem informacji dla każdej autonomicznej maszyny jest odczyt z jej czujników. Nie jest inaczej w przypadku tych tworzonych przy użyciu zestawu LEGO Mindstorms. Choć w tym przypadku jest to dużo prostsze niż w przypadku wysoko zaawansowanych technicznie rozwiązań, trzeba zachować pewne standardy obsługi tychże czujników. Z tej potrzeby powstał szablon, do który odpowiada za spójność przesyłanych komend. Jej szkielet można zobaczyć na poniższym listingu:

|  |
| --- |
| def ConstructCommand(self, daisyChainLayer, port, sensorType, sensorMode, values = 1, values1 = 0):  return b''.join((  ev3.opInput\_Device, # Moduł  ev3.READY\_SI, # Czynność  ev3.LCX(daisyChainLayer), # Warstwa Daisy Chain  ev3.LCX(port), # Numer portu  ev3.LCX(sensorType), # Typ czujnika  ev3.LCX(sensorMode), # Tryb czujnika  ev3.LCX(values), # Liczba zwracanych wartości  ev3.GVX(values1), # Dodatkowe wartości  )) |

Listing 4. Konstrukcja szablonu komend

Używanie tego szablonu jest banalnie proste. Wystarczy podać odpowiednie wartości jako argumenty tej funkcji a komenda zostanie zwrócona jako ciąg bajtów, który można bezpośrednio wysłać do jednostki sterującej LEGO Mindstorms. Poszczególne parametry i ich znaczenie zostały opisane poniżej:

* Moduł – parametr nie podlegający zmianie, ustawiony na *ev3.opInput\_Device*, który odpowiada za komunikację z urządzeniami peryferyjnymi takie jak czujniki lub serwomotory.
* Czynność – statyczna wartość, która odpowiada za typ zwracanych danych. Może być ustawiona tak jak w szablonie na *READY\_SI*, która odczytuje wartości z czujnika jako liczbę zmiennoprzecinkową lub *READY\_PCT*, wtedy zwracane wartości są w postaci procentowej (0-100).
* Warstwa Daisy Chain (ang. Daisy Chain layer) – połączenie Daisy Chain oznacza szereg spiętych ze sobą szeregowo jednostek sterujących LEGO Mindstorms, w sposób który umożliwia im swobodną komunikację. Układ działa w architekturze Master-Slave. Każda jednostka jest sterowana za pomocą jej poprzednika. Pozycja w tym łańcuchu oznacza warstwę Daisy Chain. Można połączyć do czterech urządzeń w ramach jednego łańcucha. Aby rozkazać jakąś czynność konkretnej jednostce, wystarczy w kreatorze komend podać numer warstwy Daisy Chain jako argument *daisyChainLayer*.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Jednostka |  | Jednostka |  | Jednostka |  | Jednostka |
| Przychodząca | sterująca |  | sterująca |  | sterująca |  | sterująca |
| komenda |  |  |  |  |  |  |  |
|  | Daisy Chain |  | Daisy Chain |  | Daisy Chain |  | Daisy Chain |
|  | Warstwa 1 |  | Warstwa 2 |  | Warstwa 3 |  | Warstwa 4 |

Rysunek 1. Połączenie Daisy Chain

Źródło: opracowanie własne

* Numer portu – jednostka sterująca posiada osiem fizycznych portów – cztery przeznaczone są dla silników a 4 dla czujników. Porty dla silników oznaczone są czterema pierwszymi literami alfabetu łacińskiego A, B, C oraz D zaś dla każdego portu dla czujników przyporządkowana jest liczba z zakresu 1-4. Aby odczytać wartość z wybranego czujnika należy podać numer portu, do którego jest podpięty, jako argument *port*.
* Typ czujnika – czujniki w zestawach LEGO Mindstorms mają kilka różnych typów. Mogą być to czujniki dźwięku, natężenia światła, koloru, ultradźwięków, temperatury, dotyku, fal podczerwonych oraz żyroskop. Każdemu z powyższych typów, odpowiada wartość numeryczna, którą można znaleźć w *Tabeli 1*. Aby określić typ czujnika należy odczytaną z tabeli wartość wprowadzić jako argument *sensorType* funkcji *ConstructCommand*.
* Tryb czujnika – pojedynczy czujnik zazwyczaj oferuje kilka funkcjonalności. Przykładowo, czujnik koloru może pracować w trybie pomiaru światła odbitego, natężenia światła, koloru RGB, światła zielonego, światła czerwonego lub światła niebieskiego. Każdemu z powyższych typów, odpowiada wartość numeryczna, którą można znaleźć w *Tabeli 1*. Aby określić tryb należy odczytaną z tabeli wartość wprowadzić jako argument *sensorMode* funkcji *ConstructCommand*.
* Liczba zwracanych wartości – aby określić jak wiele pomiarów powinna przeprowadzić wybrana jednostka sterująca na danym czujniku, należy podać wartość numeryczną jako argument *values* w kreatorze komend. Domyślnie ta liczba jest ustawiona na 1.
* Dodatkowe wartości – niektóre czujniki mogą oczekiwać podania dodatkowych danych. Argument stanowi jedynie otwartą furtkę dla deweloperów, nie jest on jednak wymagany i domyślnie jest ustawiony na 0.

Tabela 1. Typy oraz tryby sensorów wraz z opisami funkcji.

|  |  |  |
| --- | --- | --- |
| **Typ** | **Tryb** | **Opis** |
| *1* | *0* | *NXT-Touch* |
| *1* | *NXT-Bump* |
| *2* | *0* | *NXT-Light-Reflected* |
| *1* | *NXT-Light-Ambient* |
| *3* | *0* | *NXT-Sound-DB* |
| *1* | *NXT-Sound-DBA* |
| *4* | *0* | *NXT-Color-Reflected* |
| *1* | *NXT-Color-Ambient* |
| *2* | *NXT-Color-Color* |
| *3* | *NXT-Color-Green* |
| *4* | *NXT-Color-Blue* |
| *5* | *NXT-Color-Raw* |
| *5* | *0* | *NXT-Ultrasonic-Cm* |
| *1* | *NXT-Ultrasonic-Inch* |
| *6* | *0* | *NXT-Temperature-C* |
| *1* | *NXT-Temperature-F* |
| *7* | *0* | *EV3-Large-Motor-Degree* |
| *1* | *EV3-Large-Motor-Rotation* |
| *2* | *EV3-Large-Motor-Power* |
| *8* | *0* | *EV3-Medium-Motor-Degree* |
| *1* | *EV3-Medium-Motor-Rotation* |
| *2* | *EV3-Medium-Motor-Power* |
| *9* |  | *Free* |
| *…* | *…* | *…* |
| *13* |  | *Free* |
| *14* | *0* | *Output for 3th party devices* |
| *15* |  | *Free* |
| *16* | *0* | *EV3-Touch* |
| *1* | *EV3-Bump* |
| *17* |  | *Free* |
| *…* | *…* | *…* |
| *20* |  | *Free* |
| *21* | *0* | *Test purpose* |
| *22* |  | *Free* |
| *…* | *…* | *…* |
| *27* |  | *Free* |
| *28* | *0* | *3th party input , 1 mode, Scale 0 - 4095* |
| *1* | *3th party input , 2 mode, Scale 0 - 5000* |
| *2* | *3th party input , 3 mode, Scale 0 - 10000* |
| *3* | *3th party input , 4 mode, Scale 0 - 20000* |
| *29* | *0* | *EV3-Color-Reflected* |
| *1* | *EV3-Color-Ambient* |
| *2* | *EV3-Color-Color* |
| *3* | *EV3-Color-Reflected-Raw* |
| *4* | *EV3-Color-RGB-Raw* |
| *5* | *EV3-Color-Calibration* |
| *30* | *0* | *EV3-Ultrasonic-Cm* |
| *1* | *EV3-Ultrasonic-Inch* |
| *2* | *EV3-Ultrasonic-Listen* |
| *3* | *EV3-Ultrasonic-SI-Cm* |
| *4* | *EV3-Ultrasonic-SI-Inch* |
| *5* | *EV3-Ultrasonic-DC-Cm* |
| *6* | *EV3-Ultrasonic-DC-Inch* |
| *31* |  | *Free* |
| *32* | *0* | *EV3-Gyro-Angle* |
| *1* | *EV3-Gyro-Rate* |
| *2* | *EV3-Gyro-Fast* |
| *3* | *EV3-Gyro-Rate & Angle* |
| *4* | *EV3-Gyro-Calibration* |
| *33* | *0* | *EV3-IR-Proximity* |
| *1* | *EV3-IR-Seeker* |
| *2* | *EV3-IR-Remote* |
| *3* | *EV3-IR-Remote-Advanced* |
| *4* | *Not utilized* |
| *5* | *EV3-IR-Calibration* |
| *34* |  | *Free* |
| *…* | *…* | *…* |
| *98* |  | *Free* |
| *99* | *0* | *Energy-Meter-Voltage-In* |
| *1* | *Energy-Meter-Amps-In* |
| *2* | *Energy-Meter-Voltage-Out* |
| *3* | *Energy-Meter-Amps-Out* |
| *4* | *Energy-Meter-Joule* |
| *5* | *Energy-Meter-Watts-In* |
| *6* | *Energy-Meter-Watts-Out* |
| *7* | *Energy-Meter-All* |
| *100* | *0* | *IIC-Byte* |
| *1* | *IIC-WORD* |
| *101* | *0* | *NXT-Test* |
| *102* |  | *Free* |
| *…* | *…* | *…* |
| *120* |  | *Free* |

Źródło: LEGO MINDSTORMS EV3 Firmware Developer Kit

* 1. Korzystanie z biblioteki

Znając mechanizmy odpowiadające za wykonywanie poleceń w jednostce sterującej, można płynnie przejść do korzystania ze stworzonej biblioteki.

Pierwszym krokiem, który należy postawić jest połączenie jednostki sterującej z urządzeniem wysyłającym komendy. Jak wspomniano w drugim rozdziale tego artykułu, można tą operację przeprowadzić na trzy sposoby. Na potrzeby tego artykułu użyto kabla USB, łącząc jednostkę sterującą LEGO Mindstorms z mikrokomputerem Raspberry Pi 4. W przypadku połączenie typu Daisy Chain, wystarczy podać adres MAC pierwszej jednostki sterującej, czyli tej bezpośrednio połączonej w urządzeniem wysyłającym komendy.

Kolejnym krokiem jest utworzenie pliku w języku Python, który będzie plikiem odpowiadającym za rozpoczęcie komunikacji. Wstępną postać tego pliku pokazuje kod na *Listingu 2*. W tym momencie powinno się przetestować połączenie, aby wykluczyć przyczyny ewentualnie powstałych błędów w trakcie rozwijania programu.

Ostatni krok to utworzenie instancji czujników, z które użytkownik chce obsługiwać podczas działania swojego programu. Wystarczy zaimportować bibliotekę *sensors*, odszukać w niej podklasę wybranego czujnika odpowiadającej typowi jednostki sterującej (*NXT* lub *EV3*), a następnie utworzyć jego instancję obiektu. Dla przykładu, poniższy skrypt tworzy obiekt klasy czujnika ultradźwiękowego, po czym wypisuje na konsolę zmierzoną przez niego odległość podaną w centymetrach:

|  |
| --- |
| ultrasonicSensor1 = sensors.EV3Sensors.UltrasonicSensor(my\_ev3, 4, 0)  print(ultrasonicSensor1.GetDistanceInCm()) |

Listing 5. Kod tworzący instancję obiektu czujnika ultradźwiękowego oraz pomiar odległości

3. Podsumowanie

Aktualnie



Rysunek 3. Kod QR zawierający adres do repozytorium projektu

Źródło: opracowanie własne

Literatura

1. Lis M., *C#. Praktyczny kurs. Wydanie III*, Wydawnictwo Helion, 2016

Źródła internetowe

1. Stefan Jahn, Modified Nodal Analysis, http://qucs.sourceforge.net/tech/node14.html