Uniwersytet Zielonogórski
Instytut Sterowania i Systemów Informatycznych

Laboratorium Sterowania Robotów

Sterowanie silnikami robota Lego NXT

Uwagi wstępne

1. Wszystkie przykłady i zadania wykonujemy w środowisku MATLAB z użyciem skrzynki narzędziowej `RWTHMindstormsNXT`.
2. Komunikację z robotem rozpoczynamy od podłączenia poprzez kabel USB. Następnie wykorzystujemy połączenie poprzez `Bluetooth`, zgodnie z zaleceniami prowadzącego.
3. Poniższy opis został przygotowany na podstawie informacji umieszczonych na stronie: `http://www.mindstorms.rwth-aachen.de/`.

Wstęp teoretyczny

1. Przed tworzeniem nowej aplikacji dla robota LEGO NXT musimy upewnić się, że wszystkie zmienne globalne, wskaźniki do struktur i ustawienia są wyzerowane, a okna z wykresami zamknięte. W tym celu wystarczy wykonać poniższy zestaw instrukcji

   ```
   COM_CloseNXT('all')   % zamykamy wszystkie połączenia z NXT
   close all             % zamykamy otwarte okna
   clear all             % usuwamy wszystkie zmienne
   ```

   na początku każdej tworzonej aplikacji. Uwaga! Należy pamiętać aby polecenie `close all` następowało po poleceniu `COM_CloseNXT('all')`.

2. Aby otworzyć połączenie z robotem, możemy użyć jednego z poniższych poleceń
   - `COM_OpenNXT` - wykonanie tego polecenia jest podstawowym sposobem otwarcia połączenia z robotem; wynikiem jest uchwyt(tj. wskaźnik) do struktury z parametrami połączenia.
   - `COM_OpenNXTEx` - to rozszerzona wersja powyższej funkcji, umożliwiająca między innymi określenie z którym robotem chcemy się połączyć. Funkcja ta ma głównie zastosowania w przypadku budowy robota z wykorzystaniem wielu kostek NXT.
3. Połaczenie poprzez kabel USB jest najłatwiejszym i najszybszym sposobem połączenia z naszym robotem.

   ```
   hNXT = COM_OpenNXT(); % szukamy połaczenia przez USB
   NXT_PlayTone(440, 500, hNXT);
   ```

   Wykonując powyższe polecenia inicjujemy połączenie z robotem i odtwarzamy dźwięk aby sprawdzić połączenie. Zmienna `hNXT` jest uchwytem do struktury opisującej urządzenie z którym mamy połączenie. Dlatego zmienna `hNXT` jest argumentem funkcji `NXT_PlayTone()`, w przeciwnym wypadku nie wiadomo którego `NXT` należy użyć.

   W przypadku korzystania z jednej i tej samej kostki NXT przez naszą aplikację, możemy uniknąć ciągłego używania tego samego uchwytu jako argumentu każdej funkcji poprzez ustawienie domyślnej wartości. W tym celu należy wykonać poniższy program

   ```
   COM_CloseNXT all
   hNXT = COM_OpenNXT();       %  szukamy połączenia przez USB
   COM_SetDefaultNXT(hNXT);    % ustawiamy domyślną wartość uchwytu

   % teraz nie musimy juz używać uchwytu hNXT
   NXT_PlayTone(440, 500);
   pause(0.5);
   NXT_PlayTone(440, 500);

   % ale nie zapominajmy o odpowiednim zakończeniu naszej aplikacji
   COM_CloseNXT(hNXT);
   ```

4. Sterowanie silnikiem wymaga stworzenia odpowiedniego obiektu klasy `NXTMotor`. Obiekt ten umożliwia wysyłanie poleceń do jednego lub dwóch silników i otrzymywać od nich dane (odczyt położenia z enkoderów). Aby stworzyć obiekt klasy `NXTMotor` wystarczy użyć następującego polecenia

```
mA = NXTMotor('A')
```

Obiekt `mA` posiada odpowiednie własności i metody do sterowania sinikiem podłączonym do portu `A`. Pierwszym poleceniem sterującym będzie uruchomienie silnika (i sprawdzenie czy działa)

```
mA.Power = 50;  %  ustawiamy połowę mocy = 50%
mA.SendToNXT(); % uruchamiamy silnik
```

Silnika `A` działa... Aby go zatrzymać, wywołujemy funkcję

```
x
mA.Stop('off');
```

Powyższe polecenie po prostu odłącza zasilanie od silnika powodując jego zatrzymanie. Jeśli jednak potrzebujemy natychmiastowego zatrzymania, lepiej jest wywołać następujące polecenie.

```
mA.Stop('brake');
```

W tym przypadku tzw. aktywny hamulec będzie użyty, powodując iż silnik zatrzyma się natychmiast po otrzymaniu tego polecenia. Zatrzymanie jest natychmiastowe ale również wymaga to relatywnie dużo mocy z baterii. Dlatego po zatrzymaniu najlepiej jest wyłączyć silnik, czyli wywołujemy funkcję

```
mA.Stop('off');
```

5. **Precyzyjne sterowanie silnikiem**.
W większości przypadków, sterowanie silnikiem poprzez ustawianie tylko wartości mocy sinika nie jest wystarczające. Możemy poruszać silnikiem w następujący sposób:

```
mA = NXTMotor('A', 'Power', 50);
mA.SendToNXT();
pause(3);                  % czekaj  3 sekundy
mA.Stop('brake');
```

Korzystając z powyższego zestawu instrukcji, sinik `A` będzie obracał się przez 3 sekundy. Należy jednak pamiętać, że różne obciążenie silnika (różna masa silnika) lub opóźnienia w komunikacji mogą powodować iż działanie sinika przez 3 sekundy daje w rezultacie różne odległości pokonane przez naszego robota. Dlatego musimy poruszać silnikiem zadaną liczbę obrotów. W tym celu użyjemy właściwości `TachoLimit`.

```
mA.TachoLimit = 360;  % chcemy obrotu o 360 stopni
mA.SendToNXT();
```

Teraz silnik wykonuje dokładnie jeden obrót (360 stopni z dokładnością ± 1 stopień). Bardzo ważnym jest aby zrozumieć, że po przesłaniu polecenia do silnika, rozpoczyna on jego wykonanie a program natychmiast przechodzi do kolejnej instrukcji. Przykład:

```
mB = NXTMotor('B', 'Power', 50, 'TachoLimit', 1000);
mB.SendToNXT();
NXT_PlayTone(440, 500);
```

Wykonując powyższe instrukcje, nasz robot zacznie wykonywać ruch silnikiem `B` i natychmiast wyda dźwięk (czyli przed zakończeniem ruchu silnika)! Jeśli chcemy wydać dźwięk w momencie kiedy silnik osiągnie zadaną pozycję (czyli po wykonaniu ruchu), musimy czekać na silnik aż skończy wykonywać zadane polecenie:

```
mB.SendToNXT();
mB.WaitFor();                    % wstrzymujemy  MATLAB'a  na chwile
                                                % czyli czekamy aż silnik skończy
NXT_PlayTone(440, 500);  % teraz silnik jest zatrzymany
```

Należy teraz pamiętać: Aby silnik wykonał polecenie `SendToNXT()`, musi być w stanie bezczynności lub spoczynku, czyli wykonał już ruch zadany poprzednim poleceniem. Możemy to osiągnąć na dwa sposoby. W pierwszym przypadku oczekujemy na zakończenie ostatniego polecenia silnika wykonujemy poprzez wywołanie funkcji `WaitFor()` przed wykonaniem kolejnego polecenia `SendToNXT()`. Oczywiście, jeśli silnik aktualnie nie wykonuje żadnego polecenia (czyli jest w stanie bezczynności) to wywołanie funkcji `WaitFor()` nie spowoduje żadnego oczekiwania tylko natychmiastowe przejście to kolejnego polecenia. Drugą możliwością jest zatrzymanie silnika po wykonaniu metody `Stop()`. Jeśli nie będziemy postępować według powyższych zasad to nasz robot zignoruje drugie polecenie wysłane do silnika. Przykład:

```
% chcemy wykonac dwa obroty po 360 stopni
mB = NXTMotor('B', 'Power', 50, 'TachoLimit', 360);
mB.Stop('off');  % Zatrzymujemy dla pewnosci silnik
                          % - musi być w odpowiednim stanie czyli nie poruszać się
mB.SendToNXT();  % Silnik jest teraz zajęty - wykonuje obrót
mB.SendToNXT();  % TO NIE DZIAŁA !!! SILNIK JEST ZAJETY
% ten przykład jest zły.
% musimy użyć funkcji WaitFor() pomiędzy poleceniami  SendToNXT()
```

Aby umożliwić ruch sinika bez ograniczeń musimy ustawić właściwość `TachoLimit` na wartość 0.

6. **Różne tryby hamowania**
Ustawiając wartość własności `TachoLimit` określamy kąt obrotu silnika, a znak własności `Power` określa kierunek ruchu. Jednak wciąż mamy kilka możliwości sterowania hamowaniem silników. Właściwość odpowiedzilna za sterowanie hamowaniem to `ActionAtTachoLimit`. Możliwe wartości tej właściwości to:

   (a) '`Coast`' - powoduje że silnik są po prostu wyłączane kiedy wartość `TachoLimit` jest osiągnięta. Powoduje to jednak powolne hamowania aż do zatrzymania siników. Jeśli wartość `TachoLimit` nie będzie osiągnięta silniki spowodują przesunięcie robota zbyt daleko.

   (b) '`Brake`' - jest to wartość domyślna powodująca łagodne hamowanie aż do osiągnięcia wartości `TachoLimit`. Dzięki temu osiągamy wysoką dokładność ruchu robota. po zatrzymaniu siniki są wyłączane.

   (c) '`Holdbrake`' jest podobny '`Brake`', jednak w tym przypadku silniki nie są wyłączane ale mamy większy pobór prądu z baterii.

7. **Regulacja prędkości i rozpoczęcie ruchu silników**
Istnieją jeszcze dwie własności które wymagają omówienia

   (a) `SpeedRegulation` - ta własność ma domyślnie wartość `true`. Oznacza to, że prędkość silnika jest stała i niezależna od obciążenia (masy robota), czyli sterowanie mocą silnika jest tak naprawdę **niezależne** od nas (zmienia się dynamicznie w zależności od obciążenia). Jeśli jednak obciążenie się nie zmienia to możemy bez problemów używać tej opcji.

   (b) `SmoothStart` - ta własność pozwala na łagodne przyspieszanie silników podczas startu (unikamy poślizgu kół podczas ruszania). Ta opcja działa jednak tylko kiedy `TachoLimit`> 0 jest ustawione (ale nie działa kiedy `ActionAtTachoLimit = 'Coast'`).

8. **Sterowanie dwoma silnikami**
Wszystkie dotychczasowe przykłady odnosiły się do sterowania jednym silnikiem. Możemy jednak stworzyć dwa obiekty klasy `NXTMotor` do sterowania dwom silnikami (zakładamy że koła robota są przymocowane do silników podłączonych odpowiednio do portów `B` i `C`):

```
mB = NXTMotor('B', 'Power', 50);
mC = NXTMotor('C', 'Power', 50);
mB.SendToNXT();
mC.SendToNXT();
```

W powyższym przypadku napotykamy dwa problemy. Polecenia do silników są wysyłane jeden po drugim. W rezultacie silnik `C` zostanie uruchomiony później niż silnik `B`. Oznacza to, że robot wykona skręt przez rozpoczęciem ruchu. Po drugie silniki mogą uruchomić się z odrobinę różnymi prędkościami, czyli robot nie będzie się poruszał po linii prostej. Rozwiązaniem jest tutaj użycie jednego obiektu klasy dla dwóch silników. Oto przykład

```
mBC = NXTMotor('BC', 'Power', 50);
mBC.SendToNXT();
```

Łatwe, prawda? Możemy również użyć polecenia `WaitFor()` lub poleceń hamowania tak jak to było do rej pory. Dostęp do właściwości jest również taki sam jak poprzednio. Oto przykład:

```
myMotors             = NXTMotor();
myMotors.Port        = [MOTOR_B; MOTOR_C];  % to samo jak  myMotors.Port = 'BC';
myMotors.Power       = 50;
myMotors.TachoLimit  = 1000;
myMotors.SmoothStart = true;
myMotors.SpeedRegulation = false;  % nie zapominajmy o tym!
% czyli podczas tworzenia obiektu dla dwóch silników, regulacja prędkości
% będzie wyłączona automatycznie, czyli nie uzywamy:
% m = NXTMotor('BC');  % bo <-- m.SpeedRegulation = false   jest już ustawione

myMotors.Stop('off');  % upewnijmy się że silniki są wyłączone
                                        %przed wysłaniem polecenia wykonania ruchu.
myMotors.SendToNXT();
```

Jest jeszcze coś o czym powinniśmy pamiętać. Popatrzmy na następujący przykład:

```
mBC = NXTMotor('BC', 'Power', 50, 'TachoLimit', 1000);
mBC.ActionAtTachoLimit = 'Coast';
mBC.SendToNXT();
mBC.WaitFor();
pause(3);
```

Podczas pauzy (przez 3 sekundy), możemy usłyszeć dziwny szum pochodzący z siników. Wydaje się również że są one zasilane lub wibrują. To jest wynikiem synchronizacji, która wciąż jest aktywna. Jeśli więc wykonujemy ruch jednym silnikiem to wtedy drugi poruszany jest automatycznie. Dzięki temu jesteśmy pewni że silniki są na tej samej pozycji. Możemy wyłaczyć synchronizację korzystając z metody `Stop()`:

```
mBC.Stop('off');
```

9. **Przykłady sterowania robotem** Posiadając już podstawową wiedzę o sterowaniu sinikami, możemy przystąpić do sterowania ruchem naszego robota. Oto przykład:

```
leftWheel   = MOTOR_B;  % lewe koło napędzane silnikiem B
rightWheel  = MOTOR_C;  % prawe koło napędzane silnikiem C
bothWheels  = [leftWheel; rightWheel];
drivingPower = 60;
turningPower = 40;
drivingDist  = 1000; % w stopniach
turningDist  = 220;  % w stopniach

% tworzymy odpowiedni obiekt
mForward = NXTMotor(bothWheels, 'Power', drivingPower, 'TachoLimit', drivingDist);
mReverse = mForward; % kopia obiektu
mReverse.Power = -mForward.Power; % i zamieniamy kierunek jazdy !
```

Czyli możemy poruszać naszym robotem w przód i tył. Wyślemy więc odpowiednie polecenia i poczekamy aż zostaną wykonane:

```
% jedziemy do przodu
mForward.SendToNXT();
mForward.WaitFor();

% i wracamy do początku
mReverse.SendToNXT();
mReverse.WaitFor();

% zrobione - wydajemy dźwięk
NXT_PlayTone(440, 500);
```

Aby skręcić (np. o 90 stopni), wykonujemy 2 kroki: Po pierwsze jedno koło obraca się w jednym kierunku (czyli robot skręca o 45 stopni czyli połowę obrotu.), a później wykonujemy drugi krok czyli obrót drugim kołem w przeciwnym kierunku. W ten sposób skręt o 90 stopni będzie wykonany.

```
% do skręcania, potrzebujemy odpowiednią liczbę obiektów :
mTurnLeft1 = NXTMotor(leftWheel, 'Power', -turningPower, 'TachoLimit', turningDist);
mTurnLeft1.SpeedRegulation = false; %  tego nie potrzebujemy do skręcania

% drugi etap skrętu, Używamy  powyższych ustawień i je modyfikujemy :
mTurnLeft2 = mTurnLeft1;                 % kopia obiektu
mTurnLeft2.Port     = rightWheel;        % ale dla drugiego koła
mTurnLeft2.Power    = -mTurnLeft1.Power; % zamiana kierunku

% teraz obiekt do skrętu w prawo
mTurnRight1 = mTurnLeft1;               % pierwsza kopia...
mTurnRight2 = mTurnLeft2;
mTurnRight1.Power = -mTurnRight1.Power;
mTurnRight2.Power = -mTurnRight2.Power;
```

Teraz możemy użyć stworzone powyżej obiekty

```
% skręt w lewo
mTurnLeft1.SendToNXT();
mTurnLeft1.WaitFor();
mTurnLeft2.SendToNXT();
mTurnLeft2.WaitFor();

% czekamy chwilę
pause(1);

% skręcamy w prawo wracając do pozycji wyjściowej
mTurnRight1.SendToNXT();
mTurnRight1.WaitFor();
mTurnRight2.SendToNXT();
mTurnRight2.WaitFor();
```

10. **Odczytywanie pozycji silników** poza wysyłaniem poleceń do silników, możemy również odczytywać z nich informacje. Najważniejsza informacja jaką można odczytać to aktualna pozycja w stopniach. Aby dowiedzieć się więcej na ten temat, należy sprawdzić pomoc do funkcji `ReadFromNXT()` i `ResetPosition()`.

    Wywołanie funkcji `ReadFromNXT()` zwraca strukturę zawierającą wiele użytecznych pól z informacjami. Najważniejsze z nich to pole `Position`, gdzie jest przechowywana informacja o aktualnej pozycji silnika. Zmiana wartości pola `Position` odnosi się również do kierunku obrotu: Jeśli aktualnie zadana moc silnika jest dodatnia (pole `Power`) to wartość `Position` jest zwiększana. W przeciwnym wypadku wartość ta jest zmniejszana (i może nawet zawierać wartości ujemne).Oto przykład:

    ```
    % Oczywiście potrzebujemy odpowiedniego obiektu silnika
    % aby nim sterować...
    m = NXTMotor('A', Power, '50');
    % zerujemy licznik:
    m.ResetPosition();

    m.SendToNXT();
    pause(1);
    m.Stop('off');
    % zatrzymujemy silnik, który może jeszcze się obracać - metoda Stop('off').

    % odczytajmy informację z silnika
    data = m.ReadFromNXT();
    % i wyświetlmy w środowisku Matlab'a:
    disp(sprintf('Motor A is currently at position %d', data.Position));

    % poczekajmy jeszcze chwilkę aż silnik sie zatrzyma
    pause(2);
    % i wyświetlmy pozycję jeszcze raz:
    data = m.ReadFromNXT();
    disp(sprintf('Motor A is finally at position %d', data.Position));
    ```

Należy tutaj podkreślić iż inne własności obiektu silnika z wyłączeniem `Port` nie mają związu do not matter to use the method the methods `ReadFromNXT()` i `ResetPosition`. Również struktura danych zwracana przez `ReadFromNXT()` odnosi się tylko do aktualnego stanu silnika, i nie ma związku z innymi właściwościami silnika używanymi do odczytu danych (np. `TachoLimit` i `Power`).

Zadania do wykonania

1. Zbudować robota o napędzie różnicowym według wskazówek prowadzącego.
2. Przetestować wszystkie programy umieszczone w powyższym wprowadzeniu teoretycznym.
3. Napisać program umożliwiający wyznaczenie zależności pomiędzy zadaną mocą silnika (właściwość `.Power`) a jego prędkością obrotową. Przyjąć czas pomiaru >= 20 sekund. Przed każdym pomiarem należy zapisać (odczytać) aktualny poziom napięcia baterii. W tym celu należy użyć polecenia `voltage = NXT_GetBatteryLevel(handle)`, gdzie `voltage` to odczytany poziom napięcia baterii a `handle` to uchwyt do struktury z parametrami połączenia.
4. Zależność pomiędzy mocą a prędkością obrotową silnika należy wyznaczyć z użyciem funkcji `polyfit`, np.:
`p=polyfit(x,y,r); % x, y - serie danych, r - zadany stopien wielomianu`.
5. Zaimplementować proste funkcje umożliwiające konwersję prędkości obrotowej ([obr/min]) na prędkość kątową [rad/sec] i vice versa.
6. Napisać program umożliwiający wyznaczenie zależności pomiędzy zadaną mocą (właściwość `.Power`) a jego prędkością liniową robota. Wykorzystać wyznaczone już zależności pomiędzy mocą a prędkością obrotową silnika. Przyjąć czas pomiaru >= 20 sekund a średnicę koła napędowego = 56 mm. Przed każdym pomiarem należy zapisać (odczytać) aktualny poziom napięcia baterii. Wyznaczyć zależność z użyciem funkcji `polyfit`.
7. Napisać program do poruszania robotem 20 cm w przód i w tył. Wykonać oddzielnie sterownie niezależne dwoma silnikami osobno, a następnie to samo przy użyciu synchronicznego sterowania silnikami. Zmierzyć fizycznie pokonaną odległość przez robota. Wskazać możliwe źródła błędów.
8. Zaimplementować funkcję do skrętu w dowolną stronę i o dowolną liczbę stopni. Użyć zaimplementowaną funkcję do programu umożliwiającego poruszaniem robotem wzdłuż następujących trajektorii
   - trójkąt równoboczny, gdzie bok ma długość 30 cm;
   - kwadrat, gdzie bok ma długość 30 cm.

Sprawozdanie

Sprawozdanie z przeprowadzonego laboratorium powinno zawierać:

- Kody źródłowe wszystkich utworzonych podczas laboratorium skryptów(Nie zamieszczać kodów programów przykładowych !) wraz z komentarzem.
- Wyniki pomiarów i odpowiednie wykresy zależności.
- Uwagi i wnioski