

Sprawozdanie z działania generatora trajektorii o trapezoidalnym profilu prędkościowym

Karolina Borkowska

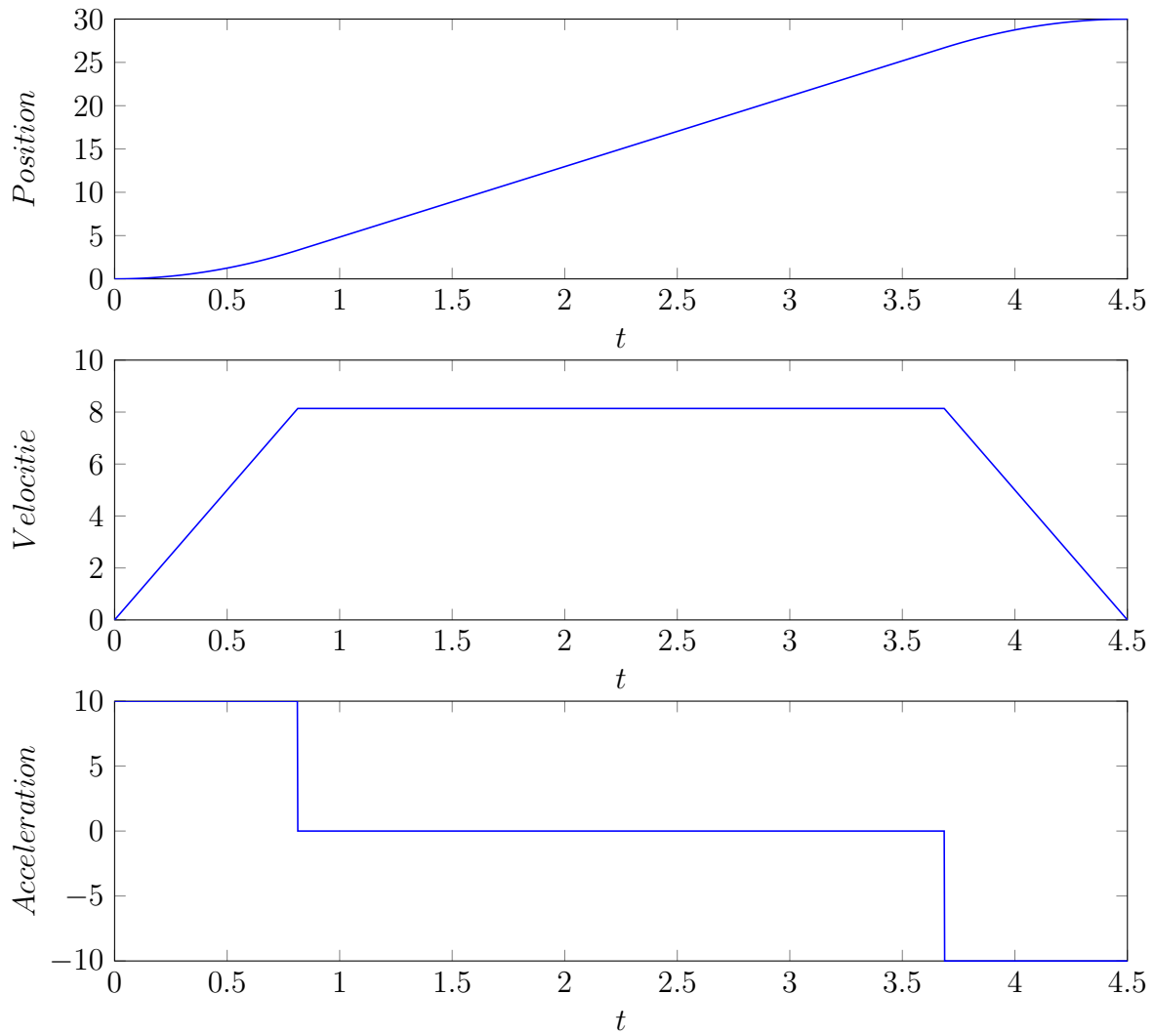
12 czerwca 2018

1. Wstęp

Zaimplementowano generator trajektorii o profilu trapezoidalnym na podstawie równań podanych w książce *Trajectory Planning for Automatic Machines and Robots* napisanej przez Luigi Biagiotti’ego i Claudio Melchiorri’ego. Działa on w dwóch podstawowych trybach: czasowym i prędkościowym, w zależności od tego, co zadaje użytkownik oraz posiada dodatkowy podtryb badawczy ograniczający dziedzinę pracy generatora. Dwa nowe komponenty zostały połączone z istniejącą strukturą systemu IRPOS, zarówno poprzez pliki konfiguracyjne jak i dostęp przez api systemu. System Do realizacji tego projektu naniesiono zmiany w trzech katalogach systemu: `irp6_robot`, `irp6_ui`, `orocos_controllers`.

2. Trajektorie o trapezoidalnym profilu prędkościowym

Do realizacji zadania identyfikacji właściwości kinematycznych robota IRp6 postanowiono zaimplementować generator trajektorii trapezoidalnej. Pozwala on na w prosty sposób zapewnić w określonych fazach ruchu stałą prędkość, czy przyspieszenie. **Rysunek 1** przedstawia przykładową trajektorię trapezoidalną, wygenerowaną za pomocą wzorów wykorzystywanych później do implementacji nowych funkcjonalności w systemie IRPOS. Współczynniki na podstawie których obliczana jest w danym momencie pozycja i/lub jej pochodne, wyprowadzone zostały na podstawie wzorów podanych w książce *Trajectory Planning for Automatic Machines and Robots* napisanej przez Luigi Biagiotti’ego i Claudio Melchiorri’ego. Dodatkowo za pomocą programu Matlab napisano symulacje generatora oraz komponentu przekształcającego nastawy stawów na położenia silników. Dzięki nim można było dopracować warunki przyjęcia żądania ruchu jako poprawnego, tak by jak zmaksymalizować możliwości systemu. Ponad to skrypty Matlab’a przydatne były w ramach testowania generatora, dzięki nim można było najpierw sprawdzić jak w pełni powinna wyglądać wygenerowana trajektoria przy zadanych wartościach.



Rysunek 1: Wykresy przedstawiające przykładową trajektorię trapezoidalną

Na **rysunku 1** fazy można jasno rozróżnić patrząc na wykresy prędkości i przyspieszenia, które mają mocno wyodrębnione momenty zmiany przyspieszenia. Kolejno są one opisane współczynnikami:

1. **Faza przyspieszania** (acceleration phase):

$$ap_1 = q_i - v_i t_i + \frac{(v_{max} - v_i)t_i^2}{(2T_a)} \quad (1)$$

$$ap_2 = v_i - \frac{(v_{max} - v_i)t_i}{T_a} \quad (2)$$

$$ap_3 = \frac{v_{max} - v_i}{(2T_a)} \quad (3)$$

2. **Faza stałej prędkości** (constant velocity phase):

$$cvp_1 = q_i + \frac{(v_i - v_{max})t_i}{2} - v_{max}t_i \quad (4)$$

$$cvp_2 = v_{max} \quad (5)$$

$$cvp_3 = 0 \quad (6)$$

3. **Faza zwalniania** (deceleration phase):

$$dp_1 = q_f - v_f(t_i + T) - \frac{(v_{max} - v_f)(t_i + T)^2}{2T_d} \quad (7)$$

$$dp_2 = v_f + \frac{(v_{max} - v_f)(t_i + T)}{T_d} \quad (8)$$

$$dp_3 = \frac{(v_f - v_{max})}{2T_d}. \quad (9)$$

Gdzie q_i -położenie początkowe, q_f -położenie końcowe, v_i -prędkość początkowa, v_f -prędkość końcowa, v_{max} -prędkość maksymalna (ew. osiągnięta w fazie zerowego przyspieszenia), t_i -chwila początkowa, T -czas trwania trajektorii, T_a -czas trwania fazy przyspieszania, T_d -czas trwania fazy zwalniania.

Ogólnie generator ma zawsze określone momenty, prędkości i położenia graniczne. Czasy trwania faz są obliczane dla każdej trajektorii w następujący sposób:

$$T_a = \frac{v_{max} - v_i}{a_{max}} \quad (10)$$

$$T_d = \frac{v_{max} - v_f}{a_{max}}. \quad (11)$$

Opcjonalnie do generatora podany może zostać czas trwania trajektorii i przyspieszenie maksymalne lub prędkość maksymalna i przyspieszenie maksymalne. W pierwszym przypadku prędkość fazy zerowego przyspieszenia obliczana jest za pomocą:

$$v_{max} = \frac{1}{2}(v_i + v_f + a_{max}T - \sqrt{a_{max}^2T^2 - 4a_{max}h + 2a_{max}(v_i + v_f)T - (v_i - v_f)^2}) \quad (12)$$

Gdzie h to dystans do przebycia.

Druga opcja wiąże się z potrzebą obliczenia czasu trwania trajektorii:

$$T = \frac{h}{v_{max}} + \frac{v_{max}}{2a_{max}}(1 - \frac{v_i}{v_{max}})^2 + \frac{v_{max}}{2a_{max}}(1 - \frac{v_f}{v_{max}}) \quad (13)$$

Gdzie h to dystans do przebycia.

W tym przypadku istnieje możliwość, że nie można osiągnąć prędkości maksymalnej, stanie się tak gdy:

$$ha_{max} \leq \sqrt{v_{lim}^2 - \frac{v_i^2 + v_f^2}{2}}. \quad (14)$$

Jeśli pomimo tego wszystkie warunki poprawności zadania są spełnione, trajektoria składa się tylko z fazy przyspieszania i zwalniania. Prędkość do jakiej generator będzie dążyć zmienia się na:

$$v_{lim} = \sqrt{ha_{max} + \frac{v_i^2 + v_f^2}{2}}. \quad (15)$$

Należy wtedy ponownie obliczyć zmienne czasowe:

$$T_a = \frac{v_{lim} - v_i}{a_{max}} \quad (16)$$

$$T_d = \frac{v_{lim} - v_f}{a_{max}} \quad (17)$$

$$T = T_a + T_d. \quad (18)$$

Zmiany jakie następują przy ćofaniu” ($q_i > q_f$) to przemnożenie przyspieszenia, prędkości oraz odległości. Następuje to po obliczeniu prędkości w przypadku zadanego czasu trwania ruchu lub od razu w drugim przypadku.

3. Krótki opis nowych modułów

W celu ułatwienia zapoznania się ze zmianami wprowadzonymi do IRPOS’a, najpierw pokrótce przedstawione zostaną najważniejsze nowe elementy, przed opisem ich budowy i zmianami wprowadzonymi do ich akomodacji.

Do systemu wprowadzono cztery nowe komponenty oraz dwa pakiety do obsługi komunikacji, kolejno:

- `Irp6pmTrapezoidTrajectoryGeneratorJoint`, generator trajektorii trapezoidalnej pracujący w przestrzeni stawów;
- `Irp6pmTrapezoidTrajectoryActionJoint`, komponent akcji, stanowiący interfejs pomiędzy generatorem, a IRPOS-api, rozumianym jako transformację wiadomości do odpowiednich formatów; wstępnie sprawdza poprawność zadanych wartości oraz ostateczny wynik ruchu;
- `Irp6pmTrapezoidTrajectoryGeneratorMotor`, generator trajektorii trapezoidalnej pracujący w przestrzeni silników, jako że stawy robota są sprzężone efekt działania tych dwóch generatorów nie jest równoważny, nawet jeśli weźmie się pod uwagę różnicę w rzędach wielkości zadanych;
- `Irp6pmTrapezoidTrajectoryActionMotor`, komponent akcji pracujący z generatorem przestrzeni silników;
- `trapezoidal_trajectory_msgs`, zbiór wiadomości stworzonych do pracy z IRPOS’em. Składa się on ze zgłoszenia typu “akcja ” do komunikacji z IRPOS-api oraz “typowych ” wiadomości ROS’owych, które przenoszą informację między komponentami, ewentualnie stanowią część wiadomości akcji;
- `rtt_trapezoidal_trajectory_msgs`, pakiet umożliwiający prace `trapezoidal_trajectory_msgs` w czasie rzeczywistym.

Komponenty akcji i generatorów działają na podstawie tego samego kodu źródłowego, niezależnie od przestrzeni ruchu.

4. Uruchamianie systemu w trybie nohardware

Poniżej przedstawiony jest uproszczony schemat uruchomienia systemu ze zwróceniem szczególnej uwagi na miejsca, w których naniesione zostały zmiany. Opisane zostaną najważniejsze zmiany zamieszczone w katalogu `irp6_ui`. Pominięte mogą zostać pliki pośrednie, konsolidujące kolejne skrypty lub tylko przekierowujące.

Uruchomienie systemu z perspektywy użytkownika sprowadza się do wywołania w terminalu trzech komend:

- `roslaunch irp6_bringup run_common_simclock.sh`, czyli skryptu który ustawia jaki zegar ma być używany, agregatory diagnostyczne i publikatory stanu robotów.
- `roslaunch irp6_bringup irp6-*--nohardware.launch`, gdzie “*” zastąpiona zostaje przez określenie jaką częścią sprzętu użytkownik chce się zajmować. Skrypt ten uruchamia pliki odpowiedzialne za przygotowanie oprogramowania do pracy z robotami, przede wszystkim węzeł ROS’owy, pod który podczepiony jest OROSCOS’owy deployer zajmujący się uruchomieniem i organizacją komponentów.
- `roslaunch rviz rviz`, opcjonalne oprogramowanie do wizualizacji robotów.

Nowo dodane komponenty uruchamiane są w kroku drugim. W jego ramach w pierwszym z pliku `irp6-p-inside.launch` podawane są wartości do serwera parametrów. Dopisano w nim atrybuty dla wszystkich nowych modułów. Generatory trajektorii trapezoidalnej, w przeciwieństwie do swoich odpowiedników spline’owych, otrzymują informację o maksymalnych wychyleniach stawów lub silników oraz przykładowe wartości maksymalne prędkości i przyspieszeń. Na obecną chwilę są to bezpieczne wartości zapewniające możliwość testowania ruchu, w trakcie którego każdy z silników ma pokonać trasę w tym samym czasie. Po przeprowadzeniu identyfikacji właściwości fizycznych systemu wartości te zostaną zastąpione właściwymi.

Kolejnym etapem jest analiza skryptów OROSCOS’owych. Plik `common-imports.ops` rozszerzono o nazwy katalogów, w których umieszczono żądania ich zaimportowania. Do oryginalnego pliku dodano komendy dla katalogów kodów źródłowych generatora, komponentu akcji oraz wiadomości.

Ostatnią fazą uruchamiania jest analiza OROSCOS’owego skryptu `irp6-p-inside.ops`. W nim komponenty są ładowane, nadaje im się parametry z serwera parametrów, oraz wywoływana jest funkcja konfiguracyjna. Ponadto, komponenty akcji mają określane priorytet i okres wywoływania. Dla nowych komponentów nadano te same wartości, które były ustawione w systemie dla ich spline’owych odpowiedników (komponenty action mają okres 0.01 sek, priorytet 2). W przypadku generatorów nie następuje bezpośrednie ustawienie częstotliwości, ani priorytetu. Są one łączone z komponentem `Irp6pScheme`, który jest instancją narzędzia `conman::Scheme`. Nakłada ono dodatkowe ograniczenia usprawniające pracę komponentów oraz zmniejsza prawdopodobieństwo wystąpienia błędów. Plik `irp6-p-inside.ops` zawiera też informacje o połączeniach portowych pomiędzy komponentami. Generatory łączy się z komponentami akcji oraz modułami odbierającymi zadane położenie i publikującymi obecne ustawienia robota. Przesyłanie wiadomości typu “akcja” pomiędzy komponentami akcji, a IRPOS-api zapewnia połączenie np. `Irp6pTrapezoidTrajectoryActionJoint` poprzez serwis `actionlib`. Modułowi jest nadawana unikatowa nazwa w tym kontekście, znana programiście api tak by mógł on za jej pomocą skonfigurować ze swojej strony klienta. Na koniec komponenty akcji są

stratowane razem ze starymi elementami systemu, a generatory pozostają nieaktywne do momentu ich uruchomienia przez api. W ten sposób tylko jeden komponent nadaje regulatorom wartości zadane.

5. Korzystanie z IRPOS-api

Do IRPOS-api wprowadzono osiem nowych funkcji, które jawnie może wywoływać użytkownik:

- `move_to_motor_position_trapezoid_velocity`,
- `move_along_motor_trajectory_trapezoid_velocity`,
- `move_to_motor_position_trapezoid_duration`,
- `move_along_motor_trajectory_trapezoid_duration`,
- `move_to_joint_position_trapezoid_velocity`,
- `move_along_joint_trajectory_trapezoid_velocity`,
- `move_to_joint_position_trapezoid_duration`,
- `move_along_joint_trajectory_trapezoid_duration`,

które przyjmują kolejno:

- punkty do osiągnięcia,
- prędkości i przyspieszenia maksymalne (tylko funkcje `velocity`),
- wartość flagi `save_data`,
- wartość flagi `research_mode`,

oraz funkcję pomocniczą do translacji kodów rezultatu ruchu:

- `trapezoid_error_code_to_string`.

Dla ułatwienia pracy z kodem wydzielono kolejną klasę `IRPOS_T`, która dziedziczy po klasie `IRPOS`, czyli oryginalnym IRPOS-api. Oprócz nowych metod do obsługi ruchu o profilu trapezoidalnym `IRPOS_T` rekonfiguruje menedżera `commanSwitch` oraz dodaje nowe klienty do komunikacji z komponentami akcji.

5.1 Różnice pomiędzy metodami `motor` i `joint`

Metody `motor` i `joint` pozwalają na wykonanie zadań odpowiednio w przestrzeni silników lub stawów. Z perspektywy implementacji różnią się one między sobą tylko wyborami inego klienta, przez co innych komponentów obliczających zadane, i wysłaniem wartości tolerancji pozycji właściwych danej przestrzeni pracy. Jako że, pomimo możliwości osiągnięcia przez staw konkretnego położenia, może on konfliktować z ustawieniami silników, przed testowaniem funkcji `joint` w pierwszej kolejności sprawdzano czy pożądana konfiguracja robota nie powoduje naruszenia granic położenia silników. Dokonano tego za pomocą skryptów programu Matlab, które symulowały zachowanie generatorów i ułożenie silników. Warto też wspomnieć, że na poziomie komponentów OROCOS'owych nie ma różnic między kodem użytym do pracy w tych przestrzeniach.

5.2 Różnice pomiędzy metodami `move_to_*_position` i `move_along_*_trajectory`

IRPOS-api pozwala na zadanie zarówno pojedynczego punktu lub kilku kolejnych do osiągnięcia. Dla użytkownika najważniejszą różnicą jest sposób przekazywania celu. Funkcje `move_to_*_position` przyjmują jako pierwszy argument jednowymiarową tablicę, pojedynczy cel, a `move_along_*_trajectory` wektor wiadomości typu `JointTrajectoryPoint`, czyli cały ich zbiór.

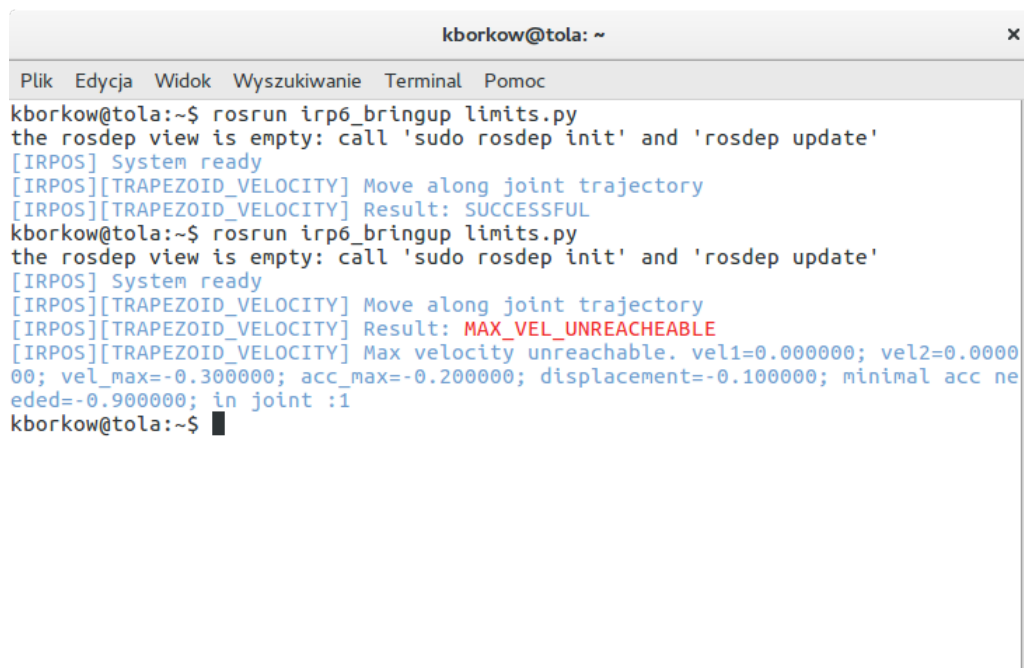
Sprawdzanie poprawności zadanych jest normalnie podzielone na dwie części, obsługiwaną w komponencie akcji i tą obliczaną przez generator. Będzie to szerzej opisane w sekcji TODO. Tu należy jedynie wspomnieć, że przekroczenie limitów pozycji zawsze zostanie wykryte przed jakimkolwiek ruchem, natomiast wystąpienie innych błędów sprawdzane jest przy obliczaniu profili prędkościowych. Efektywnie oznacza to, że przy użyciu funkcji typu `move_along_*_trajectory` może wystąpić sytuacja w której robot poruszy się do pierwszego z punktów i dopiero wtedy wykryte zostanie naruszenie warunków generatora.

5.3 Metody `velocity` i `duration`

Wybór pomiędzy funkcjami `velocity`, a `duration` polega na określeniu celu ruchu. Metody `velocity` opierają obliczanie profili prędkościowych na podstawie zadanych: położenia oraz maksymalnych prędkości i przyspieszenia. Można za ich pomocą identyfikować właściwości kinematyczne robota lub zachować większą kontrolę nad właściwościami ruchu. Nie pozwalają one jednak na zachowanie jednolitego czasu trwania ruchu na wszystkich silnikach. Wygodniejsze są metody `duration`. Zadaje się im tylko punkty do osiągnięcia, a generator sam obliczy jaki czas zajmie mu ruch na podstawie predefiniowanych parametrów. Docelowo będą to wartości zbadane podczas testów na robocie, zapewniając optymalny czas pracy. IRPOS-T, przy wysyłaniu zadanych do komponentów akcji, ustawia flagę `duration_mode`. Ustawiona na wartość "false" indykuje tryb prędkościowy. Dokładny sposób oraz konsekwencje działania tych dwóch trybów są opisane w sekcjach TODO oraz TODO2 dla funkcji `velocity` i `duration` kolejno.

5.4 Flagi

Każda z metod przyjmuje dwie flagi `save_data` i `research_mode`. Pierwsza z nich pozwala na zapisanie danych ruchu w pliku `IRPOS_results` w katalogu domowym. Należy jednak pamiętać, że ze względu na wymagania systemów czasu rzeczywistego, generator alokuje pamięć na wyniki w trakcie konfiguracji komponentu, a nie dynamicznie. Przez co może nastąpić niepełny zapis danych przy długim czasie wykonywania rozkazu. Działanie `research_mode` jest zależna od tego czy profil prędkościowy jest obliczany w trybie `duration`, czy `velocity`. W obu przypadkach blokuje on możliwość nadania prędkości początkowej lub końcowej. Jest to wprowadzone by ułatwić pracę przy niepewnych wartościach zadanych. Ponad to komponent akcji nie sprawdza czy trasa nie wykroczyła za bardzo poza granice tolerancji dla . Dodatkowo w trybie `velocity` generator ma kolejny warunek: osiągnięcie maksymalnej prędkości. Oznacza to, że jeśli użytkownik chce pokonać pewną odległość, niewystarczającą do rozpędzenia silnika przy zadanym przyspieszeniu, generator odmówi wykonania zadania. Jest możliwe przemieszczenie silnika przy niższej, ale miało by się z celem badania czy jest ona możliwa do osiągnięcia. Pierwszy tryb nie ma zaimplementowanego tego warunku, gdyż przemieszczenie wszystkich silników w tym samym czasie ogranicza do stopnia silnie utrudniającego wykorzystanie funkcji. Poza tym tryb ten nie jest stworzony do zbadania właściwości kinematycznych robota.



```
kborkow@tola: ~  
Plik  Edycja  Widok  Wyszukiwanie  Terminal  Pomoc  
kborkow@tola:~$ roslaunch irp6_bringup limits.py  
the rosdep view is empty: call 'sudo rosdep init' and 'rosdep update'  
[IRPOS] System ready  
[IRPOS][TRAPEZOID_VELOCITY] Move along joint trajectory  
[IRPOS][TRAPEZOID_VELOCITY] Result: SUCCESSFUL  
kborkow@tola:~$ roslaunch irp6_bringup limits.py  
the rosdep view is empty: call 'sudo rosdep init' and 'rosdep update'  
[IRPOS] System ready  
[IRPOS][TRAPEZOID_VELOCITY] Move along joint trajectory  
[IRPOS][TRAPEZOID_VELOCITY] Result: MAX_VEL_UNREACHABLE  
[IRPOS][TRAPEZOID_VELOCITY] Max velocity unreachable. vel1=0.000000; vel2=0.0000  
00; vel_max=-0.300000; acc_max=-0.200000; displacement=-0.100000; minimal acc ne  
eded=-0.900000; in joint :1  
kborkow@tola:~$
```

Rysunek 2: Wiadomości zwrotne otrzymane przy pracy z nowymi funkcjami systemu IRPOS, w tym jedną z informacjami o błędzie i jego parametrach.

5.5 Prędkości graniczne

Założenia generatora trajektorii trapezoidalnej oraz jego implementacja w systemie IRPOS zakładają możliwość prędkości początkowej i końcowej różnej od zera. Niemniej jednak nie zaimplementowano funkcji api, która pozwalała na ich zadanie. Nie jest to potrzebne do zbadania właściwości kinematycznych robota. W razie potrzeby można rozszerzyć funkcjonalność klasy IRPOS.T bez zmiany kodu komponentów OROCOS’owych.

5.6 Funkcje pomocnicze

Metoda `trapezoid_error_code_to_string` zamienia otrzymany kod wiadomości zwrotnej od komponentu akcji na jej słowny odpowiednik. Umożliwia to poinformowanie użytkownika w czytelny sposób o wyniku wywołanej funkcji. Translacja jest zgodna z kodem zapisanym w wiadomości `TrapezoidGeneratorResult`, opisanej w sekcji TODO. Rysunek 1 przedstawia informacje jakie otrzymał użytkownik po wywołaniu dwa razy funkcji `move_along_joint_trajectory_trapezoid_velocity`. Za drugim razem zmieniono flagę `research_mode` na “True”, co spowodowało przedwczesne zakończenie pracy i wystosowanie odpowiedniej wiadomości błędu.

6. Wiadomości typu Trapezoid

Katalog `orocos_controllers` został wzbogacony o dwa nowe katalogi dotyczące przesyłania informacji pomiędzy modułami. Jeden z nich, `rtt_trapezoidal_trajectory_msgs`, to tylko nakładka pozwalająca używać nowe wiadomości w OROCOS’owym systemie czasu rzeczywistego. Drugi, `trapezoidal_trajectory_msgs`, zawiera definicje trzech typów wiadomości:

- `TrapezoidTrajectory.action`
- `TrapezoidGeneratorGoal.msg`
- `TrapezoidGeneratorResult.msg`

Pierwsza jest używana do komunikacji z IRPOS-api, pozostałe są stosowane w głębszych warstwach systemu.

6.0.1 `TrapezoidTrajectory.action`

Jak każda wiadomość opierająca się na tzw. “ROS action protocol”, stworzonym do działania z wywłaszczanymi zadaniami[1], `TrapezoidTrajectory.action` dzieli się na trzy części, w pliku definiującym przedzielone trzema poziomymi kreskami (“- -”), które opisuje [tablica 1](#). Natomiast wszystkie parametry i ich deskrypcje znajdują się w [tablicy 2](#).

nazwa	ścieżka przesyłu w IRPOS’ie	opis
Goal	IRPOS-api → komponent akcji	kontener informacji o specyfikacji żądania od użytkownika i programisty IRPOS-api
Result	komponent akcji → IRPOS-api	wynik wykonanych operacji przesyłany po zakończeniu ruchu
Feedback	komponent akcji → IRPOS-api	cyklicznie przesyłane wiadomości o pośrednich wynikach

Tablica 1: Opis części wiadomości typu “action”

nazwa części	nazwa	typ danych	opis
Goal	trajectory	trajectory_msgs/ JointTrajectory[2]	zadana trajektoria (punkty, prędkości i przyspieszenia końcowe, itp.)
	path_tolerance	control_msgs/ JointTolerance[] [3]	tolerancja błędu definiowana dla każdego silnika.
	research_mode	bool	flaga trybu badawczego, patrz 3.2.4
	duration_mode	bool	flaga trybu duration, patrz 3.2.3
	save_data	bool	flaga zapisu danych, patrz 3.2.4
	max_velocities	float64[]	prędkości do osiągnięcia przez silniki, działa tylko w trybie velocity, patrz 3.2.3
	max_accelerations	float64[]	przyspieszenia do osiągnięcia przez silniki, działa tylko w trybie velocity, patrz 3.2.3
Result	result	TrapezoidGeneratorResult	wiadomość tożsama z tą którą zwraca komponentowi akcji generator, stąd postanowiono użyć jej bezpośrednio; pełna definicja patrz 3.3.3
Feedback	header	Header	nagłówek
	joint_names	string[]	używane nazwy silników/stawów
	desired	trajectory_msgs/ JointTrajectoryPoint[4]	punkt zadany przez generator
	actual	trajectory_msgs/ JointTrajectoryPoint[4]	punkt osiągnięty przez robota
	error	trajectory_msgs/ JointTrajectoryPoint[4]	błąd regulacji

Tablica 2: Dokładny opis wszystkich zmiennych używanych w wiadomości TrapezoidTrajectory.action

6.1 TrapezoidGeneratorGoal.msg

TrapezoidGeneratorGoal.msg jest zwykłą wiadomością ROS'ową, która przesyłana jest od komponentu akcji do generatora. Jest ona tożsama z częścią Goal przedstawionej w tablicy 2. Rozdzielenie jest pozostałością po wersji, w której istniało dużo różnic między tymi elementami.

6.2 TrapezoidGeneratorResult.msg

Wynik ruchu jest sygnalizowany komponentowi akcji przez generator poprzez wysłanie wiadomości typu `TrapezoidGeneratorResult.msg`. Składa się ona z ciągu znaków, który jest wypełniony jeśli nastąpił błąd w trakcie obliczania profilu prędkościowego oraz kodu zwrotnego, którego znaczenie jest zdefiniowane w tym samym pliku. Pisemna wiadomość została stworzona po to by łatwiej było dobrać nastawy, które będą znajdować się w przestrzeni możliwości generatora trapezoidalnego. **Tabela 3** zawiera opis parametrów `TrapezoidGeneratorResult`, a **tabela 4** opis możliwych kodów. Warto wspomnieć, że pierwsze sześć kodów pokrywa się z używanymi przy generowaniu trajektorii spline'owych. Część `error_string` nie występuje w wiadomościach stosowanych przez stary generator.

nazwa	typ danych	opis
<code>error_string</code>	<code>string</code>	dane błędu
<code>error_code</code>	<code>int32</code>	kod błędu

Tablica 3: Parametry wiadomości `TrapezoidGeneratorResult`

kod	nazwa	opis
0	SUCCESSFUL	robot osiągnął pożądaną cel
-1	INVALID_GOAL	co najmniej jeden z zadanych punktów znajduje się poza przestrzenią osiągalną przez silnik/staw
-2	INVALID_JOINTS	nazwy silników/stawów lub ich liczba podane w wiadomości Goal nie pokrywają się z tymi, które otrzymano z serwera parametrów
-3	OLD_HEADER_TIMESTAMP	błędny czas w nagłówku
-4	PATH_TOLERANCE_VIOLATED	w trakcie ruchu błąd regulacji jest zbyt duży
-5	GOAL_TOLERANCE_VIOLATED	ostateczne położenie jest zbyt daleko od zadanego
-6	INVALID_LIMIT_ARRAY	w trybie prędkościowym podano mniej maksymalnych prędkości i/lub przyspieszeń, niż jest zadeklarowanych silników/stawów
-7	TRAJECTORY_NOT_FEASIBLE	jeden z warunków trajektorii trapezoidalnej nie został spełniony
-8	CANT_CALCULATE_COEFFS	błąd wystąpił przy obliczaniu współczynników profilu prędkościowego
-9	MAX_VEL_UNREACHEABLE	w przypadku zadanych warunków niemożliwe jest osiągnięcie maksymalnej prędkości, patrz 3.2.4
-10	BREACHED_POS_LIMIT	zadana trajektoria przechodzi (nie w miejscu początkowym lub końcowym) przez miejsce wykraczające poza limity silnika/stawu, możliwe tylko gdy prędkość początkowa/końcowa jest większa od tej jaką należy osiągnąć w fazie bez przyspieszenia
-11	ACC_TOO_SMALL_FOR_DURATION	niemożna osiągnąć celu w zadanym czasie, przy danym przyspieszeniu maksymalnym
-12	DURATION_TOO_LONG	trwanie ruchu zbyt długie, duży skok w fazie przyspieszania, występuje przy niektórych prędkościach początkowych/końcowych o wartości bezwzględnej większej od tej z fazy zerowego przyspieszenia
-13	DURATION_TOO_SHORT	trwanie ruchu zbyt krótkie, duży skok w fazie spowalniania, występuje przy niektórych prędkościach początkowych/końcowych o wartości bezwzględnej większej od tej z fazy zerowego przyspieszenia
-14	IMPOSSIBLE_VELOCITY	prędkość obliczona w trybie duration , jest o wartości bezwzględnej większej niż prędkość maksymalna.

Tablica 4: Parametry wiadomości **TrapezoidGeneratorResult**

7. Nowe komponenty OROCOS

Zaimplementowano dwa nowe komponenty OROCOS'owe. Komponent akcji oraz generator trajektorii. Ich kody znajdują się w katalogu `orocos_controllers`. W tej sekcji wytłumaczone zostanie ich działanie przy użyciu fragmentów kody źródłowego. Drobne zmiany jakie wprowadzono względem oryginalnego kodu (typu deklarowanie, konfiguracja nowych wektorów itp.) zostaną pominięte.

7.1 Komponent akcji

Zaimplementowany w plikach `InternalSpaceTrapezoidTrajectoryAction.cpp` i `InternalSpaceTrapezoidTrajectoryAction.hpp`, rozpoczyna swój cykl życia od zaimportowania kolejnych parametrów z serwera ROS'owego:

- `joint_names`, nazwy silników/stawów w zależności od przestrzeni pracy;
- `lower_limits`, dolne położenia możliwe do osiągnięcia przez silniki/stawy;
- `upper_limits`, górne położenia możliwe do osiągnięcia przez silniki/stawy.

Zgodnie ze strukturą komponentów OROCOS'owych, działających cyklicznie, moduł ma funkcję `updateHook` wraz z dwiema metodami pomocniczymi `configureHook` oraz `stratHook`, ich implementacja jest wymagana[5]. Obie funkcje konfiguracyjne wywoływane są w skrypcie `irp6-p-inside.ops`. Metoda `configureHook` inicjuje wektory danych do odpowiednich rozmiarów, oraz sprawdza, czy otrzymano odpowiednią liczbę ograniczeń. Natomiast funkcja uruchamiająca pracę komponentu uruchamia serwer akcji, ale ustawia flagę aktywnego celu na *false*, skąd komponent wie, że w danym momencie nie jest przeprowadzane przetwarzanie celu. Jednakże, by w pełni móc przedstawić działanie metody `updateHook`, należy najpierw opisać funkcję `goalCB`. Jest ona asynchronicznie wywoływana, w razie odebrania z wyższej warstwy systemu (IRPOS-api) części `Goal` z wiadomości `TrapezoidTrajectory.action`. Jeśli w danym momencie jest przetwarzana już jakaś trajektoria nowy cel zostaje odrzucony. W przeciwnym przypadku komponent wykonuje kolejne czynności:

1. wypełnienie `remapTable` odpowiedzialnej za zmapowanie nazw podanych w wiadomości na kolejność nadaną w serwerze parametrów. W razie niepowodzenia, czyli niezgodnej liczby nazw silników/stawów, wysyłany jest kod błędu `INVALID_JOINTS`;
2. sprawdzenie czy wszystkie zadane punkty mieszczą się w ograniczeniach, w wypadku niespełnienia tego warunku w którymkolwiek punkcie wysyłany jest błąd `INVALID_GOAL`;
3. jeśli użytkownik ustawił flagę `duration_mode` na *false*, sprawdzona zostaje liczba zadanych prędkości i przyspieszeń maksymalnych, niezgodna z zadeklarowaną liczbą silników/stawów powoduje wysłanie wiadomości o błędzie `INVALID_LIMIT_ARRAY`;
4. na podstawie `remapTable` zmapowane zostają wszystkie zadane, łącznie z tablicami ograniczeń kinematycznych;
5. sprawdzona zostaje poprawność czasu podanego w nagłówku, błąd wywołuje odesłanie kodu `OLD_HEADER_TIMESTAMP`;

6. jeśli uda się połączyć ze wszystkimi komponentami równoległymi wypełniona i wysłana do generatora zostaje wiadomość typu `TrapezoidGoal`.

Niespełnienie któregokolwiek z warunków generujących błąd lub problem przy uruchomieniu równoległych przerywa przetwarzanie wiadomości i wywołuje odrzucenie zadanego celu. Nowością względem komponentu akcji trajektorii spline'owej jest przetwarzanie maksymalnych prędkości i przyspieszeń oraz przepisywanie ich wraz z flagami. Dla wygody przebudowano stary kod i przeniesiono jego fragmenty do dedykowanych funkcji.

Metoda `updateHook` (zgodnie z ustawieniami z dnia 30 maja 2018) wywoływana jest co 0.01 sekundy. Zadania jakie kolejno wykonuje to:

1. odczytuje dane o właściwym i zadanym położeniu silników/stawów;
2. sprawdza czy generator zasygnalizował zakończenie wykonywania trajektorii i reaguje zależnie od otrzymanego wyniku oraz ułożenia robota;
3. analizuje możliwość przekroczenia tolerancji ścieżki (pomijane w trybie `research_mode`);
4. wysyła część Feedback z wiadomości typu `action`.

Naturalnie punkt drugi wykonywany będzie najrzadziej oraz zablokuje wykonywanie kolejnych czynności. Ponad to punkt trzeci i czwarty zostaną aktywowane tylko jeśli aktywny jest cel oraz otrzymano nowe dane o konfiguracji robota. Wykrycie przekroczenia ograniczeń ścieżki lub celu skutkuje odesłaniem do użytkownika odpowiedniej wiadomości błędu (patrz `TODO`) oraz odrzuceniem celu. Wygenerowane informacje o błędach przy obliczaniu profili prędkościowych powodują tą samą reakcję. Ponownie wprowadzoną zmianą jest przebudowanie kodu, w tym oddelegowanie części zadań do odpowiednich nazwanych funkcji oraz sposób ustalania zakończenia pracy nad celem (zamiast sprawdzania, czy minął zadany czas ruchu, komponent oczekuje na wiadomość od generatora).

7.2 Generator trajektorii

Kod źródłowy generatora rozłożony jest na dwa pliki: `InternalSpaceTrapezoidTrajectoryGenerator.cpp` i `velocityprofile_trapezoid.cpp` oraz ich pliki nagłówkowe. Jest to zgodne z konwencją przyjętą dla generatora spline'owego. Komponent `InternalSpaceTrapezoidTrajectoryGenerator` w trakcie konfiguracji zwiększa wektor `vel_profile` zgodnie z podaną w parametrach liczbą silników/stawów, co równoważne jest wywołaniu tej samej liczby konstruktorów domyślnych klasy `velocityprofile_trapezoid`. Stąd komunikacja między komponentem, a jego obiektami profili prędkościowych opiera się na wywoływaniu zadanych funkcji i zwracania wartości, które odpowiadają wiadomościom podanym w [Tablicy 4](#). Wpierw opisane zostanie działanie komponentu, a następnie jego klasy z wektora `vel_profile`.

7.2.1 InternalSpaceTrapezoidTrajectoryGenerator

Tak samo jak w przypadku komponentu akcji, konstruktor inicjuje pola klasy wartościami domyślnymi, konfiguruje porty komponentu oraz pobiera kolejne parametry z serwera ROS'owego:

- `number_of_joints`, liczba silników/stawów;

- `ns_interval`, parametr tolerancji opóźnień/przyspieszeń wywołania funkcji `updateHook`;
- `lower_limits`, dolne położenia możliwe do osiągnięcia przez silniki/stawy;
- `upper_limits`, górne położenia możliwe do osiągnięcia przez silniki/stawy;
- `max_velocities`, prędkości maksymalne (obecnie bezpieczne wartości, które po badaniach zostaną zamienione na właściwe);
- `max_accelerations`, przyspieszenia maksymalne (obecnie bezpieczne wartości, które po badaniach zostaną zamienione na właściwe).

W przypadku generatora mamy do czynienia tylko z czterema funkcjami właściwymi komponentom OROCOS'owym. Są to metody `*Hook`. Funkcję konfiguracyjną `configureHook` można pokrótce opisać jako inicjalizującą porty (`setDataSample`) oraz zmienne zależne od parametrów pobranych z serwera. Wywoływana jest ona na etapie przetwarzania skryptu `irp6-p-inside.opt`. Warto zauważyć, że w momencie uruchamiania skryptu użytkownika komponent wciąż nie jest gotowy do synchronicznej pracy, bo w przeciwieństwie do `InternalSpaceTrapezoidTrajectoryAction`, pliki wywoływane przez `roslaunch irp6.bringup irp6-*-nohardware.launch` nie zawierają komendy start dla niego. Dopiero gdy wywołana zostanie jedna z metod opisanych na początku [sekcji 4](#), poprzez `connmanSwitch` aktywuje generator. Zapewnia to brak konfliktów między różnymi generatorami, korzystającymi z tych samych portów. Dopiero teraz metoda `statrHook` sygnalizuje swoją aktywność przez porty synchronizacji. Zmiany jakie wprowadzono do wyżej wymienionych funkcji są drobne i polegają na inicjalizacji dodanych pól.

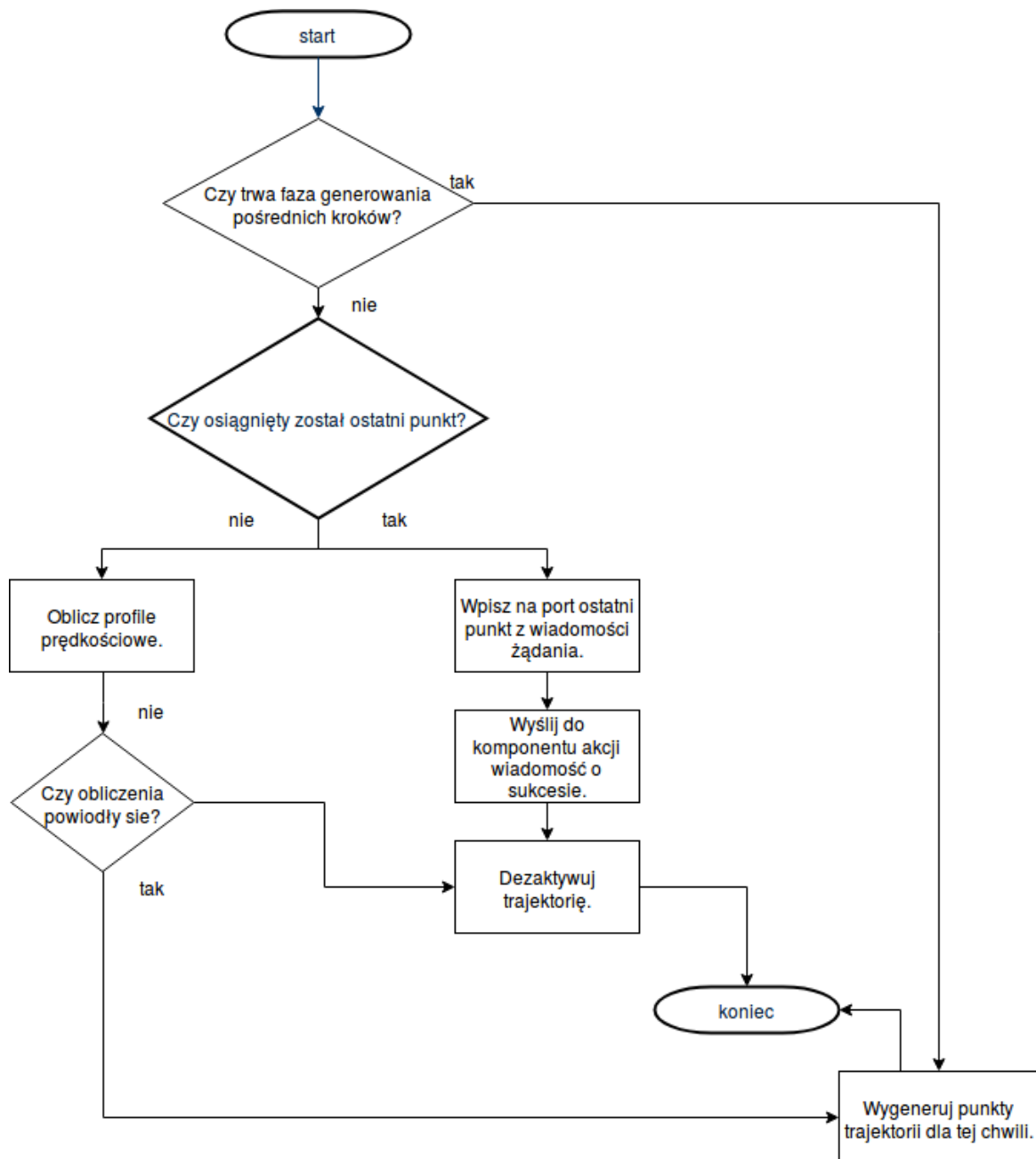
Najważniejszą częścią komponentu jest metoda `updateHook`. Każde jej wywołanie pociąga za sobą zasygnalizowanie o aktywności komponentu oraz sprawdzenie czy nadesłany został nowy cel. Niezależnie od wyniku poprzedniego testu, czyli od tego czy w danym momencie zmienna zadanych punktów trajektorii pochodzi z pierwszego odczytu położenia silników/stawów, czy z obliczeń klas profili prędkościowych, na port zadanych wpisywana jest zawartość wektora `setpoint_`. Co więcej przy flagie `save_data` ustawionej na `true` przed wysłaniem danych zostaną one zapisane do wektorów-kontenerów. Ze względu na naturę procesów czasu rzeczywistego predefiniowana jest wielkość wektorów, więc przy długim czasie pracy generatora może dojść do niepełnego zapisu. Nadejście nowej wiadomości od komponentu akcji skutkuje przepisaniem danych z niej do pól obiektu generatora oraz ustawieniem wartości zmiennej `trajectory_active_` na `true`. Ta flaga, obok poprawności nagłówka wiadomości, jest sprawdzana przed rozpoczęciem jakichkolwiek obliczeń. Zachowano z generatora spline'owego nadpisywanie w przypadku gdy w trakcie pracy nad jednym celem przyjdzie nowe żądanie.

Gdy aktywny jest cel, w zależności od trybu zdefiniowanego przez użytkownika, uruchamiana jest jedna z dwóch metod:

- `updateHookWithVelocityBasedProfiles`;
- `updateHookWithDurationBasedProfiles`.

Ich działanie niewiele różni się od siebie. Postanowiono zastosować oddzielne funkcje ze względu na ewentualne łatwiejsze wprowadzanie oddzielnych, bardziej prawdopodobnych zmian. Schemat działania (uogólniony dla obu trybów) pokazany jest na [rysunku 2](#). Każdy z kroków przedstawiony na tym schemacie (poza start i koniec) wiąże się z wywołaniem podfunkcji. Obliczanie profili prędkościowych wiąże się z wywołaniem odpowiedniej dla trybu działania metody

obiektu klasy `velocityprofile_trapezoid`. Dodatkowo niepowodzenie przy obliczaniu profilu wywołuje wysłanie wiadomości o niepowodzeniu do komponentu akcji, a przy fladze `save_data` ustawionej na `true` dezaktywacja trajektorii powoduje zapis danych na dysk.



Rysunek 3: Schemat pracy uogólniony dla obu trybów pracy `updateHook`.

Różnice pomiędzy działaniem funkcji dla trybu prędkościowego i czasowego podsumowane są w [tablicy 5](#).

etap	tryb czasowy	tryb prędkościowy
sprawdzenie czy trwa faza generowania kroków;	testowanie czy minął czas przeznaczony na daną fazę;	testowanie czy wszystkie flagi aktywności silników/stawów ustawione są na <i>false</i> ;
obliczanie profilu prędkościowego	poprzedzone poszukiwaniem, który z silników/stawów ma do przebycia najdłuższą trasę i przypisanie tego czasu zadanemu okresowi ruchu dla wszystkich silników/stawów; wykorzystuje metodę <code>setProfileDuration</code> klasy <code>velocityprofile.trapezoid</code> ;	wykorzystuje metodę <code>setProfileVelocity</code> klasy <code>velocityprofile.trapezoid</code>

Tablica 5: Różnice pomiędzy działaniem funkcji `updateHook` w zależności od ustawionego trybu

Komponent, choć wzorowano się na odpowiedniku spline’owym, poza funkcjami konfiguracyjnymi został stworzony od nowa.

7.2.2 `velocityprofile.trapezoid`

Trzy zadania generatora zostały oddelegowane do klasy `velocityprofile.trapezoid`:

- sprawdzenie poprawności żądania;
- obliczenie współczynników profilu prędkościowego;
- obliczanie pozycji/prędkości/przyśpieszenia dla danego momentu.

Tablica 6 przedstawia jakie warunki muszą zostać spełnione by żądanie zostało uznane za wykonywalne. Odpowiadają one błędom od -7 do -14 z **tablicy 4**.

warunek	tryb pracy	opis
$T \geq T_a + T_d$	czasowy	jeśli trajektoria określana jest na podstawie czasu trwania, przyspieszenie maksymalne musi być wystarczająco duże by czas fazy przyspieszenia i zwalniania łącznie nie przekroczyły czasu trwania ruchu;
$ v \leq v_{max}$	czasowy	obliczona prędkość nie może być większa od maksymalnej
$v_{max} \geq 0$	czasowy	ustalono konwencję, wg. której należy podawać prędkości maksymalne jako wartości bezwzględne
$v_{max} > 0$	prędkościowy	oprócz konwencji podawania wartości bezwzględnej, warunek ten chroni przed podaniem zera do obliczania czasu trwania ruchu, skutkowałoby to dzieleniem przez zero;
$v_i \wedge v_f = 0$	badawczy	ustalono konwencję, wg. której w trybie badawczym prędkości skrajne muszą być zerami
$a_{max}h < \frac{ v_0^2 - v_0'^2 }{2}$	wszystkie	przy dany przyspieszeniu maksymalnym, dystans może być za krótki by można było go przebyć przy zachowaniu podanych wartości granicznych prędkości;
$T > 0$	wszystkie	sprawdzanie przed obliczeniem współczynników, w tym momencie taki czas trwania sygnalizuje błąd obliczeń; zerowy czas trwania, czyli brak ruchu, powinien zostać wykryty wcześniej;
$T \neq 0$	czasowy	czas trwania ruchu musi być większy od zera, jeśli jest równy zeru od razu wysyłana zostaje wiadomość o dotarciu do celu;
$ s \geq \varepsilon$	prędkościowy	droga musi być większa niż ustalona granica błędu położenia, jeśli jest mniejsza od razu wysyłana zostaje wiadomość o dotarciu do celu;
brak skoków trajektorii w punktach T_a i T_d	wszystkie	istnieje możliwość, iż przy zadaniu prędkości granicznych większych niż prędkość fazy zerowego przyspieszenia, nastąpi skok trajektorii w punktach zmiany fazy; jest to wykrywane po obliczeniu współczynników, dzięki czemu, kilka przypadków takiego określenia prędkości zostaje przyzwolonych;
brak przekroczeń położień granicznych	wszystkie	istnieje możliwość, że przy niezerowych prędkościach granicznych ich osiągnięcie wiąże się z przejściem przez punkty położone za celem;
$T > T_a + T_d$	prędkościowy-badawczy	jeśli czas trwania jest równy czasowi trwania faz przyspieszenia i spowolnienia nie występuje faza zerowego przyspieszenia, więc niemożliwe jest zbadanie prędkości maksymalnej;

Tablica 6: Warunki realizowalności żądania użytkownika

Współczynniki profilu prędkościowego obliczane są na podstawie wzorów z [sekcji 2](#). Natomiast obliczanie położenia dla danej chwili następuje przy wywołaniu publicznej funkcji przez klasę główną generatora. Przemnażają one współczynniki przez kolejne potęgi podanej chwili t .

8. Testowanie

Testowanie generatora rozpoczęto od sprawdzenia, czy działał on przy najprostszych przykładach. Następnie przetestowano wykrywanie nierealizowanych zadanych. Każda próba została poprzedzona oceną zadanych za pomocą skryptów Matlab'owych. Dla funkcji działających w przestrzeni stawów dodatkowo sprawdzano czy przeniesienie trajektorii do przestrzeni silników nie spowoduje problemów. Wyniki oceniano na podstawie zebranych danych o położeniach trajektorii wytworzonej przez generator, ostatecznego wyniku pobranego przez IRPOS-api oraz na podstawie obrazu wizualizacji. Każda z funkcji przedstawionych w [sekcji 5](#) została przetestowana oddzielnie. Zgodnie z założeniami przedstawionymi w [sekcji 5.5](#), nie testowano nadania prędkości granicznych.

8.1 `move_to_motor_position_trapezoid_velocity`

Testowanie funkcji działającej w przestrzeni stawów w trybie prędkościowym.

8.1.1 Proste działanie w trybie niebadawczym

Początkowo zadano nastawy, które wymuszały ruch tylko jednego silnika. Podane są w [tablicy 7](#), każdy wiersz reprezentuje oddzielną symulację. Następnie zastosowano te same nastawy, ale tym razem jednocześnie. W kolejnym kroku przeprowadzono symulację ruchu w drugą stronę.

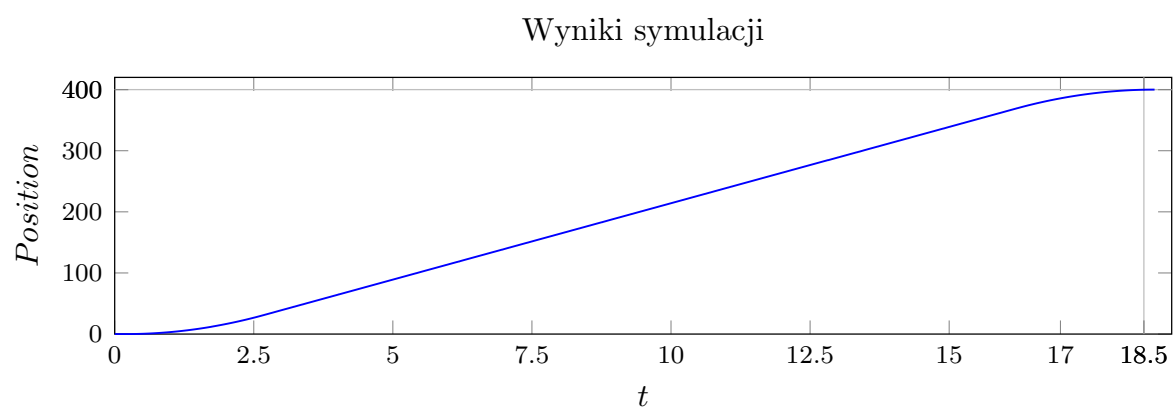
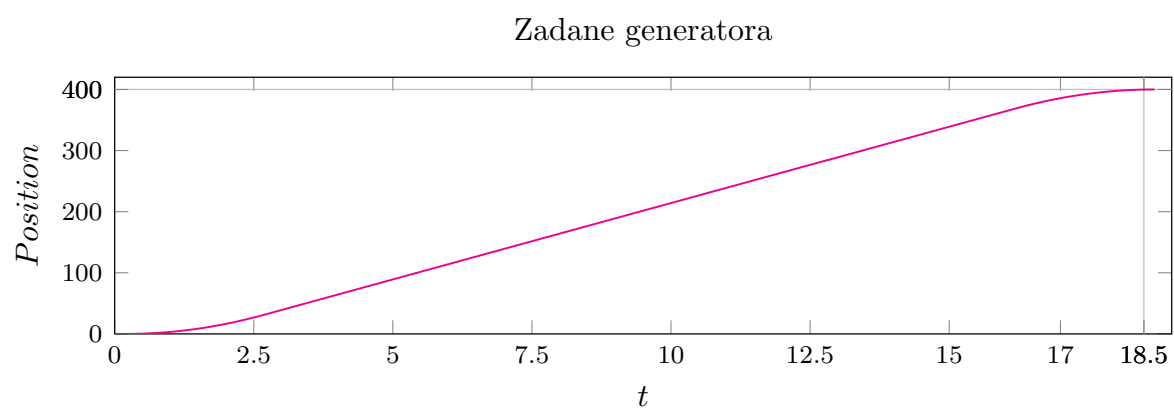
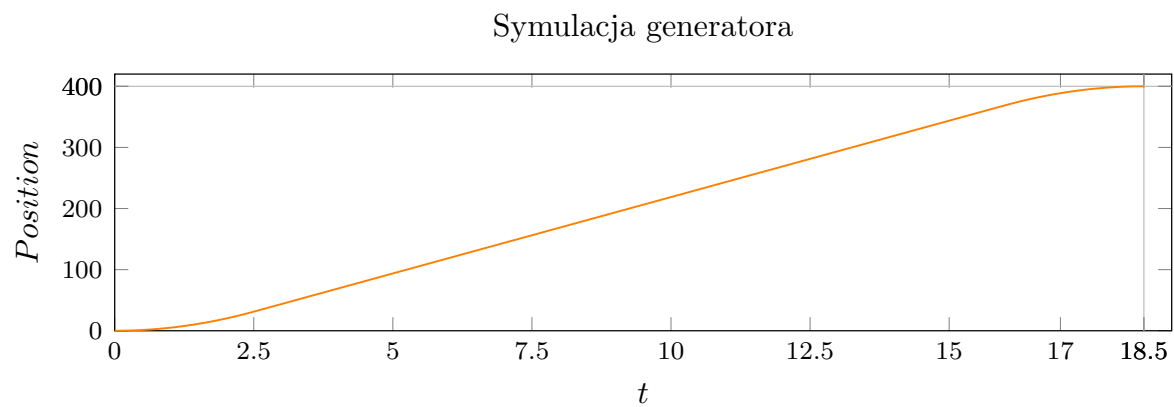
Ze względu na liczbę symulacji wykresy z wynikami pierwszych dwóch testów przedstawiono tylko dla jednego silnika. [Rysunek 6](#) jest zbiorem wykresów kolejnych stawów, gdy uruchomiono zmianę położenia z [tablicy 7](#) jednocześnie. Widać na nim, że zgodnie z założeniami trybu prędkościowego każdy staw kończy poruszanie się w innym momencie.

silnik	q_i	q_f	v_i	v_f	v_{max}	a_{max}	wynik symulacji
1	0.0	400.0	0.0	0.0	25.0	10	✓
2	0.0	100.0	0.0	0.0	35.0	20	✓
3	0.0	100.0	0.0	0.0	35.0	20	✓
4	0.0	350.0	0.0	0.0	35.0	20	✓
5	0.0	400.0	0.0	0.0	35.0	20	✓
6	0.0	2000.0	0.0	0.0	35.0	20	✓

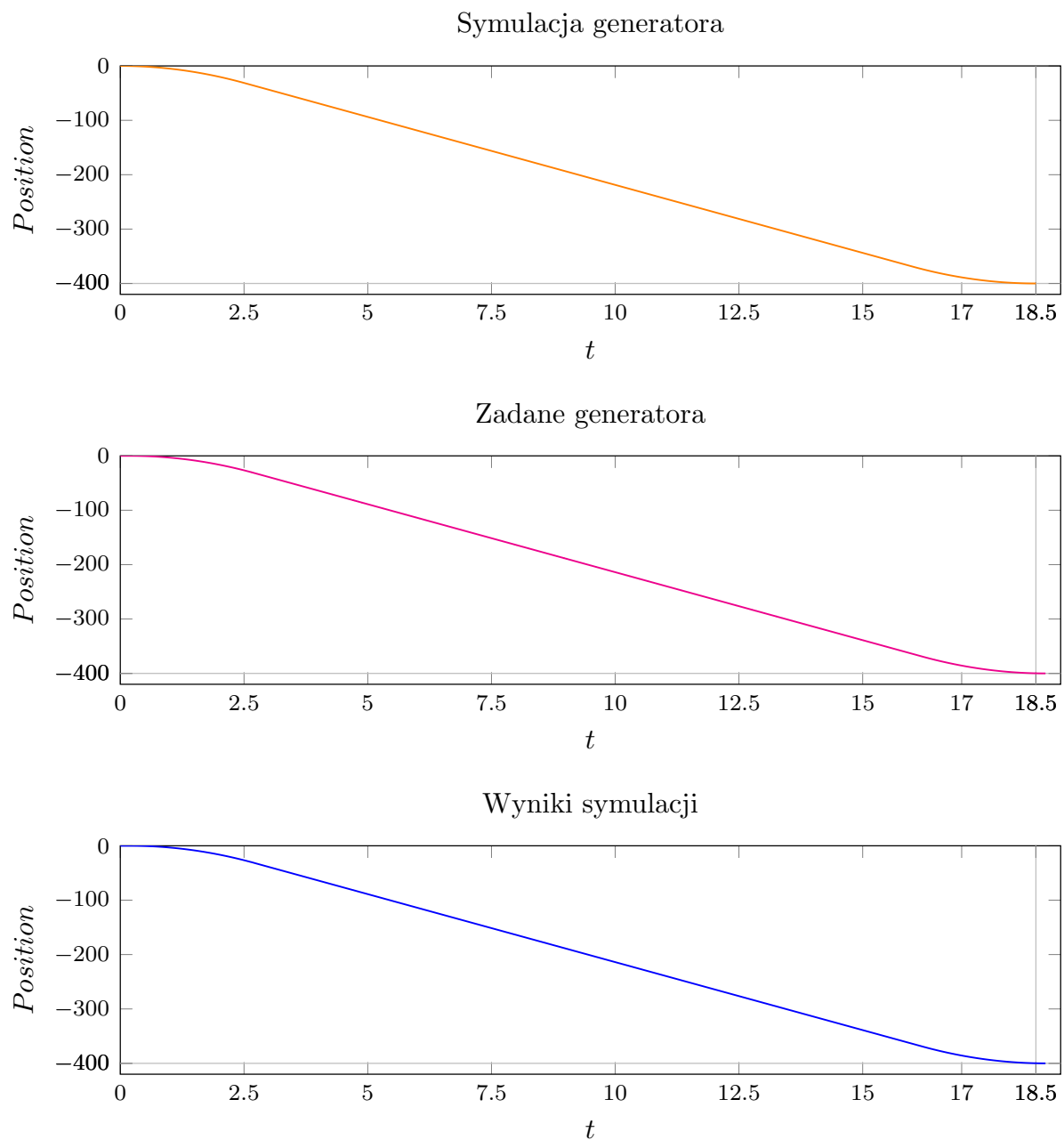
Tablica 7: Wyniki symulacji najprostszego przypadku, badanie oddzielnych silników.

silnik	q_i	q_f	v_i	v_f	v_{max}	a_{max}	wynik symulacji
1	0.0	-400.0	0.0	0.0	25.0	10	✓
2	0.0	-100.0	0.0	0.0	35.0	20	✓
3	0.0	-70.0	0.0	0.0	35.0	20	✓
4	0.0	-60.0	0.0	0.0	35.0	20	✓
5	0.0	-70.0	0.0	0.0	35.0	20	✓
6	0.0	-900.0	0.0	0.0	35.0	20	✓

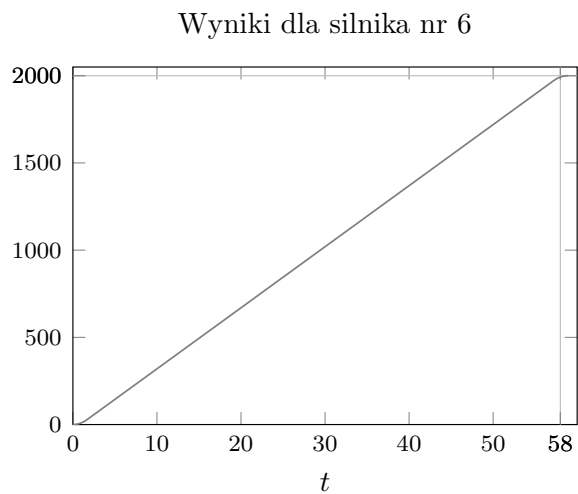
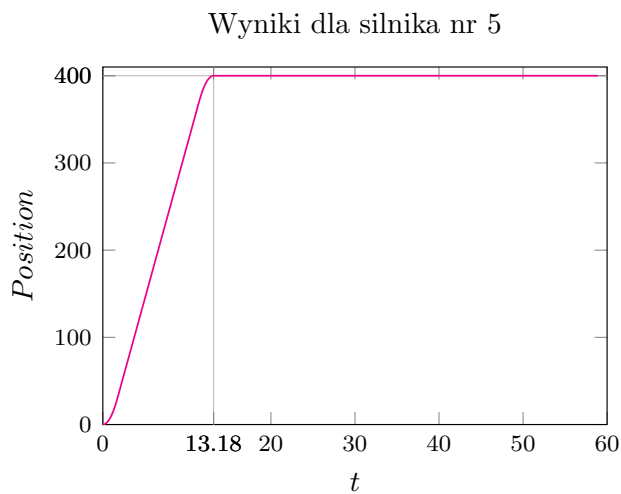
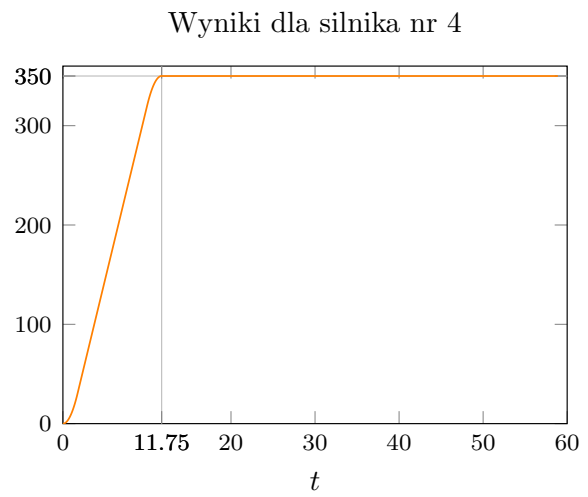
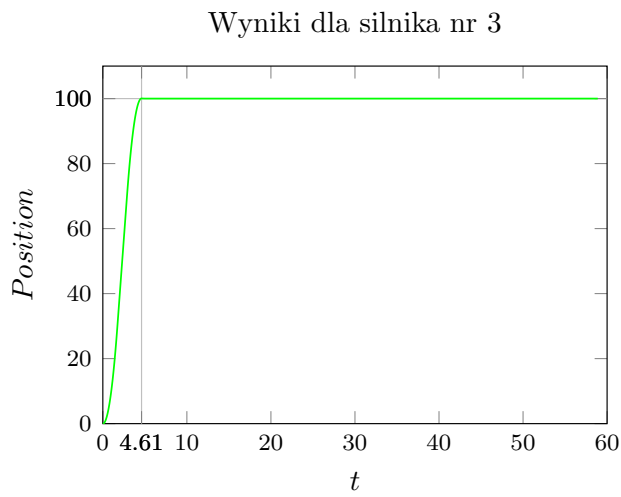
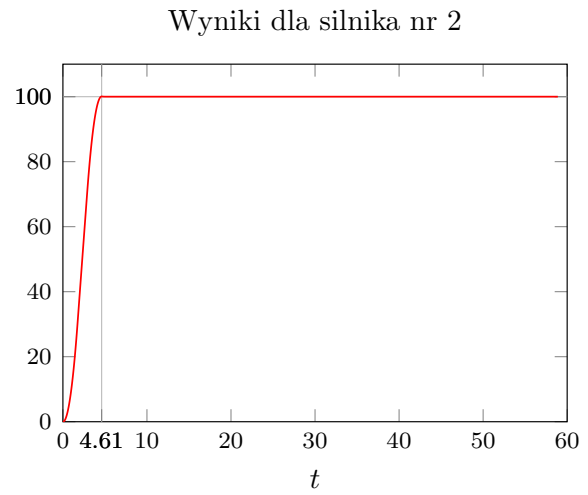
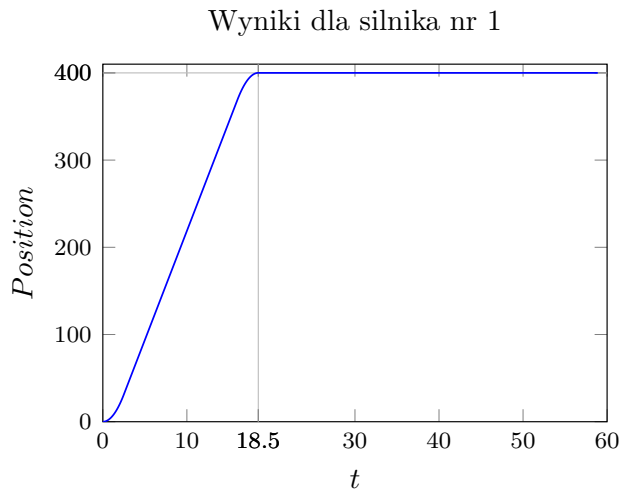
Tablica 8: Wyniki symulacji najprostszego przypadku, badanie oddzielnych silników.



Rysunek 4: Wyniki testu trajektorii dla silnika pierwszego funkcji `move_to_motor_position_trapezoid_velocity` przy nastawach z [tablicy 7](#).



Rysunek 5: Wyniki testu trajektorii dla silnika pierwszego funkcji `move.to_motor_position_trapezoid_velocity` przy nastawach z [tablicy 8](#).



Rysunek 6: Wyniki testu trajektorii dla silnika pierwszego funkcji `move_to_motor_position_trapezoid_velocity` przy nastawach z [tablicy 8](#).

8.1.2 Proste działanie w trybie badawczym

Porównano wyniki dla z poprzedniej sekcji, tym razem uruchamiając tryb badawczy. Spodziewano się, że tylko w przypadku ruchu wstecznego na silniku 4, odległość do przebycia będzie zbyt krótka do osiągnięcia maksymalnej prędkości, ergo odesłana zostanie wiadomość **MAX_VEL_UNREACHABLE**.

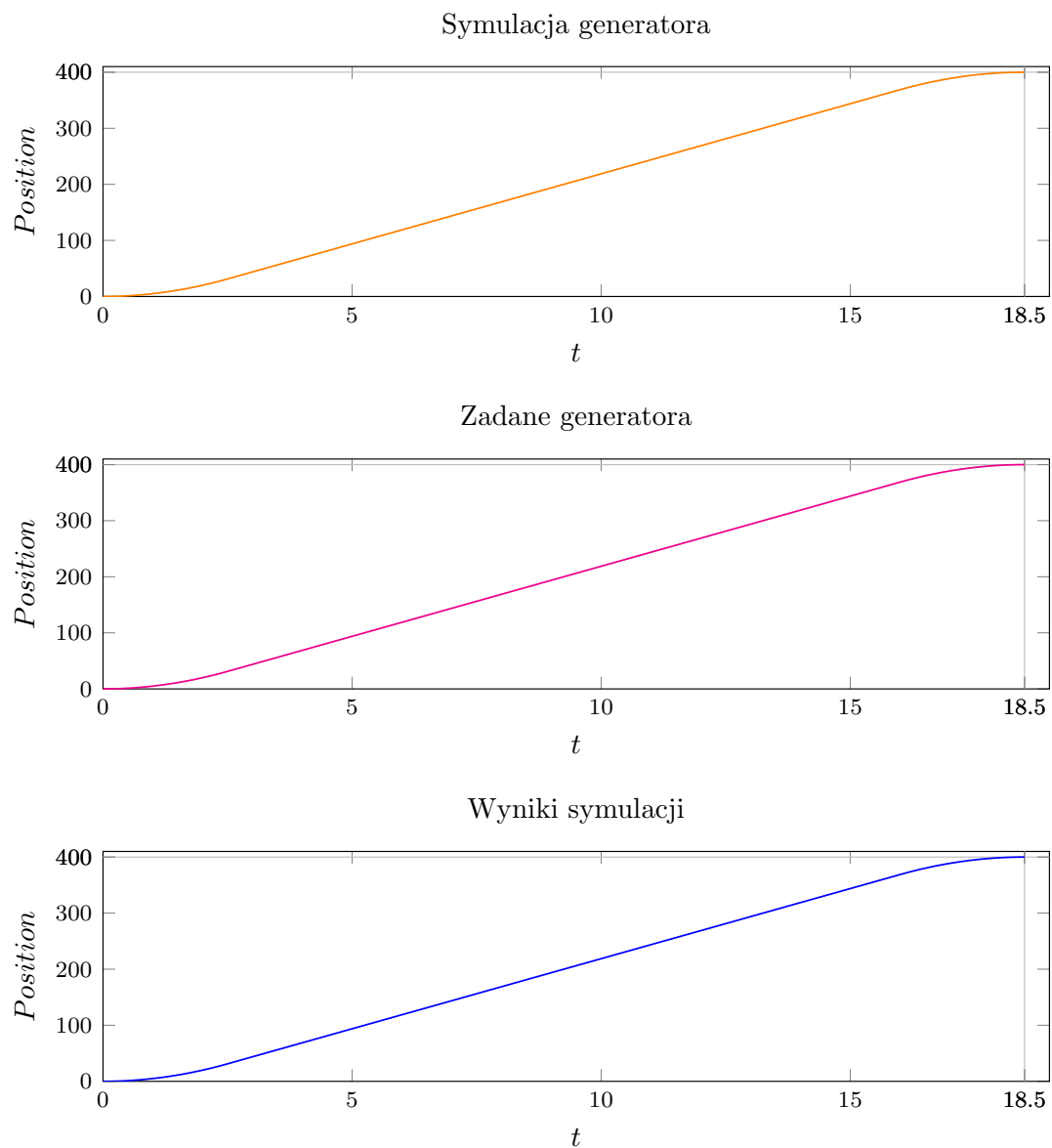
silnik	q_i	q_f	v_i	v_f	v_{max}	a_{max}	wynik symulacji
1	0.0	400.0	0.0	0.0	25.0	10	✓
2	0.0	100.0	0.0	0.0	35.0	20	✓
3	0.0	100.0	0.0	0.0	35.0	20	✓
4	0.0	350.0	0.0	0.0	35.0	20	✓
5	0.0	400.0	0.0	0.0	35.0	20	✓
6	0.0	2000.0	0.0	0.0	35.0	20	✓

Tablica 9: Wyniki symulacji najprostszego przypadku, badanie oddzielnych silników. Tryb badawczy.

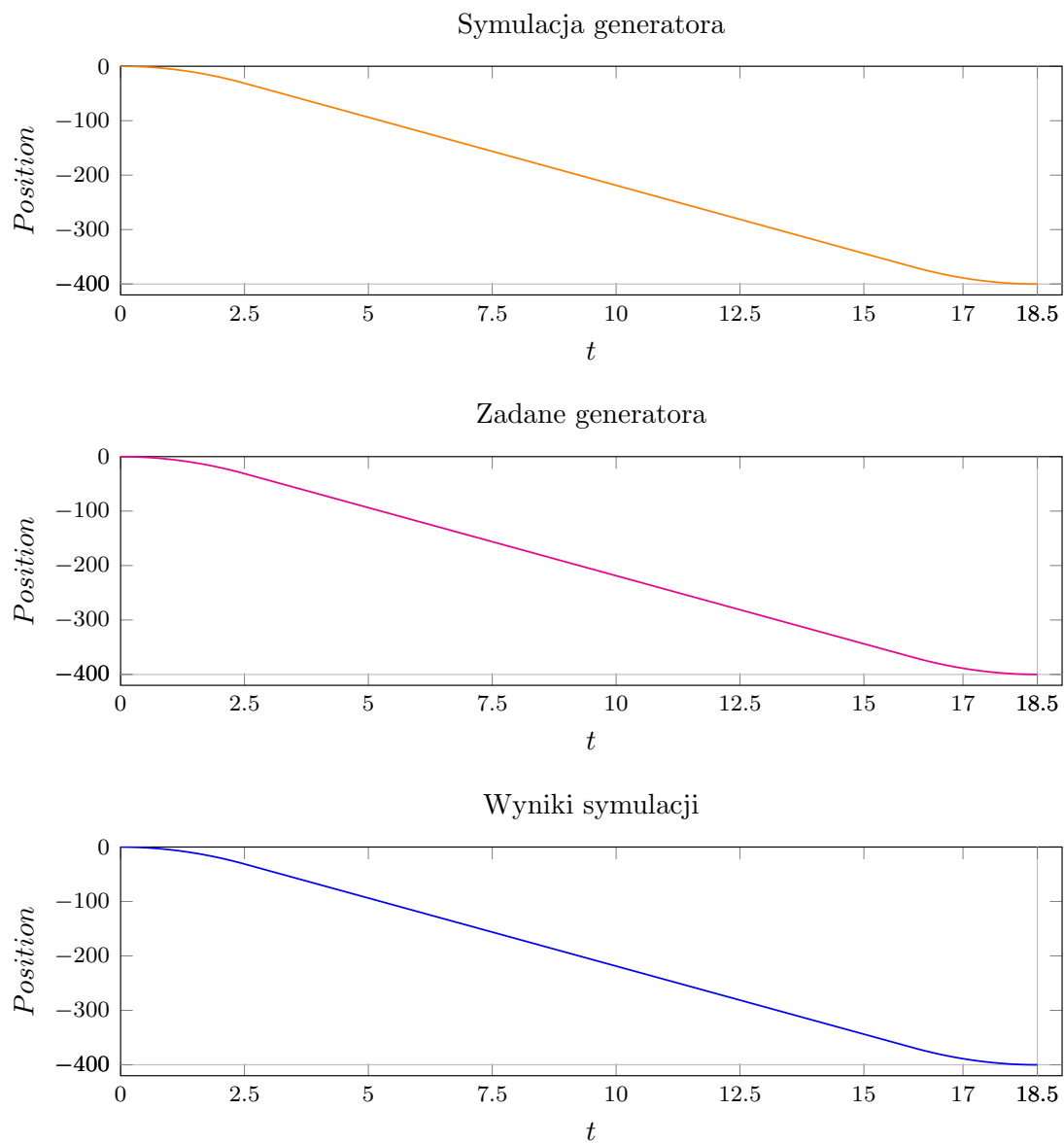
silnik	q_i	q_f	v_i	v_f	v_{max}	a_{max}	wynik symulacji
1	0.0	-400.0	0.0	0.0	25.0	10	✓
2	0.0	-100.0	0.0	0.0	35.0	20	✓
3	0.0	-70.0	0.0	0.0	35.0	20	✓
4	0.0	-60.0	0.0	0.0	35.0	20	X
5	0.0	-70.0	0.0	0.0	35.0	20	✓
6	0.0	-900.0	0.0	0.0	35.0	20	✓

Tablica 10: Wyniki symulacji najprostszego przypadku, badanie oddzielnych silników. Tryb badawczy.

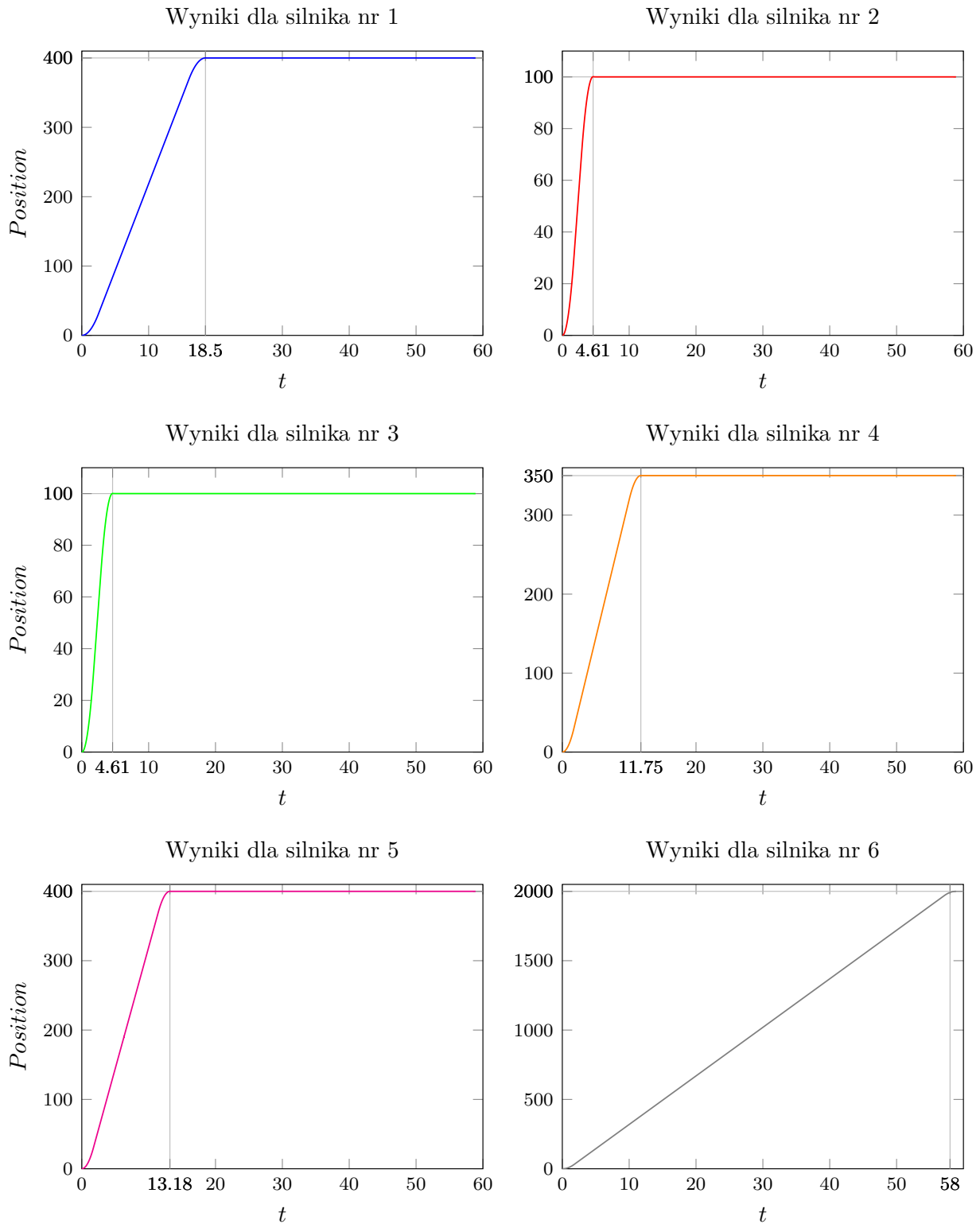
Wyniki uzyskane dla trybu badawczego w postaci wykresów ([rysunek 7](#),[rysunek 8](#),[rysunek 9](#)), są tożsame z wynikami z [sekcji 8.1.1](#).



Rysunek 7: Wyniki testu trajektorii dla silnika pierwszego funkcji `move_to_motor_position_trapezoid_velocity` przy nastawach z [tablicy 10](#). Tryb badawczy.



Rysunek 8: Wyniki testu trajektorii dla silnika pierwszego funkcji `move_to_motor_position_trapezoid_velocity` przy nastawach z [tablicy 11](#). Tryb badawczy.



Rysunek 9: Wyniki testu trajektorii dla silnika pierwszego funkcji `move_to_motor_position_trapezoid_velocity` przy nastawach z [tablicy 10](#). Tryb badawczy.

Jak wspomniano, spodziewano się błędu na silniku 4, przy ruchu $q_i > q_f$. Takowy otrzymano, co potwierdza [rysunek 10](#) udokumentowujący go. Warto zaznaczyć, że system numeruje silniki

od zera, więc liczba porządkowa jest mniejsza o jeden.

```
[IRPOS][TRAPEZOID_VELOCITY] Move to motor position
[IRPOS][TRAPEZOID_VELOCITY] Result: MAX_VEL_UNREACHABLE
[IRPOS][TRAPEZOID_VELOCITY] Max velocity unreachable. vel1=0.000000; vel2=0.0000
00; vel_max=-35.000000; acc_max=-20.000000; displacement=60.000000; minimal acc
needed=20.416667; in joint :3
```

Rysunek 10: Wyniki testu trajektorii dla silnika pierwszego funkcji `move_to_motor_position_trapezoid_velocity` przy nastawach dla silnika 4 z [tablicy 11](#). Tryb badawczy.

Otrzymane wykresy są tożsame z wykresami otrzymanymi w sekcji 8.1.1, co łącząc z wynikami ruchu odwrotnego silnika 4, pozwala stwierdzić, że tryb badawczy działa zgodnie z założeniami.

8.2 `move_along_motor_trajectory_trapezoid_velocity`

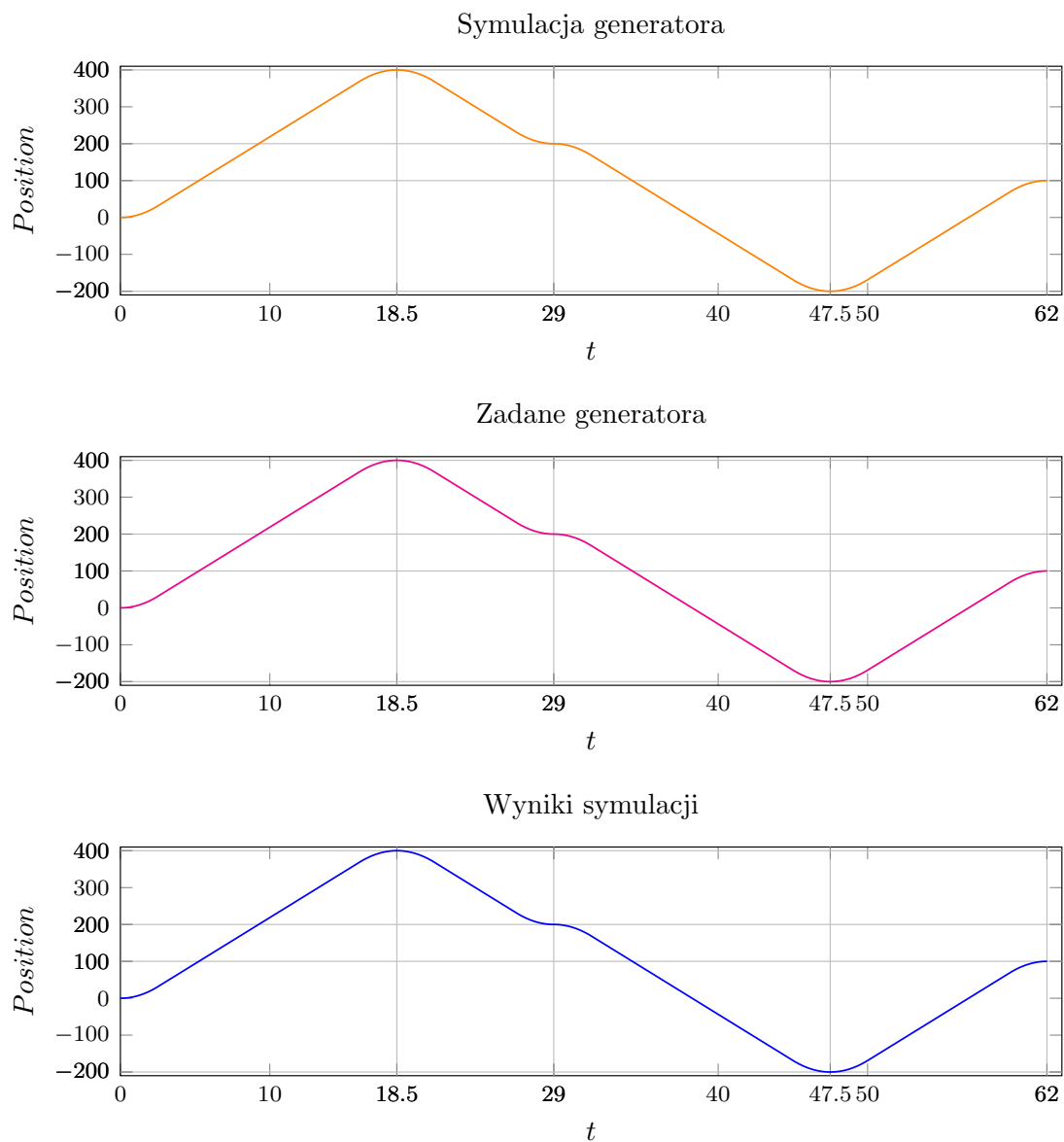
Testowanie funkcji działającej w przestrzeni stawów w trybie prędkościowym, pozwalającej na osiągnięcie kilku kolejnych punktów przy jednym wywołaniu funkcji.

8.2.1 Proste działanie w trybie niebadawczym

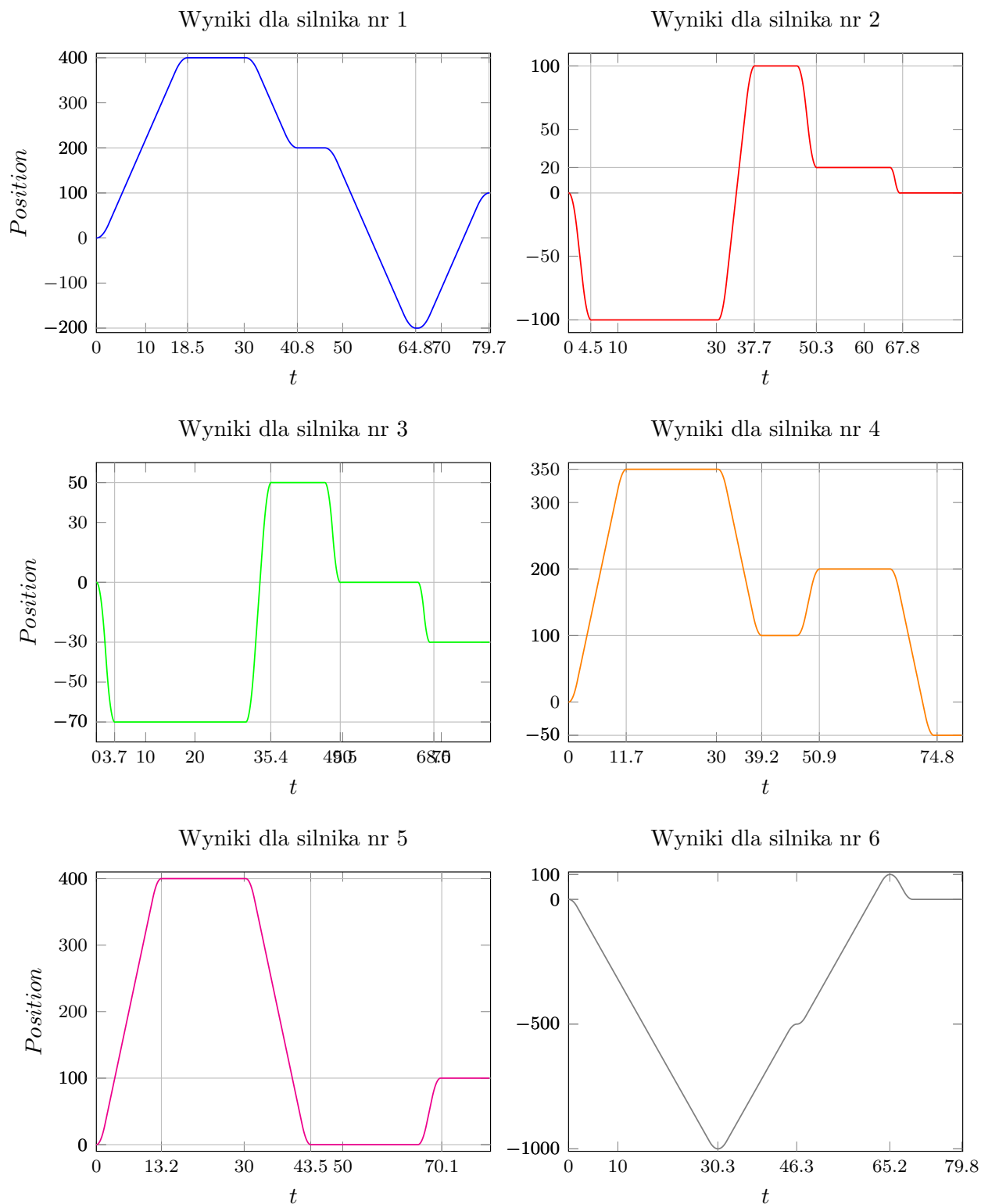
Ponownie przeprowadzono najpierw badania trajektorii oddzielnie dla każdego silnika, następnie dla wszystkich naraz. Nie przeprowadzano badań oddzielnie dla ruchu $q_i > q_f$ ze względu na zawarcie ich w trajektorii zadanej. Zadane przedstawiono w [tablicy 11](#). Pominęto w niej prędkości początkowe i końcowe, które nie podgalaają badaniu.

silnik	q_i	q_2	q_3	q_4	q_f	v_{max}	a_{max}	wynik symulacji
1	0.0	400.0	200.0	-200.0	100.0	25.0	10	✓
2	0.0	-100.0	100.0	20.0	0.0	35.0	20	✓
3	0.0	-70.0	50.0	0.0	-30.0	35.0	20	✓
4	0.0	350.0	100.0	200.0	-50.0	35.0	20	✓
5	0.0	400.0	0.0	0.0	100.0	35.0	20	✓
6	0.0	-1000.0	-500.0	100.0	0.0	35.0	20	✓

Tablica 11: Wyniki symulacji najprostszego przypadku, badanie oddzielnych silników. Tryb badawczy.



Rysunek 11: Wyniki testu trajektorii dla silnika pierwszego funkcji `move_along_motor_trajectory_trapezoid_velocity` przy nastawach z [tablicy 11](#).

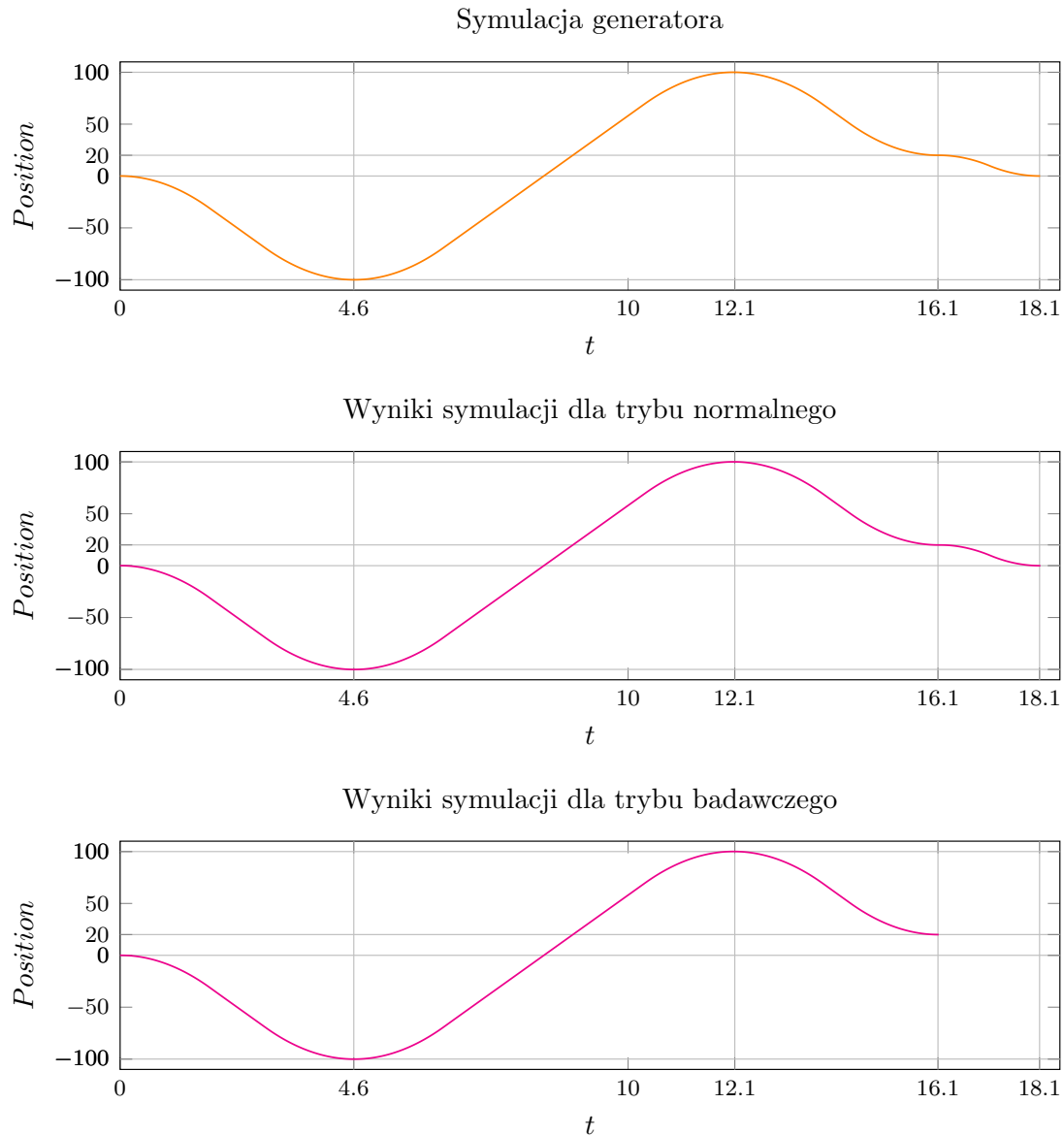


Rysunek 12: Wyniki testu trajektorii dla wszystkich silników funkcji `move.along_motor_trajectory_trapezoid_velocity` przy nastawach z [tablicy 11](#).

Jak widać na [rysunku 12](#), silniki czekają na siebie. Wizualizują to wypłaszczenia na wykresach. Jest to zgodne z założeniami generatora.

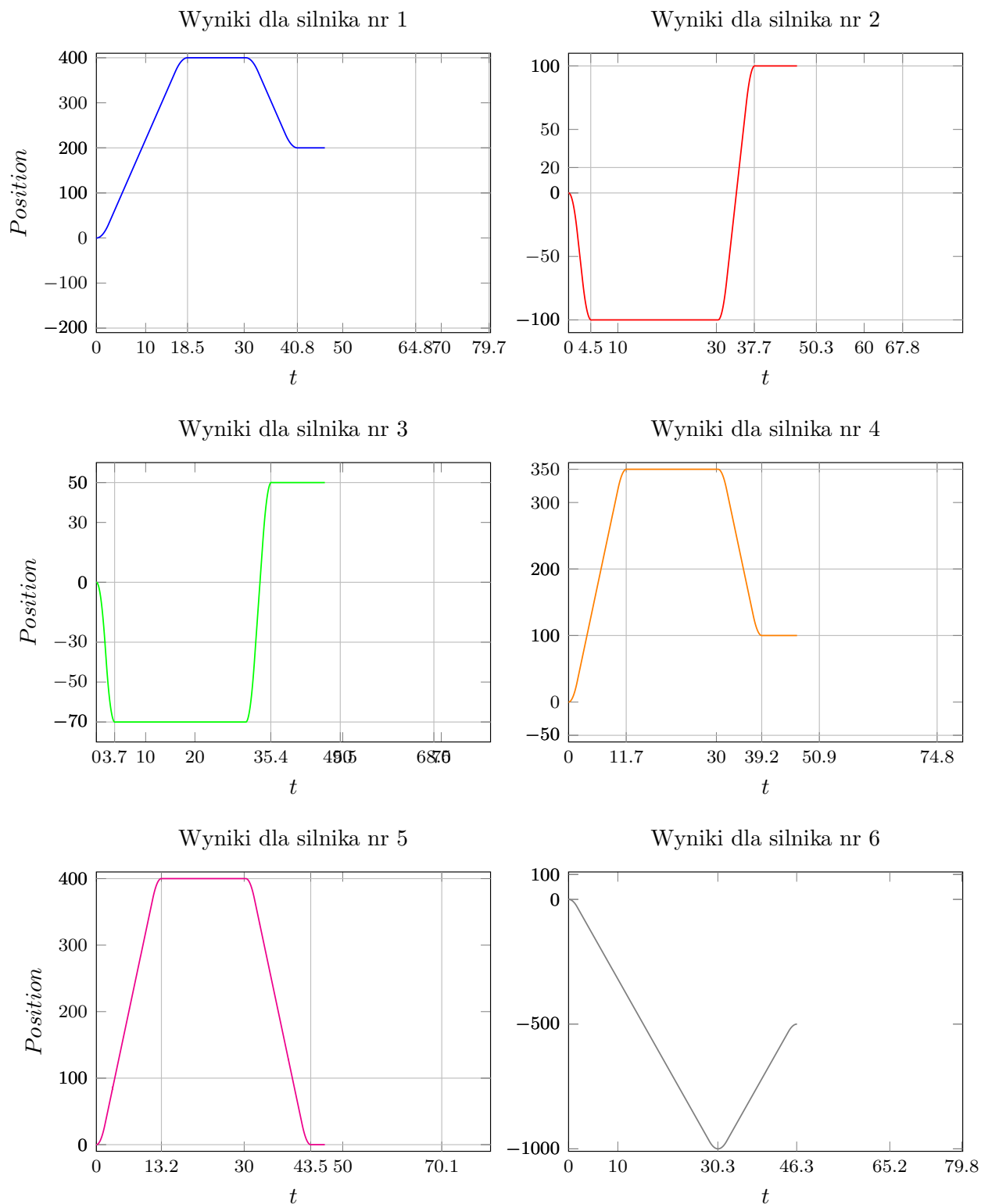
8.2.2 Proste działanie w trybie badawczym

Działanie w trybie badawczym powinno nie pozwolić na ruch silnika dla nastaw z [tablicy 11](#), dla silników 2 i 3.



Rysunek 13: Wyniki testu trajektorii dla trzeciego silnika funkcji `move_along_motor_trajectory_trapezoid_velocity` przy nastawach z [tablicy 11](#). Tryb badawczy.

Rysunek 13 pokazuje, różnice pomiędzy trajektorią wygenerowaną w Matlab'ie, trybem normalnym i badawczym. Jak się spodziewano tryb badawczy przerwał działanie gdy napotkał problem niemożliwości osiągnięcia maksymalnej prędkości. Jest to zgodne z założeniami trybu badawczego. Przerwanie pracy następuje nie przed rozpoczęciem całej trajektorii, gdyż zakłada się, że zebranie przynajmniej początkowych danych może być przydatne. Ze względu na alegoryczny wygląd wiadomości błędu do rysunku 10, pominięto ponowne jego przedstawienie.



Rysunek 14: Wyniki testu trajektorii dla wszystkich silników funkcji `move.along_motor_trajectory_trapezoid_velocity` przy nastawach z tablicy 11. Tryb badawczy.

Na wykresie 14, szczególnie w porównaniu z 12, widać moment w którym generator wykrył

błąd zadanych na silniku 2. Tabela 12 przedstawia wyniki dla oddzielnego badania trajektorii.

silnik	q_i	q_2	q_3	q_4	q_f	v_{max}	a_{max}	wynik symulacji
1	0.0	400.0	200.0	-200.0	100.0	25.0	10	✓
2	0.0	-100.0	100.0	20.0	0.0	35.0	20	X
3	0.0	-70.0	50.0	0.0	-30.0	35.0	20	X
4	0.0	350.0	100.0	200.0	-50.0	35.0	20	✓
5	0.0	400.0	0.0	0.0	100.0	35.0	20	✓
6	0.0	-1000.0	-500.0	100.0	0.0	35.0	20	✓

Tablica 12: Wyniki symulacji najprostszego przypadku, badanie oddzielnych silników. Tryb badawczy.

Literatura

- [1] Dokumentacja pakietu actionlib
<http://wiki.ros.org/actionlib>, Open Source Robotics Foundation, kwiecień 2018.
- [2] Dokumentacja wiadomości trajectory_msgs/JointTrajectory
http://docs.ros.org/api/trajectory_msgs/html/msg/JointTrajectory.html, Open Source Robotics Foundation, maj 2018.
- [3] Dokumentacja wiadomości control_msgs/JointTolerance.msg
http://docs.ros.org/jade/api/control_msgs/html/msg/JointTolerance.html, Open Source Robotics Foundation, luty 2016.
- [4] Dokumentacja wiadomości trajectory_msgs/JointTrajectoryPoint.msg
http://docs.ros.org/api/trajectory_msgs/html/msg/JointTrajectoryPoint.html, Open Source Robotics Foundation, maj 2018.
- [5] The Orocos Component Builder's Manual
<http://www.orocos.org/stable/documentation/rtt/v2.x/doc-xml/orocos-components-manual.html>, Open Robot COntrol Software, 2012.