



# Politechnika Wrocławska



Katedra Cybernetyki i Robotyki

## **Projekt robota usługowego do zastosowań domowych**

Albert Lis

Promotor:

Dr inż. Mateusz Cholewiński

Wrocław, 30 stycznia 2020



# Plan prezentacji

- 1 Wprowadzenie
- 2 Założenia projektowe
- 3 Konstrukcja
- 4 Teoria
- 5 Software
- 6 Kinematyka robota
- 7 Symulacje sterownika kinematycznego



# Co robot powinien posiadać?

Robot powinien:

- ① Posiadać wirujące szczotki myjące podłogi
- ② Posiadać system dozujący wodę z detergentem
- ③ Posiadać system zbierający zużytą wodę z podłogi
- ④ Potrafić rozpoznawać przeszkody
- ⑤ Potrafić rozpoznawać niebezpieczne różnice wysokości podłogi
- ⑥ Posiadać system jazdny pozwalający łatwo osiągać zadane położenia
- ⑦ Posiadać możliwość określania swojej bieżącej pozycji
- ⑧ Sygnalizować zdarzenia nadzwyczajne



# Wybór klasy robota

## Robot klasy 2,0

- ① prostota sterowania
- ② prostota konstrukcji
- ③ obrót w miejscu
- ④ wrażliwość na nierówności

## Sterowanie

- ① Sterowanie TYLKO kinematyczne
- ② Kontroler kinematyczny - algorytm Samsona



# Podzespoły

Mikrokontroler:  
STM32F103C8T6 - Blue Pill



Pomiar odległości od podłogi  
Czujnik IR i komparatorem LM393



## Detekcja kolizji

Bumper wykonany z 2 wyłączników krańcowych.

## Komunikacja z użytkownikiem

Przyciski:

- Start/Stop
- Włącz/Wyłącz

Sygnalizacja zdarzeń:

- Buzzer



## Podzespoły - cd

### Silniki DC z przekładnią



Prędkość – 100 obr/min. W testach praktycznych nawet 200 obr/min

### Enkodery



Enkoder optyczny mocowany do wału silnika. Płytką szczelinową posiada rozdzielcość 20 linii na obrót. Badając oba zboce mam 40 impulsów na obrót, a z uwzględnieniem przekładni wbudowanej w silnik  $48 * 40 = 1920$ .



## Podzespoły - cd

### Zasilanie

- 2 akumulatory Li-Ion 18650 połączone szeregowo. Napięcie nominalne pakietu 7.4 V
- 2 przetwornice Step-Down. Prąd ciągły maksymalny 1.8 A
- Zabezpieczenie moduł BMS

### Układy pompujące wodę

Zraszanie podłogi – pompka od- Zasysanie brudnej wody - pompka śródkowa





# Wykorzystany software i styl pisania kodu

Software:

- ① IDE: PlatformIO w połączeniu z CLion oraz VS Code
- ② Symulacje: Matlab oraz Simulink
- ③ Schematy: Altium Designer
- ④ Rysunki: Inkscape
- ⑤ Graficzna prezentacja danych:
  - Processing (wizualizacja w czasie rzeczywistym)
  - Python + Matplotlib (cały przebieg)

Styl pisania kodu:

- Kod pisany obiektowo. Ostatecznie 18 klas.
- Dzięki obiektowemu podejściu możliwość przeciążenia operatora << (łatwy debug).
- Standard C++11 – przydomek constexpr.
- -Wall, -Wextra.
- Makro F() – ciągi w flash zamiast RAM.

Problemy:

- Framework nie jest najlepiej napisany. Użycie -Wall -Wextra -O3 powoduje dużo warningów.
- Błędy komplikacji – użycie zmiennej statycznej w złym miejscu powoduje 3x wzrost objętości programu i przekroczenie dostępnej pamięci. W jednym przypadku konieczne było użycie git reset.
- CLion nie jest przystosowany do PlatformIO - czasami się psuje.

Sumarycznie jednak zaoszczędzony czas w zupełności wynagrodził te problemy.



# Obliczenia

## Ograniczenia w postaci Pfaffa

$$A(q)\dot{q} = \begin{bmatrix} \sin(\varphi) & -\cos(\varphi) & 0 & 0 & 0 \\ \cos(\varphi) & \sin(\varphi) & -I & -R & 0 \\ \cos(\varphi) & \sin(\varphi) & I & 0 & -R \end{bmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\varphi} \\ \dot{\phi}_1 \\ \dot{\phi}_2 \end{pmatrix}$$

## Bezdryfowy układ sterowania

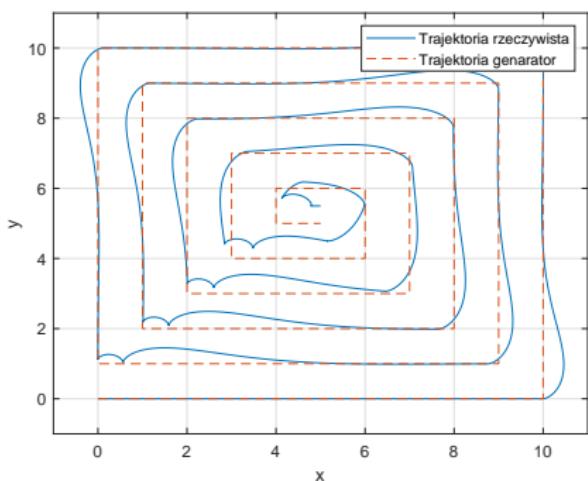
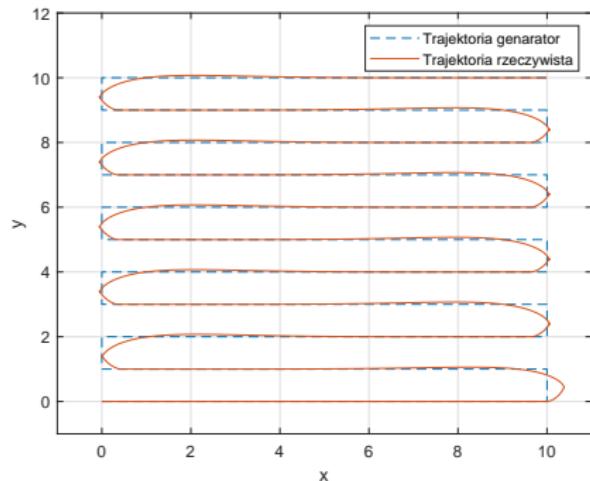
$$\dot{q} = G(q)\eta = \begin{bmatrix} \cos(\varphi) & \cos(\varphi) \\ \sin(\varphi) & \sin(\varphi) \\ \frac{1}{I} & -\frac{1}{I} \\ \frac{2}{R} & 0 \\ 0 & \frac{2}{R} \end{bmatrix} \begin{pmatrix} \eta_1 \\ \eta_2 \end{pmatrix}, \quad \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\varphi} \end{pmatrix} = \begin{bmatrix} \cos(\varphi) & 0 \\ \sin(\varphi) & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix}$$

## Algorytm Samsona

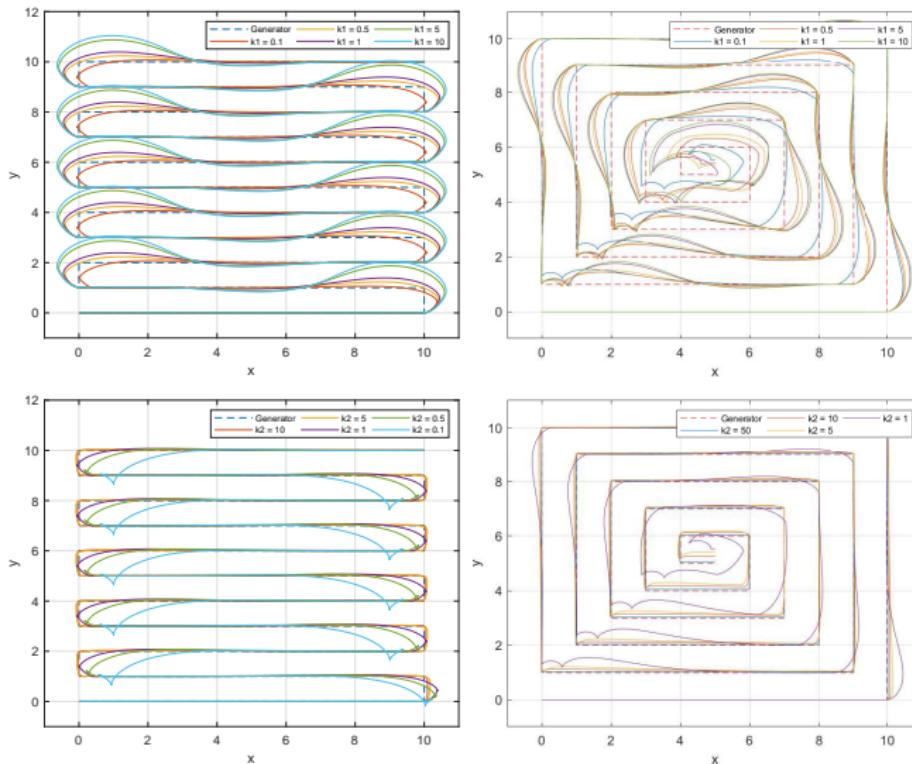
$$\begin{pmatrix} v_r \\ \omega_r \end{pmatrix} = \begin{pmatrix} k_1 x_e + v_d \cos(\varphi_e) \\ \omega_d + k_2 \varphi_e + v_d y_e \frac{\sin(\varphi_e)}{\varphi_e} \end{pmatrix}$$

# Test generowania trajektorii jazdy po pokoju

Wykonany dla parametrów  $k_1$  i  $k_2$  odpowiednio 0.1 oraz 1:

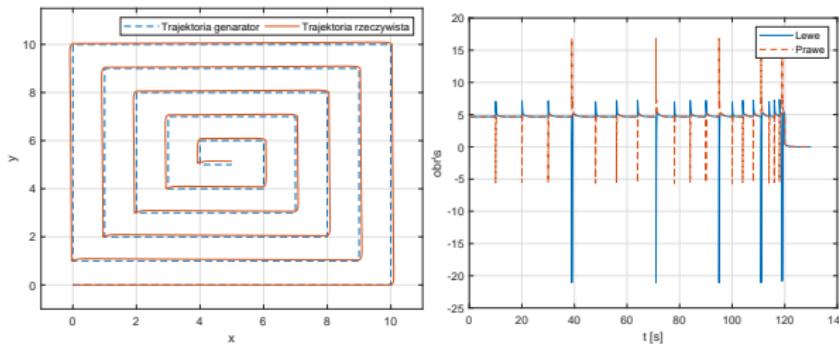
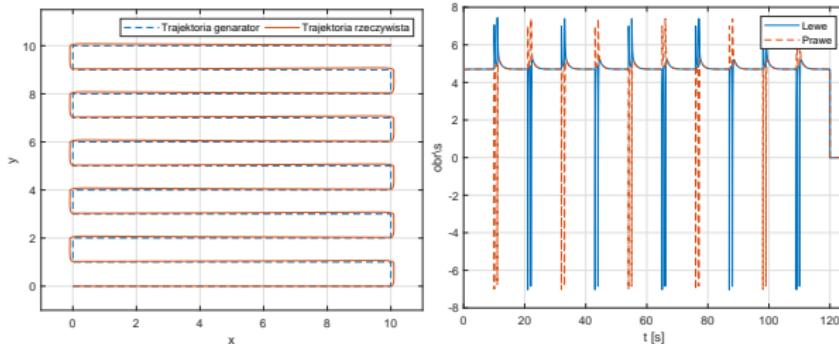


# Porównanie wpływu parametru $k_1$ i $k_2$



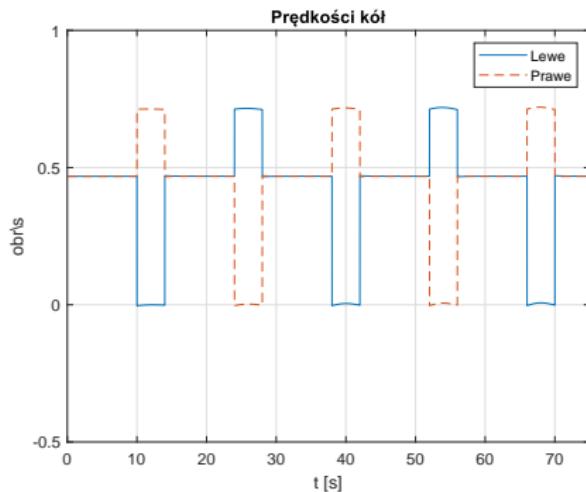
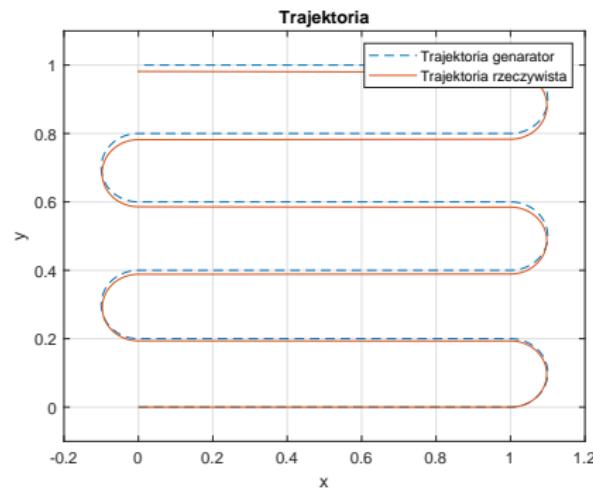
# Czy ta realizacja jest możliwa?

Aby wiedzieć czy robot jest w stanie to zrealizować należy sprawdzić prędkości obrotowe kół. Dla kolejnych symulacji  $k_1 = 1, k_2 = 2$ . Prędkość obrotowa silników to 100 obr/s czyli  $1\frac{2}{3}$  obr/s.



## Jak to rozwiązać?

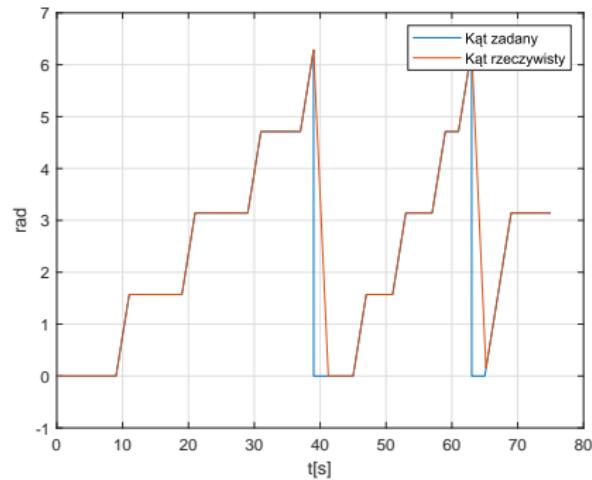
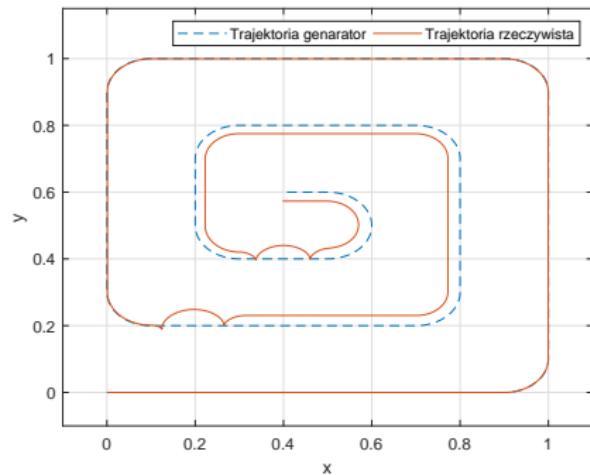
Z poprzednich symulacji wynikało że największy problem generowały gwałtowne zmiany kierunku. Spróbujmy je wygładzić.



Jest dobrze. Prędkości kół nie osiągają wartości granicznych.

# Problem z trajektorią do środka

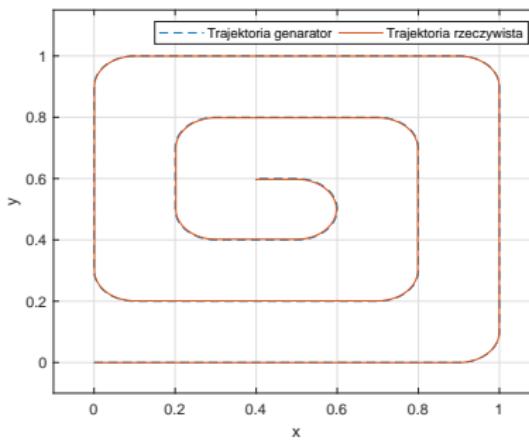
Aby sprawdzić co jest nie tak porównajmy kąt obrotu zadany z rzeczywistym.



Widać, że robot zamiast gładko przejść do pozycji  $2\pi$  odkręca się do pozycji 0.

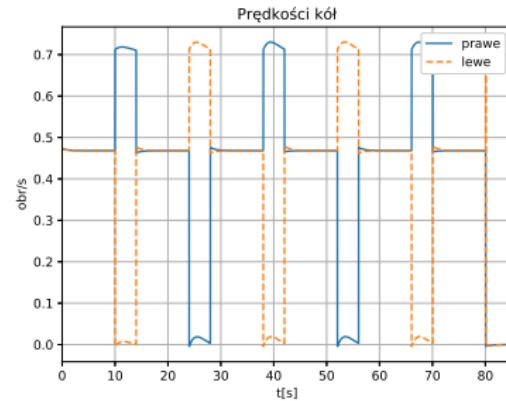
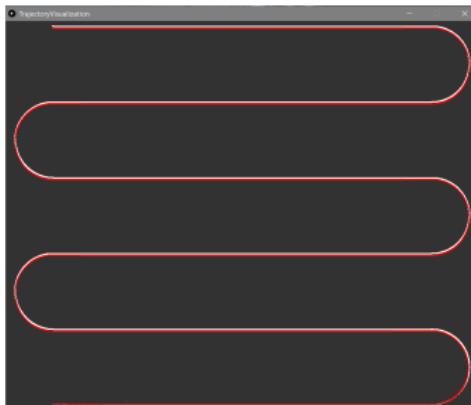
# Rozwiążanie

Choć pozycje  $0$  i  $2\pi$  są tym samym punktem w układzie współrzędnych, potrzebujemy dodatkowo informacji o ilościach obrotów. Najprościej to osiągnąć zmieniając zakres kąta obrotu z  $\langle 0, 2\pi \rangle$  do  $(-\infty, +\infty)$ .



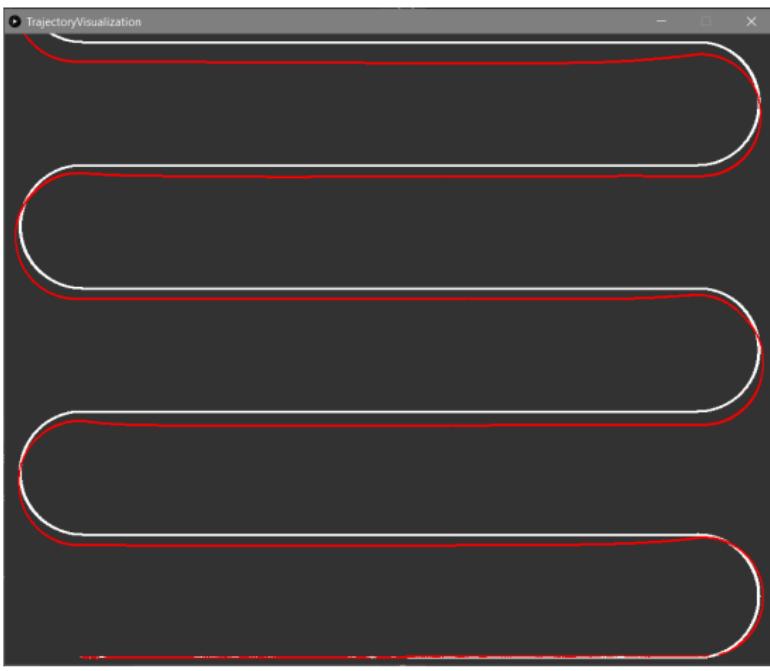
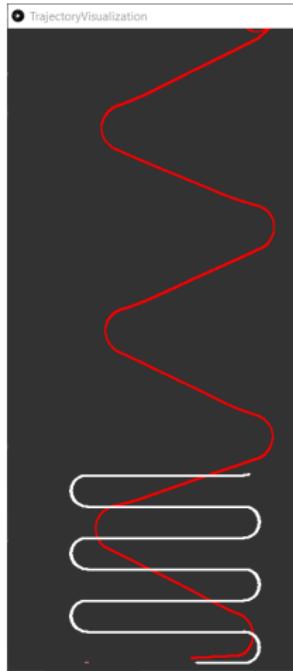
# Symulacja sterownika kinematycznego

Parametry symulacji ustawiono na  $k_1 = 1$ ,  $k_2 = 1$ . Prędkość robota 10 cm/s. Prędkość transmisji 7.2 MBd. Kolor biały reprezentuje trajektorię zadaną, natomiast czerwony rzeczywistą. Dodatkowo dane logowane logowane do pliku .txt. Wyświetlane w skrypcie Python z użyciem biblioteki Matplotlib.

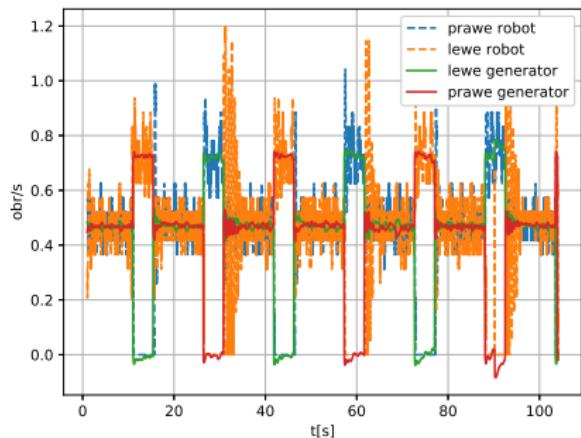
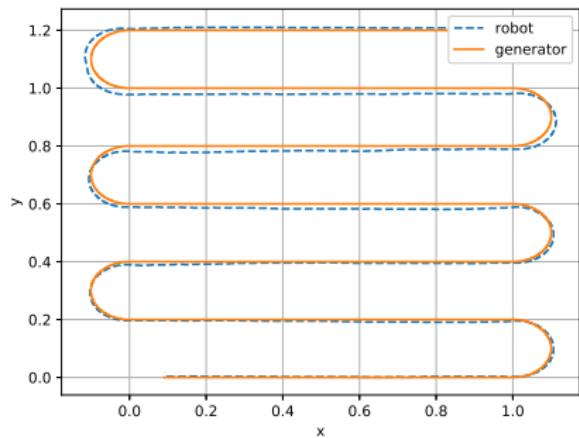




# Testy na rzeczywistym robocie

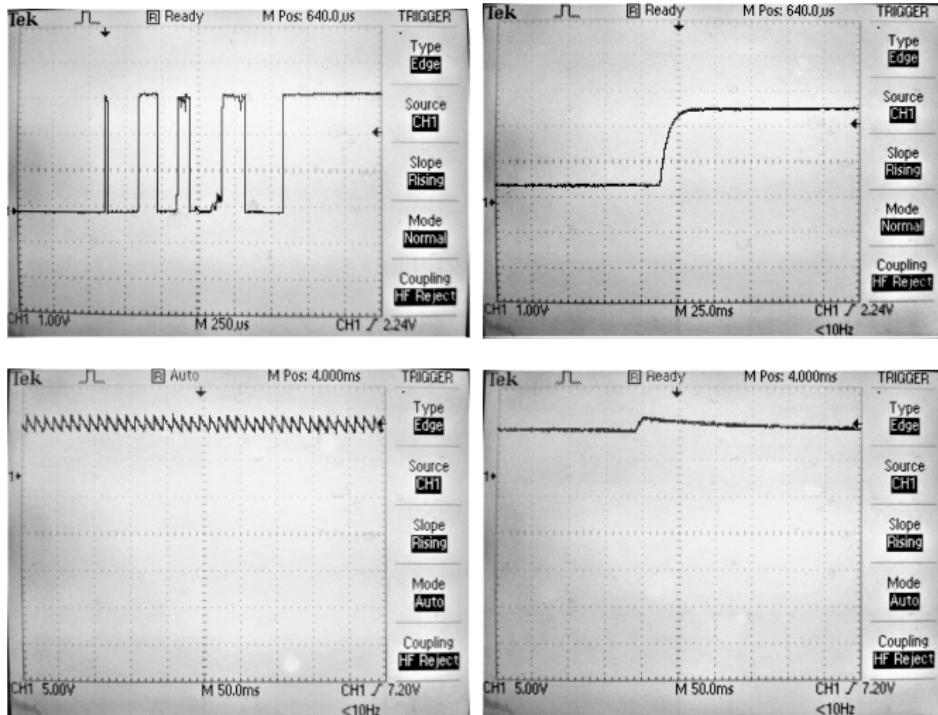


# Ostateczne wyniki

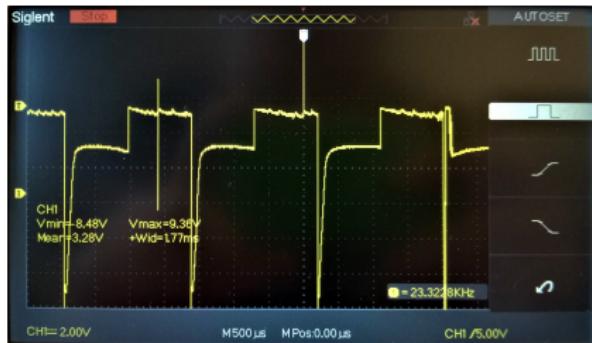


Jeden z silników jest niskiej jakości i stawia nierówny opór. Jednak nie wpływa to znacząco na trajektorię.

# Drgania styków i filtracja zasilania



# Elementy indukcyjne w obwodzie





# Czas wykonywania obliczeń

Porównanie zmiennych dla flagi -O0

Numer testu	Czas trwania obliczeń [ms]		
	int	float	double
1	529,446	3482,773	4332,720
2	529,470	3482,749	4332,743
3	529,467	3482,749	4332,743
4	529,469	3482,751	4332,743
5	529,467	3482,749	4332,744

Czas wykonywania pętli. Typ zmiennych – double. Pętla 100 razy /s — > czas jednej pętli 10 ms.

Czas [ $\mu$ s]			
-O0	-Os	-O3	-Ofast
844	452	450	414

Mimo obiektowego podejścia i "ciężkiego" frameworku wykorzystanie procesora w najgorszym przypadku wynosi od 4.14 % do 8.44 %. Czasy pochodzą z zakrętów gdzie jest dużo cos i sin. Można użyć funkcji constexpr i wyliczyć Look up table.

Rozmiar pamięci RAM mikrokontrolera to 20 480 B, natomiast Flash 65 536 B.

Pamięć	Wykorzystanie pamięci			
	-O0	-Os	-O3	-Ofast
RAM	22.9 % (4696 B)	22.9 % (4680 B)	22.9 % (4680 B)	22.9 % (4680 B)
Flash	84.6 % (55 424 B)	59.0 % (38 688 B)	62.2 % (40 736 B)	62.2 % (40 736 B)



Koniec

Dziękuję za uwagę