Projekt manipulatora antropomorficznego - kod programu

Wygenerowano przez Doxygen 1.8.11

Spis treści

1 Indeks hierarchiczny

1.1 Hierarchia klas

Ta lista dziedziczenia posortowana jest z grubsza, choć nie całkowicie, alfabetycznie:

2 Indeks klas

2.1 Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

3 Indeks plików

3.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

4 Dokumentacja klas

4.1 Dokumentacja klasy Czujnik

Klasa abstrakcyjna Czujnik. Klasa zawierająca wirtualne metody które przy wywoływaniu nadpisywane są przez metody klasy dziedziczącej.

Diagram dziedziczenia dla Czujnik

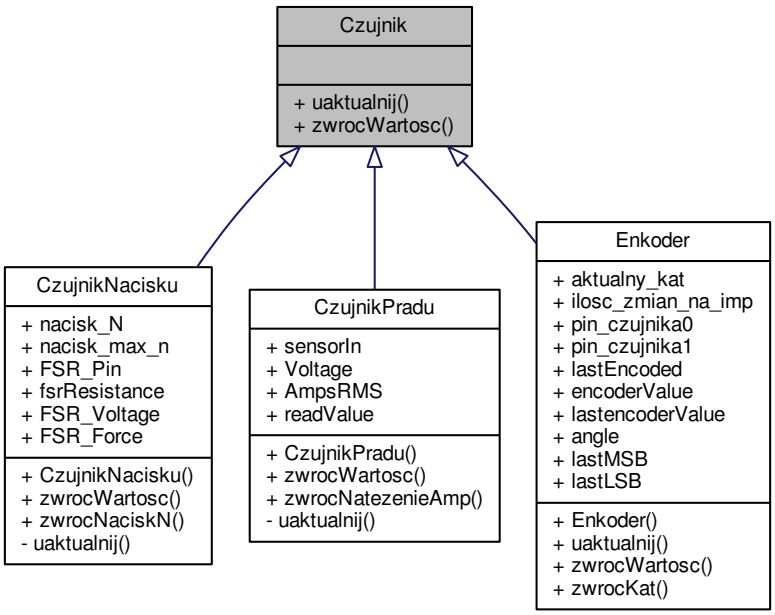
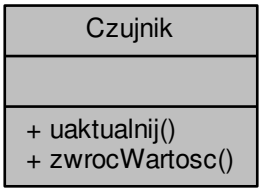


Diagram współpracy dla Czujnik:



Metody publiczne

- virtual int uaktualnij ()=0

  metoda bazowa uaktualnij()
- virtual float zwrocWartosc ()=0

  metoda zwracająca dowolną wartość

4.1.1 Opis szczegółowy

Klasa abstrakcyjna Czujnik. Klasa zawierająca wirtualne metody które przy wywoływaniu nadpisywane są przez metody klasy dziedziczącej.

Definicja w linii 18 pliku Czujnik.cpp.

4.1.2 Dokumentacja funkcji składowych

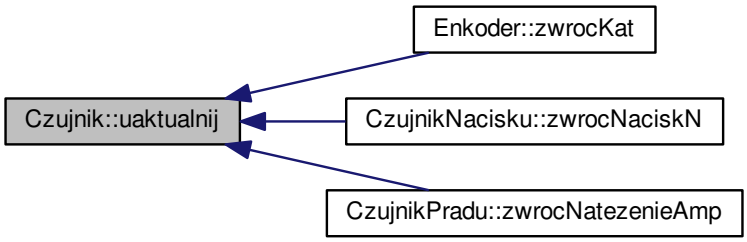
4.1.2.1 virtual int Czujnik::uaktualnij ( ) [pure virtual]

metoda bazowa uaktualnij()

Implementowany w Enkoder, CzujnikPradu i CzujnikNacisku.

Odwołania w Enkoder::zwrocKat(), CzujnikNacisku::zwrocNaciskN() i CzujnikPradu::zwrocNatezenieAmp().

Oto graf wywoływań tej funkcji:



4.1.2.2 virtual float Czujnik::zwrocWartosc ( ) [pure virtual]

metoda zwracająca dowolną wartość

Implementowany w Enkoder, CzujnikPradu i CzujnikNacisku.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Czujnik.cpp

4.2 Dokumentacja klasy CzujnikNacisku

Klasa ta zawiera metody liczące siłę nacisku i zwracające wartość tego nacisku do programu głównego.

Diagram dziedziczenia dla CzujnikNacisku

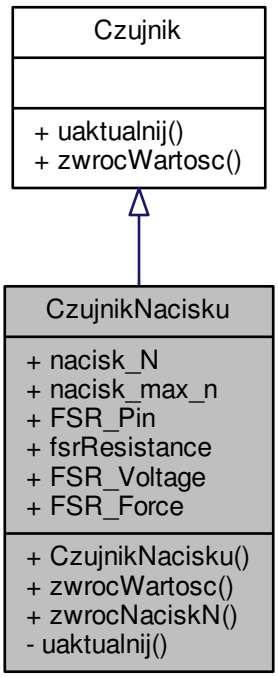
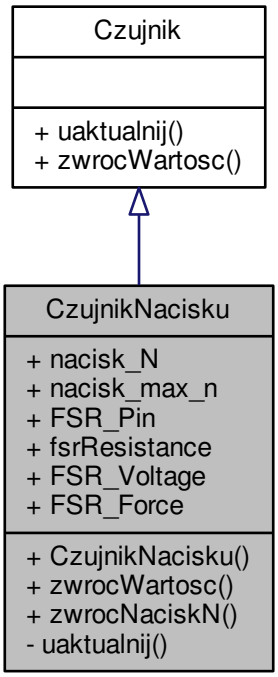


Diagram współpracy dla CzujnikNacisku:



Metody publiczne

- CzujnikNacisku (int pin_czuj)

  Konstruktor - po otrzymaniu paramtru pin_czuj przypisuję go do zmiennej odpowiadającej pinu z którego odczytywana jest wartosc nacisku i ustawia ten pin jako wejściowy.
- float zwrocWartosc ()

  metoda zwracająca wartość nacisku. Nadpisuje metodę z klasy bazowej.
- float zwrocNaciskN ()

  metoda zwracająca wartość nacisku w Newtonach.

Atrybuty publiczne

- float nacisk_N
- float nacisk_max_n
- int FSR_Pin
- float fsrResistance
- float FSR_Voltage
- float FSR_Force

Metody prywatne

- int uaktualnij ()

  Metoda uaktualniająca zmienne wewnętrzne obiektu tej klasy. Liczony jest tu siła nacisku na czujnik nacisku w Newtonach.

4.2.1 Opis szczegółowy

Klasa ta zawiera metody liczące siłę nacisku i zwracające wartość tego nacisku do programu głównego.

Definicja w linii 28 pliku Czujnik.cpp.

4.2.2 Dokumentacja konstruktora i destruktora

4.2.2.1 CzujnikNacisku::CzujnikNacisku (int *pin_czuj*) [inline]

Konstruktor - po otrzymaniu paramtru pin_czuj przypisuję go do zmiennej odpowiadającej pinu z którego odczytywana jest wartosc nacisku i ustawia ten pin jako wejściowy.

Parametry

pin_czuj	Wyjście cyfrowe odpowiadające za komunikację czujnika z płytką Arduino Due.Konstruktor - po otrzymaniu paramtru pin_czuj przypisuję go do zmiennej odpowiadającej pinu z którego odczytywana jest wartosc nacisku i ustawia ten pin jako wejściowy.

Definicja w linii 69 pliku Czujnik.cpp.

4.2.3 Dokumentacja funkcji składowych

4.2.3.1 int CzujnikNacisku::uaktualnij ( ) [inline], [private], [virtual]

Metoda uaktualniająca zmienne wewnętrzne obiektu tej klasy. Liczony jest tu siła nacisku na czujnik nacisku w Newtonach.

Implementuje Czujnik.

Definicja w linii 31 pliku Czujnik.cpp.

Odwołuje się do DEBUG_MODE.

4.2.3.2 float CzujnikNacisku::zwrocNaciskN ( ) [inline]

metoda zwracająca wartość nacisku w Newtonach.

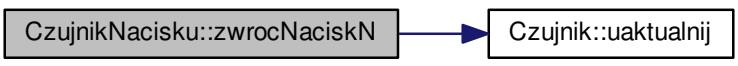
Zwraca

Wartość siły w Newtonach.

Definicja w linii 84 pliku Czujnik.cpp.

Odwołuje się do Czujnik::uaktualnij().

Oto graf wywołań dla tej funkcji:



4.2.3.3 float CzujnikNacisku::zwrocWartosc ( ) [inline], [virtual]

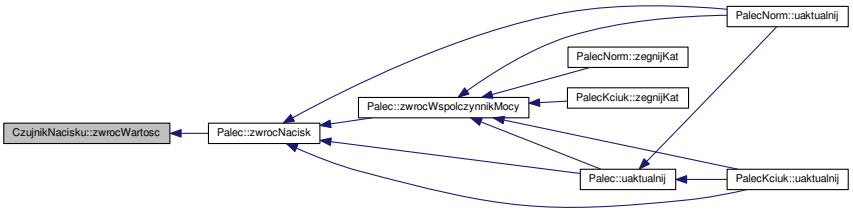
metoda zwracająca wartość nacisku. Nadpisuje metodę z klasy bazowej.

Implementuje Czujnik.

Definicja w linii 76 pliku Czujnik.cpp.

Odwołania w Palec::zwrocNacisk().

Oto graf wywoływań tej funkcji:



4.2.4 Dokumentacja atrybutów składowych

4.2.4.1 float CzujnikNacisku::FSR_Force

Definicja w linii 64 pliku Czujnik.cpp.

4.2.4.2 int CzujnikNacisku::FSR_Pin

Definicja w linii 61 pliku Czujnik.cpp.

4.2.4.3 float CzujnikNacisku::FSR_Voltage

Definicja w linii 63 pliku Czujnik.cpp.

4.2.4.4 float CzujnikNacisku::fsrResistance

Definicja w linii 62 pliku Czujnik.cpp.

4.2.4.5 float CzujnikNacisku::nacisk_max_n

Definicja w linii 60 pliku Czujnik.cpp.

4.2.4.6 float CzujnikNacisku::nacisk_N

Definicja w linii 59 pliku Czujnik.cpp.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Czujnik.cpp

4.3 Dokumentacja klasy CzujnikPradu

Klasa CzujnikPradu dziedziczy metody uaktualnij() i zwrocWartosc() z abstrakcyjnej klasy Czujnik.

Diagram dziedziczenia dla CzujnikPradu

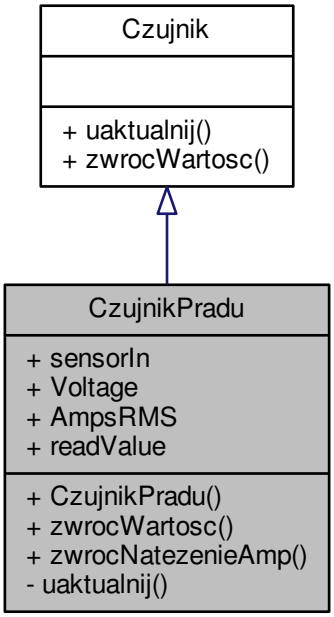
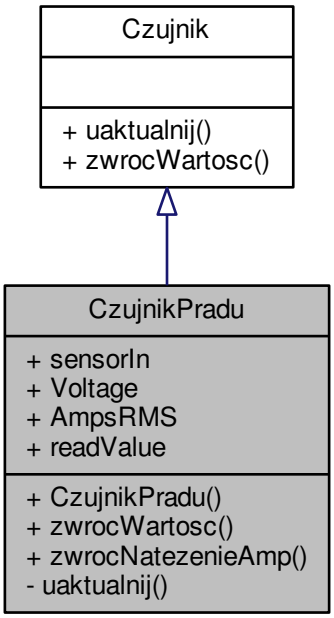


Diagram współpracy dla CzujnikPradu:



Metody publiczne

- CzujnikPradu (int pin_czujnika)

  Konstruktor zawirający deklarację zmiennych i ustawia otrzymany parametr jako pin wejściowy.
- float zwrocWartosc ()

  metoda zwracająca dowolną wartość. Nadpisuje metodę z klasy bazowej.
- float zwrocNatezenieAmp ()

  metoda zwracająca wartość natężenia prądu

Atrybuty publiczne

- int sensorIn = A7
- double Voltage = 0
- double AmpsRMS = 0
- float readValue = 0

Metody prywatne

- int uaktualnij ()

  Metoda przelicza liczbę odczytaną z przetwornika A/C na wartość pobieranego prądu w [A].

4.3.1 Opis szczegółowy

Klasa CzujnikPradu dziedziczy metody uaktualnij() i zwrocWartosc() z abstrakcyjnej klasy Czujnik.

Definicja w linii 98 pliku Czujnik.cpp.

4.3.2 Dokumentacja konstruktora i destruktora

4.3.2.1 CzujnikPradu::CzujnikPradu (int *pin_czujnika*) [inline]

Konstruktor zawirający deklarację zmiennych i ustawia otrzymany parametr jako pin wejściowy.

Parametry

pin_czujnika	Pin odpowiedzialny za komunikację z Arduino Due.

Definicja w linii 126 pliku Czujnik.cpp.

4.3.3 Dokumentacja funkcji składowych

4.3.3.1 int CzujnikPradu::uaktualnij ( ) [inline], [private], [virtual]

Metoda przelicza liczbę odczytaną z przetwornika A/C na wartość pobieranego prądu w [A].

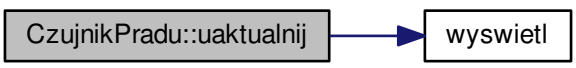
Metoda uaktualniająca zmienne wewnętrzne obiektu tej klasy. Liczony jest tu prąd w Amperach.

Implementuje Czujnik.

Definicja w linii 104 pliku Czujnik.cpp.

Odwołuje się do DEBUG_MODE i wyswietl().

Oto graf wywołań dla tej funkcji:



4.3.3.2 float CzujnikPradu::zwrocNatezenieAmp ( ) [inline]

metoda zwracająca wartość natężenia prądu

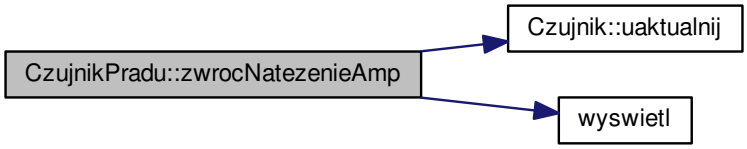
Zwraca

Prąd w Amperach.

Definicja w linii 141 pliku Czujnik.cpp.

Odwołuje się do DEBUG_MODE, Czujnik::uaktualnij() i wyswietl().

Oto graf wywołań dla tej funkcji:



4.3.3.3 float CzujnikPradu::zwrocWartosc ( ) [inline], [virtual]

metoda zwracająca dowolną wartość. Nadpisuje metodę z klasy bazowej.

Zwraca

Prąd w Amperach.

Implementuje Czujnik.

Definicja w linii 134 pliku Czujnik.cpp.

4.3.4 Dokumentacja atrybutów składowych

4.3.4.1 double CzujnikPradu::AmpsRMS = 0

Definicja w linii 120 pliku Czujnik.cpp.

4.3.4.2 float CzujnikPradu::readValue = 0

Definicja w linii 121 pliku Czujnik.cpp.

4.3.4.3 int CzujnikPradu::sensorIn = A7

Definicja w linii 117 pliku Czujnik.cpp.

4.3.4.4 double CzujnikPradu::Voltage = 0

Definicja w linii 119 pliku Czujnik.cpp.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Czujnik.cpp

4.4 Dokumentacja klasy Enkoder

Klasa Enkoder definiuje zachowanie Enkodera inkrementalnego, pozwala na podstawie ilości impulsów i sekwencji zmian na impuls określić kąt obrotu.

Diagram dziedziczenia dla Enkoder

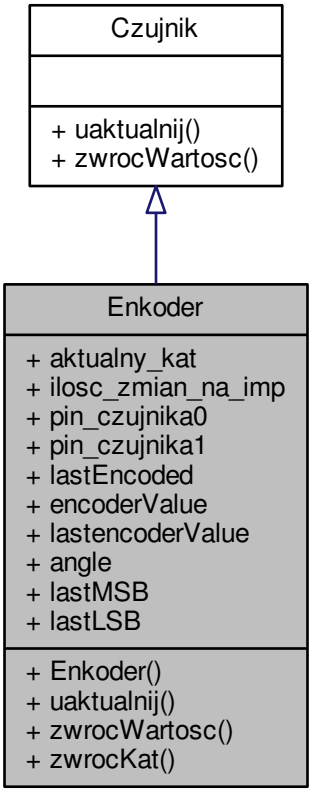
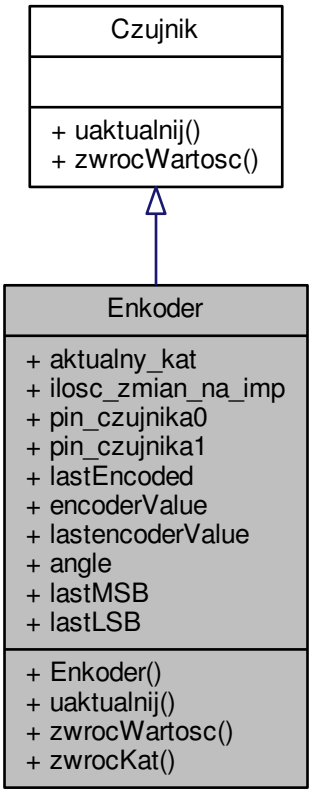


Diagram współpracy dla Enkoder:



Metody publiczne

- Enkoder (int pin0, int pin1, int il_zm_na_imp)

  Konstruktor zawirający deklarację zmiennych i ustawia otrzymane parametry jako pin wejściowy.
- int uaktualnij ()

  Metoda ta zlicza impulsy i rejestruje zmiany wartości na podstawie czego liczy wartość kąta w stopniach o jaki obrócił się wał silnika.
- float zwrocWartosc ()

  Metoda wywołuję metodę zwrocKat()
- float zwrocKat ()

  Metoda zwracająca wartość kąta w stopniach o jak obrócił się wał silnika.

Atrybuty publiczne

- float aktualny_kat
- int ilosc_zmian_na_imp
- int pin_czujnika0
- int pin_czujnika1
- volatile int lastEncoded = 0

- volatile long encoderValue = 0
- long lastencoderValue = 0
- volatile float angle = 0
- int lastMSB = 0
- int lastLSB = 0

4.4.1 Opis szczegółowy

Klasa Enkoder definiuje zachowanie Enkodera inkrementalnego, pozwala na podstawie ilości impulsów i sekwencji zmian na impuls określić kąt obrotu.

Definicja w linii 156 pliku Czujnik.cpp.

4.4.2 Dokumentacja konstruktora i destruktora

4.4.2.1 Enkoder::Enkoder (int *pin0*, int *pin1*, int *il_zm_na_imp*) [inline]

Konstruktor zawirający deklarację zmiennych i ustawia otrzymane parametry jako pin wejściowy.

Konstruktor.

Parametry

pin0	Pin odpowiedzialny za komunikację z Arduino Due podłączony do kanału A enkodera.
pin1	Pin odpowiedzialny za komunikację z Arduino Due podłączony do kanału B enkodera.
il_zm_na_imp	Zmienna określająca ilość zmian na jeden impuls.

Definicja w linii 177 pliku Czujnik.cpp.

4.4.3 Dokumentacja funkcji składowych

4.4.3.1 int Enkoder::uaktualnij ( ) [inline], [virtual]

Metoda ta zlicza impulsy i rejestruje zmiany wartości na podstawie czego liczy wartość kąta w stopniach o jaki obrócił się wał silnika.

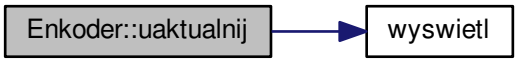
Wartość kąta zawiera się w przedziale (-180, 180).

Implementuje Czujnik.

Definicja w linii 196 pliku Czujnik.cpp.

Odwołuje się do DEBUG_MODE i wyswietl().

Oto graf wywołań dla tej funkcji:



4.4.3.2 float Enkoder::zwrocKat ( ) [inline]

Metoda zwracająca wartość kąta w stopniach o jak obrócił się wał silnika.

Metoda zwracająca wartość kąta obrotu wału silnika.

Zwraca

Kąt w stopniach.

Definicja w linii 242 pliku Czujnik.cpp.

Odwołuje się do Czujnik::uaktualnij().

Oto graf wywołań dla tej funkcji:

4.4.3.3 float Enkoder::zwrocWartosc ( ) [inline], [virtual]

Metoda wywołuję metodę zwrocKat()

Metoda dziedziczonej klasy Czujnik wywołująca metodę zwrocKat(). Nadpisuje metodę z klasy bazowej.

Zwraca

Kąt w stopniach.

Implementuje Czujnik.

Definicja w linii 234 pliku Czujnik.cpp.

4.4.4 Dokumentacja atrybutów składowych

4.4.4.1 float Enkoder::aktualny_kat

Definicja w linii 158 pliku Czujnik.cpp.

4.4.4.2 volatile float Enkoder::angle = 0

Definicja w linii 165 pliku Czujnik.cpp.

4.4.4.3 volatile long Enkoder::encoderValue = 0

Definicja w linii 163 pliku Czujnik.cpp.

4.4.4.4 int Enkoder::ilosc_zmian_na_imp

Definicja w linii 159 pliku Czujnik.cpp.

4.4.4.5 volatile int Enkoder::lastEncoded = 0

Definicja w linii 162 pliku Czujnik.cpp.

4.4.4.6 long Enkoder::lastencoderValue = 0

Definicja w linii 164 pliku Czujnik.cpp.

4.4.4.7 int Enkoder::lastLSB = 0

Definicja w linii 168 pliku Czujnik.cpp.

4.4.4.8 int Enkoder::lastMSB = 0

Definicja w linii 167 pliku Czujnik.cpp.

4.4.4.9 int Enkoder::pin_czujnika0

Definicja w linii 160 pliku Czujnik.cpp.

4.4.4.10 int Enkoder::pin_czujnika1

Definicja w linii 161 pliku Czujnik.cpp.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Czujnik.cpp

4.5 Dokumentacja klasy Naped

Klasa Naped zawiera metody wirtualne które nadpisywane są przez klasy dziedziczące.

Diagram dziedziczenia dla Naped

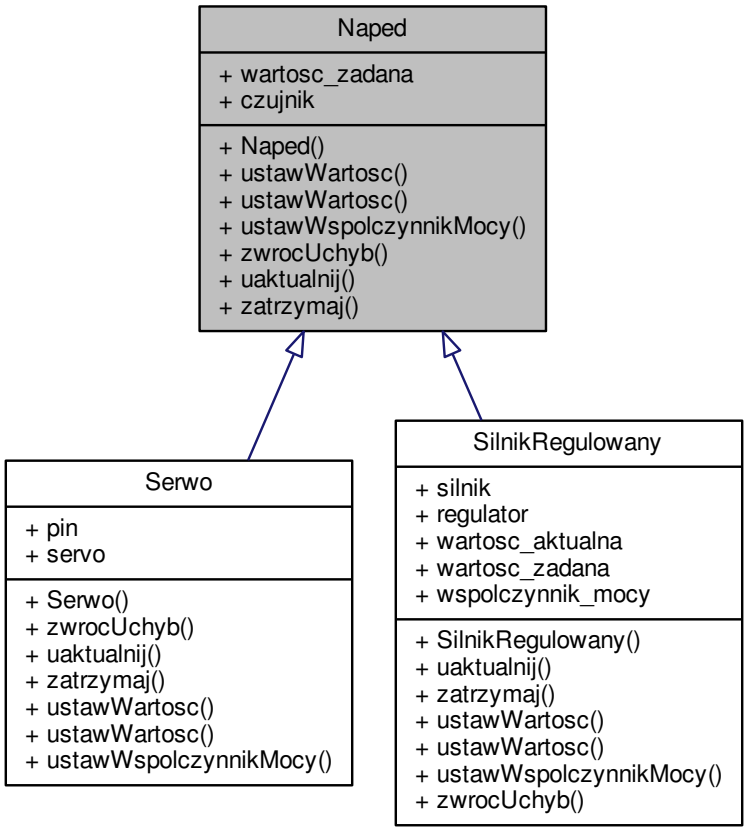
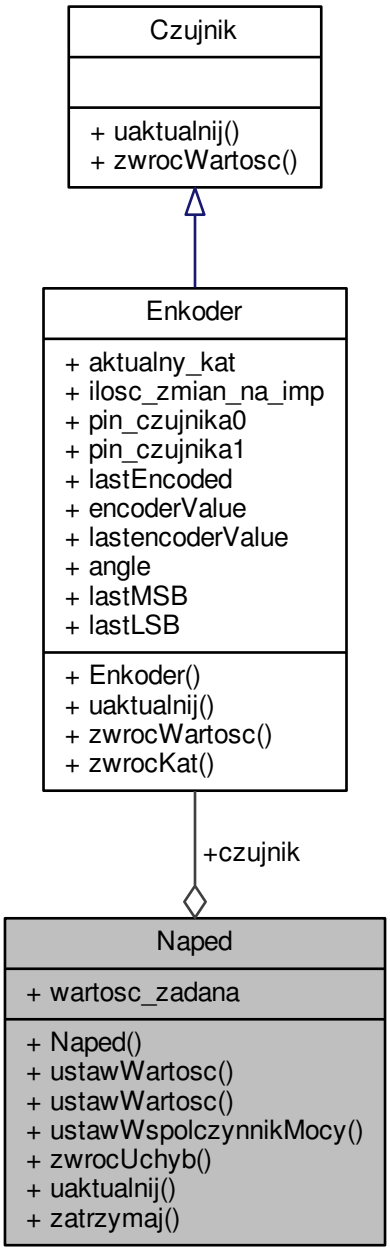


Diagram współpracy dla Naped:



Metody publiczne

- Naped ()
- virtual int ustawWartosc (float wartosc)=0
- virtual int ustawWartosc (float wartosc, float wspolczynnik_mocy)=0
- virtual int ustawWspolczynnikMocy (float wspol_mocy)=0
- virtual float zwrocUchyb ()=0
- virtual int uaktualnij ()=0
- virtual int zatrzymaj ()=0

Atrybuty publiczne

- float wartosc_zadana = 0.0
- Enkoder ∗ czujnik

4.5.1 Opis szczegółowy

Klasa Naped zawiera metody wirtualne które nadpisywane są przez klasy dziedziczące.

Definicja w linii 25 pliku Naped.cpp.

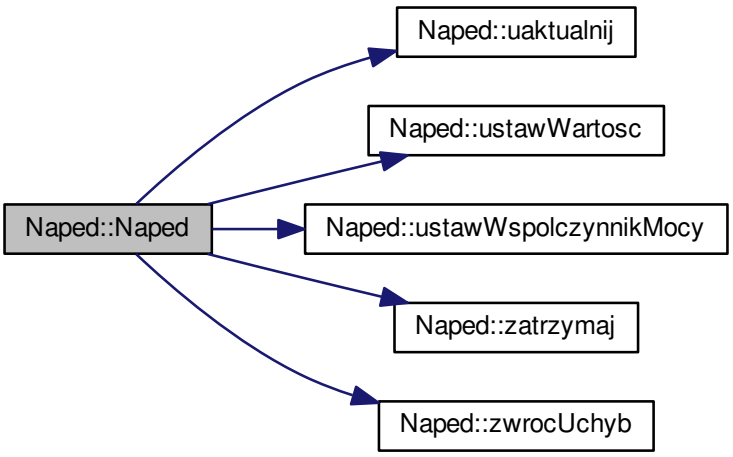
4.5.2 Dokumentacja konstruktora i destruktora

4.5.2.1 Naped::Naped ( ) [inline]

Definicja w linii 30 pliku Naped.cpp.

Odwołuje się do uaktualnij(), ustawWartosc(), ustawWspolczynnikMocy(), zatrzymaj() i zwrocUchyb().

Oto graf wywołań dla tej funkcji:



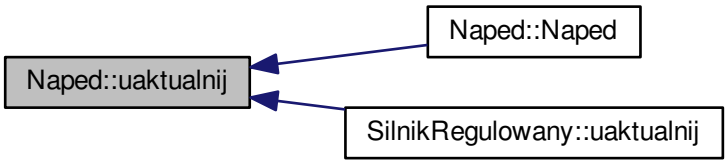
4.5.3 Dokumentacja funkcji składowych

4.5.3.1 virtual int Naped::uaktualnij ( ) [pure virtual]

Implementowany w SilnikRegulowany i Serwo.

Odwołania w Naped() i SilnikRegulowany::uaktualnij().

Oto graf wywoływań tej funkcji:

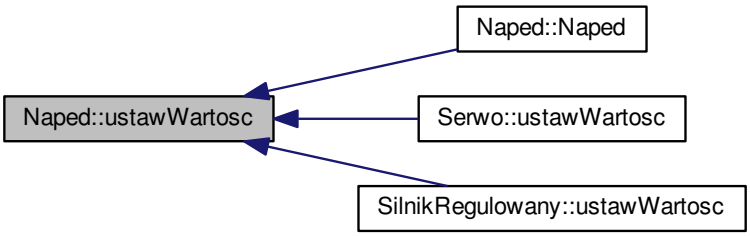


4.5.3.2 virtual int Naped::ustawWartosc (float *wartosc*) [pure virtual]

Implementowany w SilnikRegulowany i Serwo.

Odwołania w Naped(), Serwo::ustawWartosc() i SilnikRegulowany::ustawWartosc().

Oto graf wywoływań tej funkcji:



4.5.3.3 virtual int Naped::ustawWartosc (float *wartosc,* float *wspolczynnik_mocy*) [pure virtual]

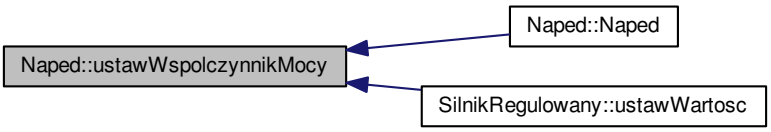
Implementowany w SilnikRegulowany i Serwo.

4.5.3.4 virtual int Naped::ustawWspolczynnikMocy (float *wspol_mocy*) [pure virtual]

Implementowany w SilnikRegulowany i Serwo.

Odwołania w Naped() i SilnikRegulowany::ustawWartosc().

Oto graf wywoływań tej funkcji:

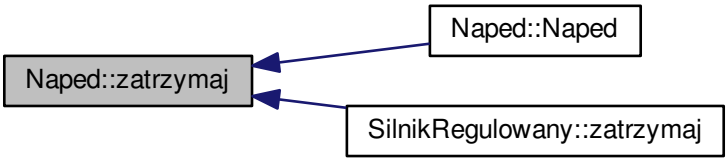


4.5.3.5 virtual int Naped::zatrzymaj ( ) [pure virtual]

Implementowany w SilnikRegulowany i Serwo.

Odwołania w Naped() i SilnikRegulowany::zatrzymaj().

Oto graf wywoływań tej funkcji:

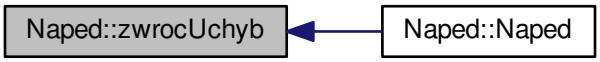


4.5.3.6 virtual float Naped::zwrocUchyb ( ) [pure virtual]

Implementowany w SilnikRegulowany i Serwo.

Odwołania w Naped().

Oto graf wywoływań tej funkcji:



4.5.4 Dokumentacja atrybutów składowych

4.5.4.1 Enkoder* Naped::czujnik

Definicja w linii 28 pliku Naped.cpp.

Odwołania w SilnikRegulowany::SilnikRegulowany() i SilnikRegulowany::uaktualnij().

4.5.4.2 float Naped::wartosc_zadana = 0.0

Definicja w linii 27 pliku Naped.cpp.

Odwołania w Serwo::ustawWartosc().

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Naped.cpp

4.6 Dokumentacja klasy Palec

Klasa Palec zawiera klasy dziedziczące PalecNorm i PalecKciuk. Klasa zaiwera metody wirtualne które nadpisywane są przez klasy dziedziczące.

Diagram dziedziczenia dla Palec

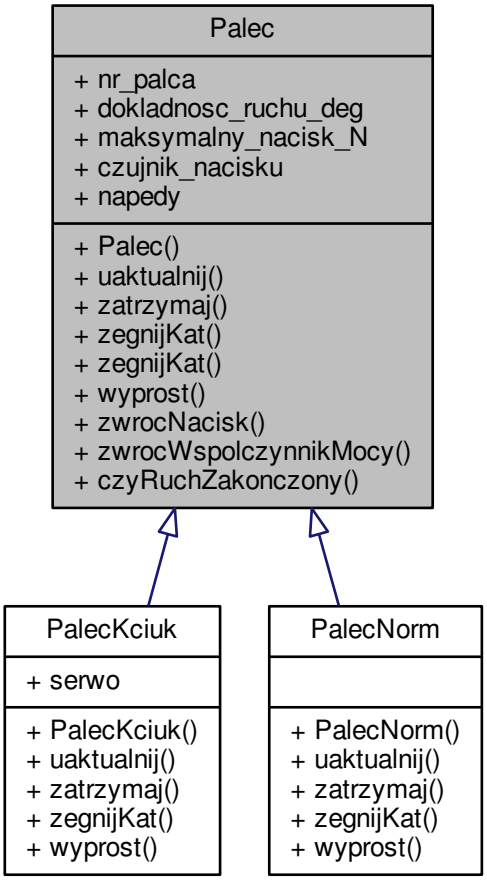
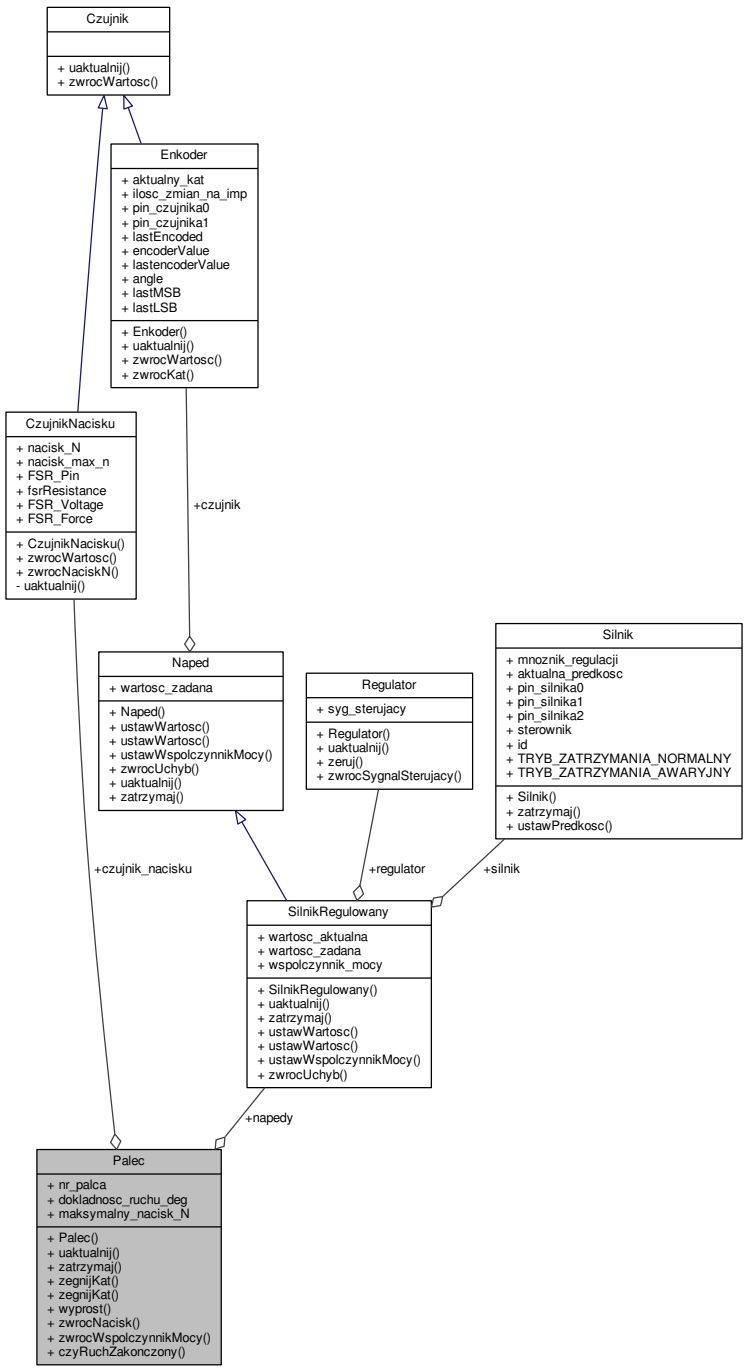


Diagram współpracy dla Palec:



Metody publiczne

- Palec ()
- virtual int uaktualnij ()

  Metoda uaktualniająca zmienne wewnętrzne obiektu tej klasy.
- virtual int zatrzymaj ()=0
- virtual int zegnijKat (float c, float a, float b, float maxnacisk)

Metoda wirtualna nadpisywana przez metodą klasy dziedziczącej.

- virtual int zegnijKat (float a, float b, float maxnacisk)
- virtual int wyprost ()=0

  mwtoda prostująca palec
- float zwrocNacisk ()

  Metoda zwracająca wartość nacisku na czujnik nacisku w [N].
- float zwrocWspolczynnikMocy ()

  Metoda zwracająca wartość współczynnika mocy dla wartości prędkości silnika. Mspółczynnik mocy obliczany jest na podstawie siły nacisku na pole czujnika nacisku.
- bool czyRuchZakonczony ()

  Metoda sprawdzająca czy ruch palca (silnikók w palcu)został zakończony.

Atrybuty publiczne

- char nr_palca
- float dokladnosc_ruchu_deg = 15
- float maksymalny_nacisk_N = 100
- CzujnikNacisku ∗ czujnik_nacisku
- SilnikRegulowany ∗ napedy [2]

4.6.1 Opis szczegółowy

Klasa Palec zawiera klasy dziedziczące PalecNorm i PalecKciuk. Klasa zaiwera metody wirtualne które nadpisywane są przez klasy dziedziczące.

Definicja w linii 22 pliku Palec.cpp.

4.6.2 Dokumentacja konstruktora i destruktora

4.6.2.1 Palec::Palec ( ) [inline]

Definicja w linii 33 pliku Palec.cpp.

4.6.3 Dokumentacja funkcji składowych

4.6.3.1 bool Palec::czyRuchZakonczony ( ) [inline]

Metoda sprawdzająca czy ruch palca (silnikók w palcu)został zakończony.

Definicja w linii 89 pliku Palec.cpp.

4.6.3.2 virtual int Palec::uaktualnij ( ) [inline], [virtual]

Metoda uaktualniająca zmienne wewnętrzne obiektu tej klasy.

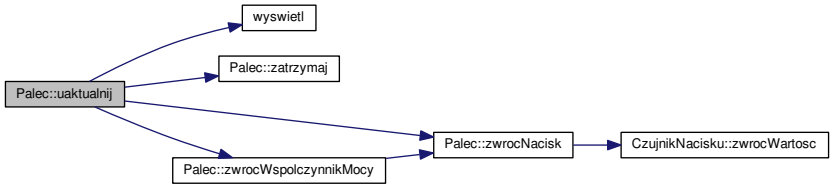
Reimplementowana w PalecKciuk i PalecNorm.

Definicja w linii 37 pliku Palec.cpp.

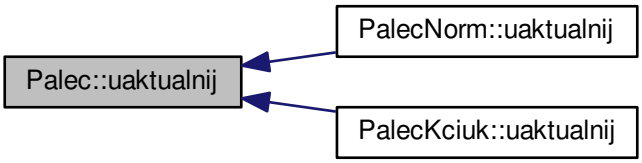
Odwołuje się do wyswietl(), zatrzymaj(), zwrocNacisk() i zwrocWspolczynnikMocy().

Odwołania w PalecNorm::uaktualnij() i PalecKciuk::uaktualnij().

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:



4.6.3.3 virtual int Palec::wyprost ( ) [pure virtual]

mwtoda prostująca palec

Implementowany w PalecKciuk i PalecNorm.

Odwołania w zegnijKat().

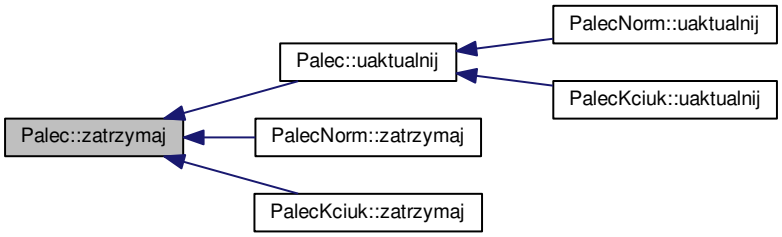
Oto graf wywoływań tej funkcji:

4.6.3.4 virtual int Palec::zatrzymaj ( ) [pure virtual]

Implementowany w PalecKciuk i PalecNorm.

Odwołania w uaktualnij(), PalecNorm::zatrzymaj() i PalecKciuk::zatrzymaj().

Oto graf wywoływań tej funkcji:



4.6.3.5 virtual int Palec::zegnijKat (float *c*, float *a*, float *b*, float *maxnacisk*) [inline], [virtual]

Metoda wirtualna nadpisywana przez metodą klasy dziedziczącej.

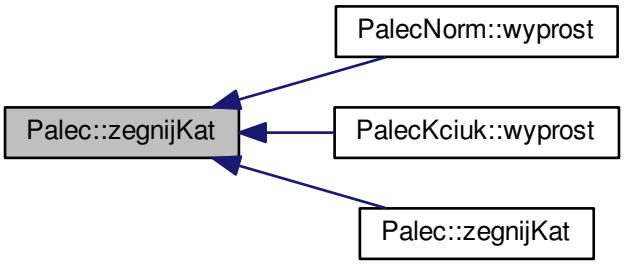
Reimplementowana w PalecKciuk.

Definicja w linii 54 pliku Palec.cpp.

Odwołuje się do DEBUG_MODE.

Odwołania w PalecNorm::wyprost(), PalecKciuk::wyprost() i zegnijKat().

Oto graf wywoływań tej funkcji:



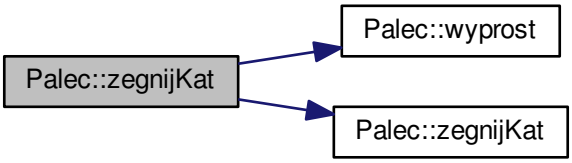
4.6.3.6 virtual int Palec::zegnijKat (float *a*, float *b*, float *maxnacisk*) [inline], [virtual]

Reimplementowana w PalecNorm.

Definicja w linii 61 pliku Palec.cpp.

Odwołuje się do DEBUG_MODE, wyprost() i zegnijKat().

Oto graf wywołań dla tej funkcji:



4.6.3.7 float Palec::zwrocNacisk ( ) [inline]

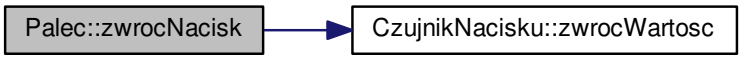
Metoda zwracająca wartość nacisku na czujnik nacisku w [N].

Definicja w linii 70 pliku Palec.cpp.

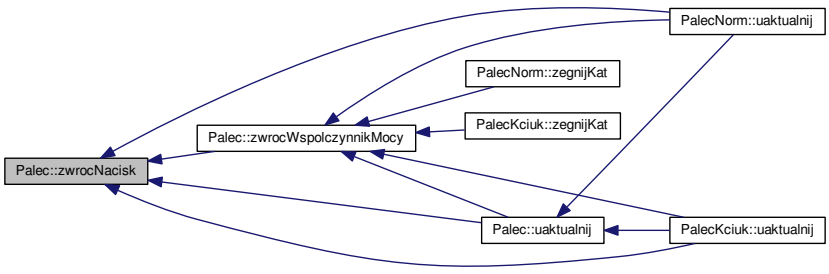
Odwołuje się do CzujnikNacisku::zwrocWartosc().

Odwołania w uaktualnij(), PalecNorm::uaktualnij(), PalecKciuk::uaktualnij() i zwrocWspolczynnikMocy().

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:



4.6.3.8 float Palec::zwrocWspolczynnikMocy ( ) [inline]

Metoda zwracająca wartość współczynnika mocy dla wartości prędkości silnika. Mspółczynnik mocy obliczany jest na podstawie siły nacisku na pole czujnika nacisku.

Zwraca

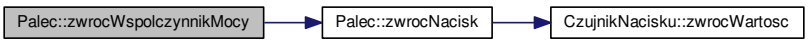
wspolczynnik_mocy

Definicja w linii 78 pliku Palec.cpp.

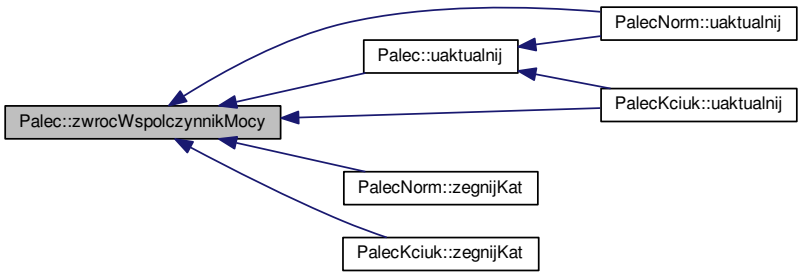
Odwołuje się do zwrocNacisk().

Odwołania w uaktualnij(), PalecNorm::uaktualnij(), PalecKciuk::uaktualnij(), PalecNorm::zegnijKat() i PalecKciuk::zegnijKat().

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:



4.6.4 Dokumentacja atrybutów składowych

4.6.4.1 CzujnikNacisku* Palec::czujnik_nacisku

Definicja w linii 29 pliku Palec.cpp.

Odwołania w PalecKciuk::PalecKciuk() i PalecNorm::PalecNorm().

4.6.4.2 float Palec::dokladnosc_ruchu_deg = 15

Definicja w linii 26 pliku Palec.cpp.

4.6.4.3 float Palec::maksymalny_nacisk_N = 100

Definicja w linii 27 pliku Palec.cpp.

Odwołania w PalecNorm::zegnijKat() i PalecKciuk::zegnijKat().

4.6.4.4 SilnikRegulowany* Palec::napedy[2]

Definicja w linii 30 pliku Palec.cpp.

Odwołania w PalecNorm::uaktualnij(), PalecKciuk::uaktualnij(), PalecNorm::zatrzymaj(), PalecKciuk::zatrzymaj(), PalecNorm::zegnijKat() i PalecKciuk::zegnijKat().

4.6.4.5 char Palec::nr_palca

Definicja w linii 24 pliku Palec.cpp.

Odwołania w PalecKciuk::PalecKciuk(), PalecNorm::PalecNorm(), PalecKciuk::uaktualnij(), PalecNorm::wyprost(), PalecKciuk::wyprost(), PalecNorm::zatrzymaj(), PalecKciuk::zatrzymaj(), PalecNorm::zegnijKat() i PalecKciuk::zegnijKat().

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Palec.cpp

4.7 Dokumentacja klasy PalecKciuk

Klasa PalecKciuk Klasa zaiwera metody które realizują pracę kciuka. Metody te odnoszą sie do palca który w swojej budowie zawiera dwa silniki i serwomechanizm.

Diagram dziedziczenia dla PalecKciuk

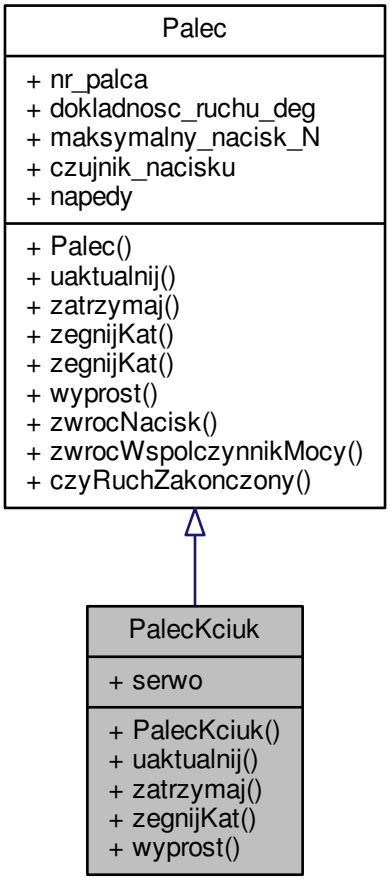
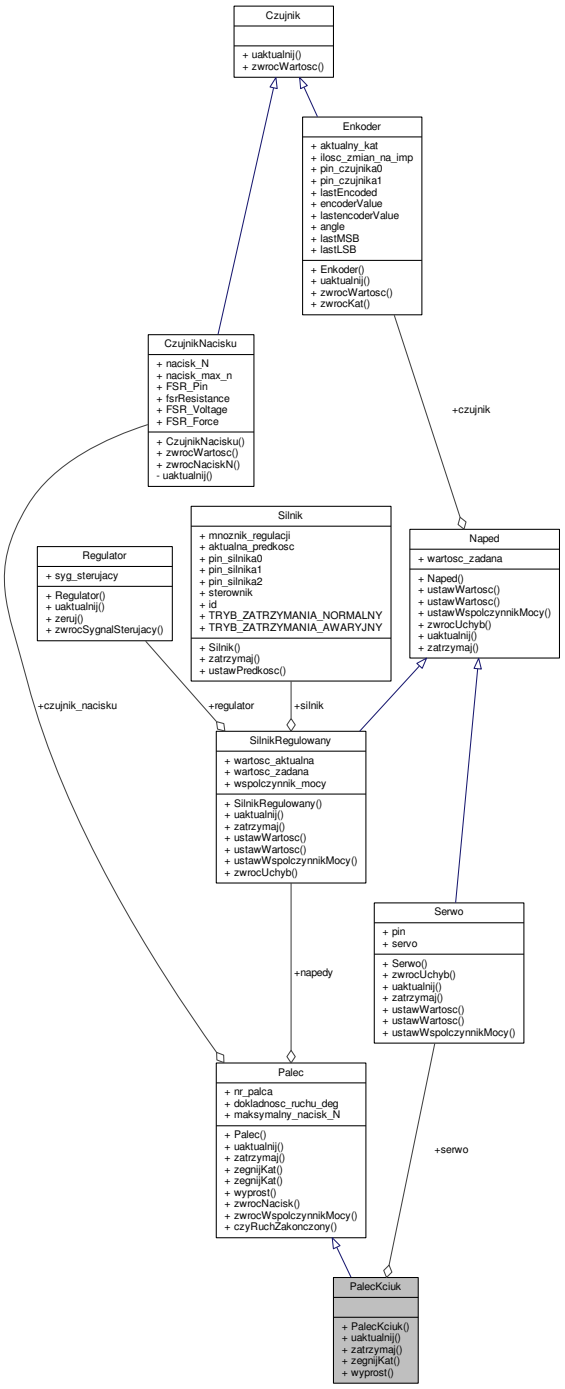


Diagram współpracy dla PalecKciuk:



Metody publiczne

- PalecKciuk (char nr_palca, SilnikRegulowany ∗napedy[ ], Serwo ∗serwo, CzujnikNacisku ∗czuj_nac)

  Konstruktor PalecKciuk realizujący przypisanie otrzymanych parametrów do zmiennych lokalnych.
- int uaktualnij ()

  Metoda uaktualniająca zmienne wewnętrzne obiektu tej klasy.
- int zatrzymaj ()

Metoda zatrzymująca ruch palca odowłując się do metody zatrzymującej napędy i metody zatrzymującej serwomechanizm znajdujęce się w danym palcu.

- int zegnijKat (float c, float a, float b, float maxnacisk)

  Metoda ustawiająca wartość kątów pod jakim mają się zgiąć napędy znajdujące sie w palcu - naped[0] i npaed[1] oraz kąt pod jakim ma się obrócić serwomechanizm.

- int wyprost ()

  Metoda ustawiająca wszystkie napędy w danym palcu oraz serwomechanizm na osiągnięcie położenia początkowego.

Atrybuty publiczne

- Serwo ∗ serwo

4.7.1 Opis szczegółowy

Klasa PalecKciuk Klasa zaiwera metody które realizują pracę kciuka. Metody te odnoszą sie do palca który w swojej budowie zawiera dwa silniki i serwomechanizm.

Definicja w linii 174 pliku Palec.cpp.

4.7.2 Dokumentacja konstruktora i destruktora

4.7.2.1 PalecKciuk::PalecKciuk (char *nr_palca,* SilnikRegulowany ∗ *napedy[ ],* Serwo ∗ *serwo,* CzujnikNacisku ∗ *czuj_nac*) [inline]

Konstruktor PalecKciuk realizujący przypisanie otrzymanych parametrów do zmiennych lokalnych.

Definicja w linii 178 pliku Palec.cpp.

Odwołuje się do czuj_nac, Palec::czujnik_nacisku i Palec::nr_palca.

4.7.3 Dokumentacja funkcji składowych

4.7.3.1 int PalecKciuk::uaktualnij ( ) [inline],[virtual]

Metoda uaktualniająca zmienne wewnętrzne obiektu tej klasy.

Parametry

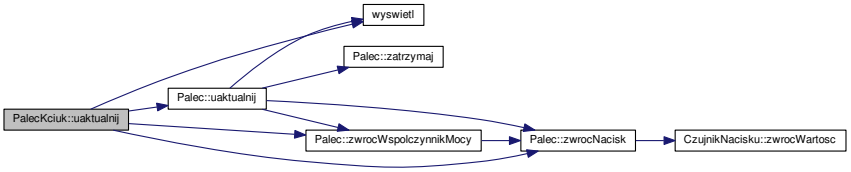
nr_palca	nr ID palca.
napedy[ ]	wskaźnik na obiekt typu Naped przypisany do danego obiektu typu PalecKciuk
serwo	wskaźnik na obiekt typu Naped przypisany do danego obiektu PalecKciuk
czuj_nac	wskaźnik na obiekt typu Czujnik przypisany do danego obiektu typu PalecKciuk

Reimplementowana z Palec.

Definicja w linii 192 pliku Palec.cpp.

Odwołuje się do DEBUG_MODE, Palec::napedy, Palec::nr_palca, Palec::uaktualnij(), wyswietl(), Palec::zwroc↩Nacisk() i Palec::zwrocWspolczynnikMocy().

Oto graf wywołań dla tej funkcji:



4.7.3.2 int PalecKciuk::wyprost ( ) [inline], [virtual]

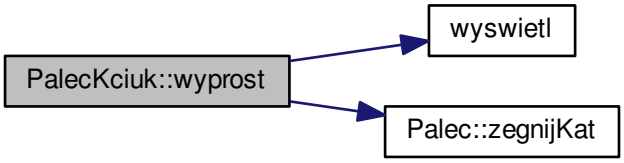
Metoda ustawiająca wszystkie napędy w danym palcu oraz serwomechanizm na osiągnięcie położenia początkowego.

Implementuje Palec.

Definicja w linii 235 pliku Palec.cpp.

Odwołuje się do Palec::nr_palca, wyswietl() i Palec::zegnijKat().

Oto graf wywołań dla tej funkcji:



4.7.3.3 int PalecKciuk::zatrzymaj ( ) [inline], [virtual]

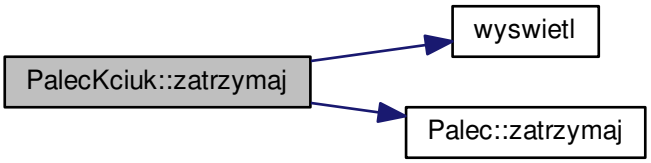
Metoda zatrzymująca ruch palca odowłując się do metody zatrzymującej napędy i metody zatrzymującej serwomechanizm znajdujęce się w danym palcu.

Implementuje Palec.

Definicja w linii 209 pliku Palec.cpp.

Odwołuje się do Palec::napedy, Palec::nr_palca, wyswietl() i Palec::zatrzymaj().

Oto graf wywołań dla tej funkcji:



4.7.3.4 int PalecKciuk::zegnijKat (float *c,* float *a,* float *b,* float *maxnacisk*) [inline],[virtual]

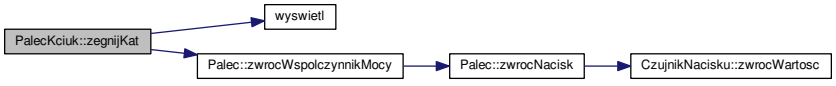
Metoda ustawiająca wartość kątów pod jakim mają się zgiąć napędy znajdujące sie w palcu - naped[0] i npaed[1] oraz kąt pod jakim ma się obrócić serwomechanizm.

Reimplementowana z Palec.

Definicja w linii 221 pliku Palec.cpp.

Odwołuje się do DEBUG_MODE, Palec::maksymalny_nacisk_N, Palec::napedy, Palec::nr_palca, wyswietl() i Palec::zwrocWspolczynnikMocy().

Oto graf wywołań dla tej funkcji:



4.7.4 Dokumentacja atrybutów składowych

4.7.4.1 Serwo* PalecKciuk::serwo

Definicja w linii 176 pliku Palec.cpp.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Palec.cpp

4.8 Dokumentacja klasy PalecNorm

Klasa PalecNorm Klasa zaiwera metody które realizują pracę poszczególnych palów. Metody te odnoszą sie do palców które w swojej budowie zawierają dwa silniki.

Diagram dziedziczenia dla PalecNorm

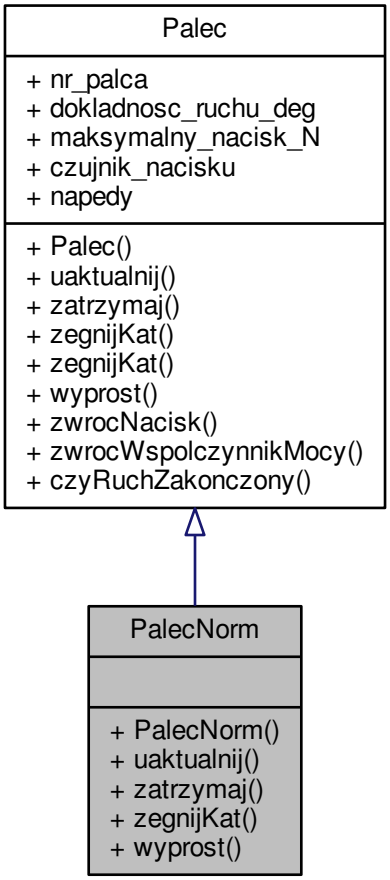
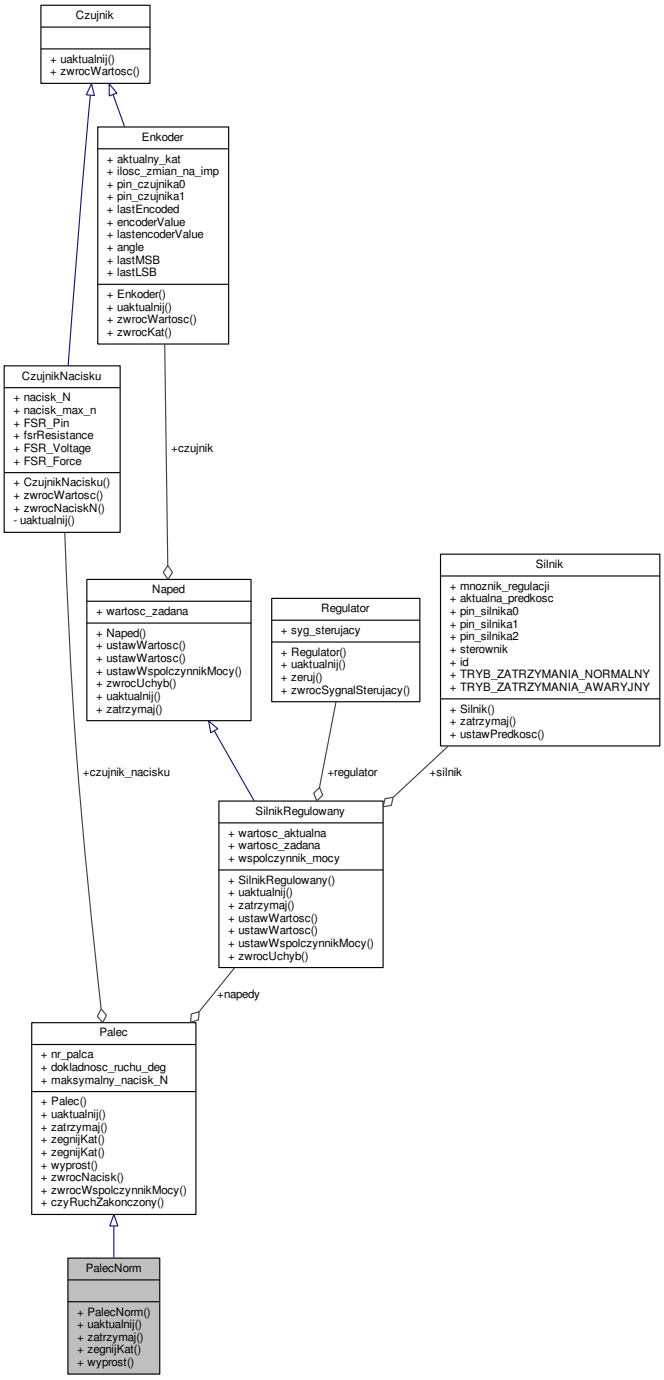


Diagram współpracy dla PalecNorm:



Metody publiczne

- PalecNorm (char nr_palca, SilnikRegulowany ∗napedy[ ], CzujnikNacisku ∗czuj_nac)

  Konstruktor PalecNorm realizujący przypisanie otrzymanych parametrów do zmiennych lokalnych.
- int uaktualnij ()

  Metoda uaktualniająca zmienne wewnętrzne obiektu tej klasy.
- int zatrzymaj ()

Metoda zatrzymująca ruch palca odowłując się do metody zatrzymującej napędy znajdujęce się w danym palcu.

- int zegnijKat (float a, float b, float maxnacisk)

  Metoda ustawiająca wartość kątów pod jakim ma zgiąć się palec (napędy znajdujące sie w palcu - naped[0] i naped[1]).

- int wyprost ()

  Metoda ustawiająca wszystkie napędy w danym palcu na osiągnięcie położenia początkowego.

Dodatkowe Dziedziczone Składowe

4.8.1 Opis szczegółowy

Klasa PalecNorm Klasa zaiwera metody które realizują pracę poszczególnych palów. Metody te odnoszą sie do palców które w swojej budowie zawierają dwa silniki.

Definicja w linii 105 pliku Palec.cpp.

4.8.2 Dokumentacja konstruktora i destruktora

4.8.2.1 PalecNorm::PalecNorm (char *nr_palca,* **SilnikRegulowany** ∗ *napedy[ ],* **CzujnikNacisku** ∗ *czuj_nac*) [inline]

Konstruktor PalecNorm realizujący przypisanie otrzymanych parametrów do zmiennych lokalnych.

Parametry

nr_palca	nr ID palca.
napedy[ ]	wskaźnik na obiekt typu Naped przypisany do danego obiektu typu PalecNorm
czuj_nac	wskaźnik na obiekt typu Czujnik przypisany do danego obiektu typu PalecNorm

Definicja w linii 113 pliku Palec.cpp.

Odwołuje się do czuj_nac, Palec::czujnik_nacisku i Palec::nr_palca.

4.8.3 Dokumentacja funkcji składowych

4.8.3.1 int PalecNorm::uaktualnij ( ) [inline], [virtual]

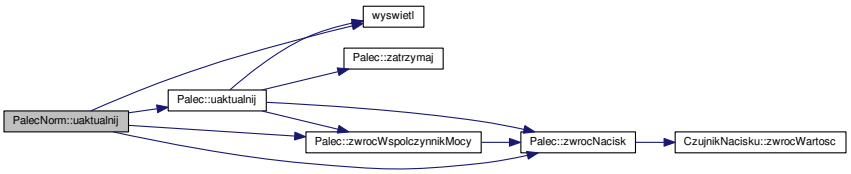
Metoda uaktualniająca zmienne wewnętrzne obiektu tej klasy.

Reimplementowana z Palec.

Definicja w linii 121 pliku Palec.cpp.

Odwołuje się do Palec::napedy, Palec::uaktualnij(), wyswietl(), Palec::zwrocNacisk() i Palec::zwrocWspolczynnikMocy().

Oto graf wywołań dla tej funkcji:



4.8.3.2 int PalecNorm::wyprost ( ) [inline], [virtual]

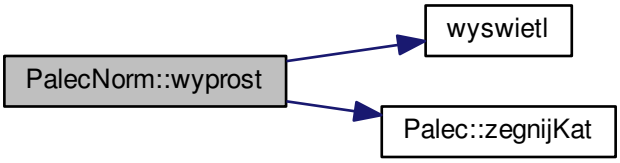
Metoda ustawiająca wszystkie napędy w danym palcu na osiągnięcie położenia początkowego.

Implementuje Palec.

Definicja w linii 160 pliku Palec.cpp.

Odwołuje się do DEBUG_MODE, Palec::nr_palca, wyswietl() i Palec::zegnijKat().

Oto graf wywołań dla tej funkcji:



4.8.3.3 int PalecNorm::zatrzymaj ( ) [inline], [virtual]

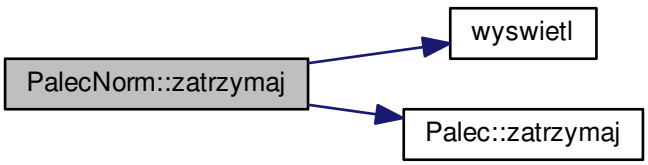
Metoda zatrzymująca ruch palca odowłując się do metody zatrzymującej napędy znajdujęce się w danym palcu.

Implementuje Palec.

Definicja w linii 138 pliku Palec.cpp.

Odwołuje się do DEBUG_MODE, Palec::napedy, Palec::nr_palca, wyswietl() i Palec::zatrzymaj().

Oto graf wywołań dla tej funkcji:



4.8.3.4 int PalecNorm::zegnijKat (float *a*, float *b*, float *maxnacisk*) [inline], [virtual]

Metoda ustawiająca wartość kątów pod jakim ma zgiąć się palec (napędy znajdujące sie w palcu - naped[0] i naped[1]).

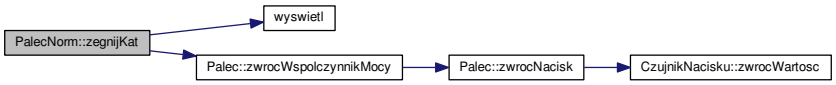
< Metoda uaktualniająca zmienne wewnętrzne obiektu tej klasy.

Reimplementowana z Palec.

Definicja w linii 148 pliku Palec.cpp.

Odwołuje się do DEBUG_MODE, Palec::maksymalny_nacisk_N, Palec::napedy, Palec::nr_palca, wyswietl() i Palec::zwrocWspolczynnikMocy().

Oto graf wywołań dla tej funkcji:



Dokumentacja dla tej klasy została wygenerowana z pliku:

- Palec.cpp

4.9 Dokumentacja klasy Regulator

Bazowa klasa Regulator zawierająca metody wirtualne które są nadpisywane przez metody klas dziedziczących.

Diagram dziedziczenia dla Regulator

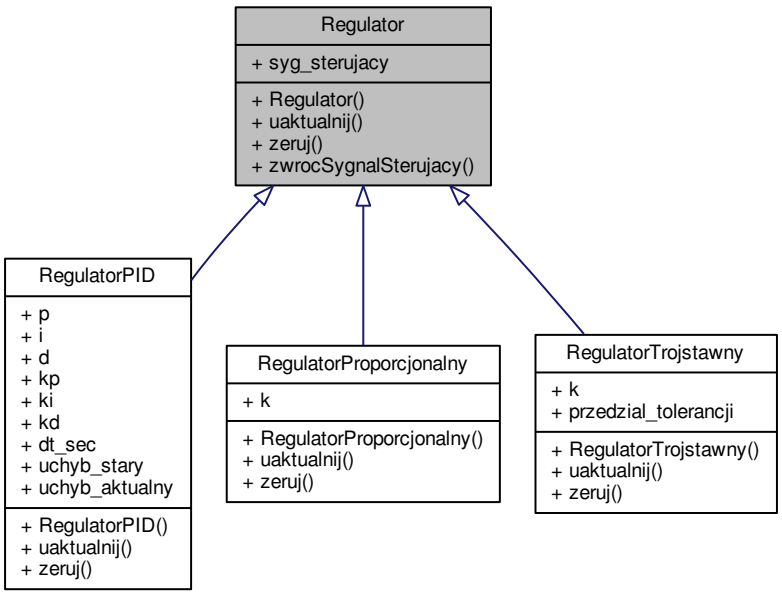
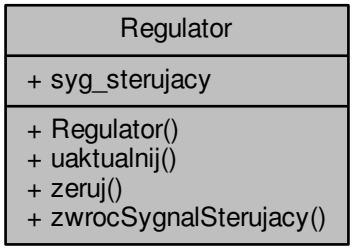


Diagram współpracy dla Regulator:



Metody publiczne

- Regulator ()
- virtual int uaktualnij (float uchyb)=0
- virtual int zeruj ()=0
- virtual float zwrocSygnalSterujacy ()

Atrybuty publiczne

- float syg_sterujacy

4.9.1 Opis szczegółowy

Bazowa klasa Regulator zawierająca metody wirtualne które są nadpisywane przez metody klas dziedziczących.

Definicja w linii 19 pliku Regulator.cpp.

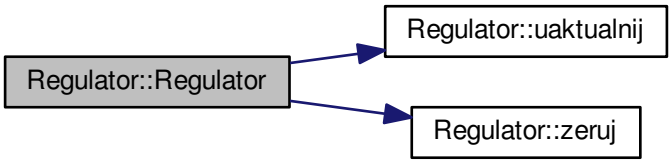
4.9.2 Dokumentacja konstruktora i destruktora

4.9.2.1 Regulator::Regulator ( ) [inline]

Definicja w linii 23 pliku Regulator.cpp.

Odwołuje się do uaktualnij() i zeruj().

Oto graf wywołań dla tej funkcji:



4.9.3 Dokumentacja funkcji składowych

4.9.3.1 virtual int Regulator::uaktualnij (float *uchyb*) [pure virtual]

Implementowany w RegulatorTrojstawny, RegulatorProporcjonalny i RegulatorPID.

Odwołania w Regulator().

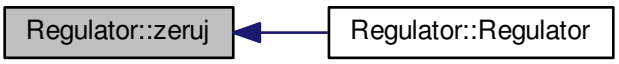
Oto graf wywoływań tej funkcji:

4.9.3.2 virtual int Regulator::zeruj ( ) [pure virtual]

Implementowany w RegulatorTrojstawny, RegulatorProporcjonalny i RegulatorPID.

Odwołania w Regulator().

Oto graf wywoływań tej funkcji:



4.9.3.3 virtual float Regulator::zwrocSygnalSterujacy ( ) [inline], [virtual]

Definicja w linii 31 pliku Regulator.cpp.

Odwołuje się do syg_sterujacy.

4.9.4 Dokumentacja atrybutów składowych

4.9.4.1 float Regulator::syg_sterujacy

Definicja w linii 21 pliku Regulator.cpp.

Odwołania w RegulatorPID::uaktualnij(), RegulatorProporcjonalny::uaktualnij(), RegulatorTrojstawny::uaktualnij() i zwrocSygnalSterujacy().

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Regulator.cpp

4.10 Dokumentacja klasy RegulatorPID

Klasa RegulatorPID. Klasa ta zawiera algorytm liczący wartość sygnału sterującego prędkością silników.

Diagram dziedziczenia dla RegulatorPID

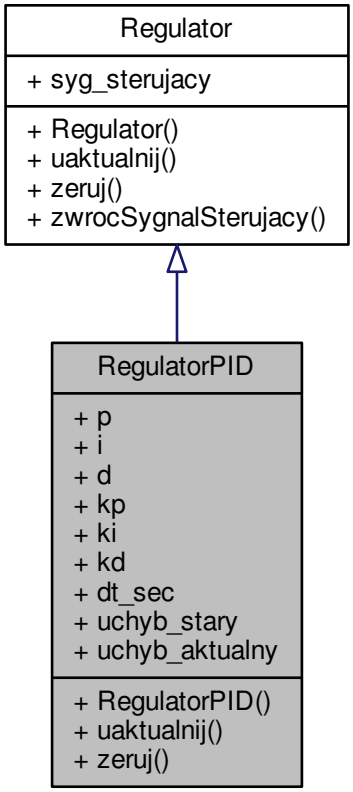
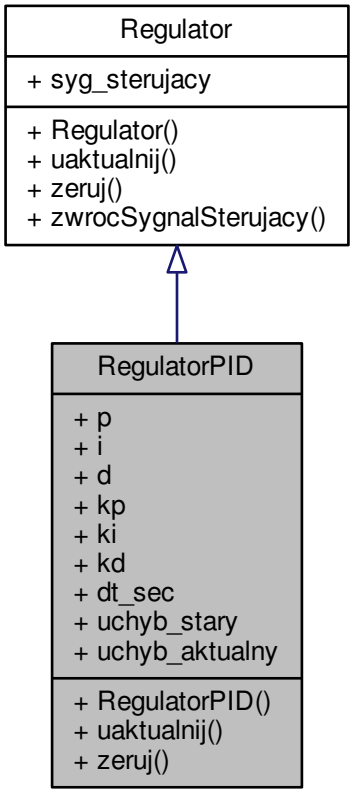


Diagram współpracy dla RegulatorPID:



Metody publiczne

- RegulatorPID (float kp, float ki, float kd, float dt_s)

  Konstruktor zawierający definicje otrzymanych paramentow.
- int uaktualnij (float uchyb)

  Metoda licząca wartość sygnału sterującego przyjmując jako parametr wartość aktualnego uchybu.
- int zeruj ()

  Metoda zerująca wartości członów regulatora.

Atrybuty publiczne

- float p
- float i
- float d
- float kp
- float ki
- float kd
- float dt_sec
- float uchyb_stary
- float uchyb_aktualny

4.10.1 Opis szczegółowy

Klasa RegulatorPID. Klasa ta zawiera algorytm liczący wartość sygnału sterującego prędkością silników.

Definicja w linii 40 pliku Regulator.cpp.

4.10.2 Dokumentacja konstruktora i destruktora

4.10.2.1 RegulatorPID::RegulatorPID (float *kp,* float *ki,* float *kd,* float *dt_s*) [inline]

Konstruktor zawierający definicje otrzymanych paramentow.

Definicja w linii 49 pliku Regulator.cpp.

4.10.3 Dokumentacja funkcji składowych

4.10.3.1 int RegulatorPID::uaktualnij (float *uchyb*) [inline], [virtual]

Metoda licząca wartość sygnału sterującego przyjmując jako parametr wartość aktualnego uchybu.

Implementuje Regulator.

Definicja w linii 66 pliku Regulator.cpp.

Odwołuje się do DEBUG_MODE, dt_sec, Regulator::syg_sterujacy i wyswietl().

Oto graf wywołań dla tej funkcji:

4.10.3.2 int RegulatorPID::zeruj ( ) [inline], [virtual]

Metoda zerująca wartości członów regulatora.

Implementuje Regulator.

Definicja w linii 83 pliku Regulator.cpp.

4.10.4 Dokumentacja atrybutów składowych

4.10.4.1 float RegulatorPID::d

Definicja w linii 42 pliku Regulator.cpp.

4.10.4.2 float RegulatorPID::dt_sec

Definicja w linii 44 pliku Regulator.cpp.

4.10.4.3 float RegulatorPID::i

Definicja w linii 42 pliku Regulator.cpp.

4.10.4.4 float RegulatorPID::kd

Definicja w linii 43 pliku Regulator.cpp.

4.10.4.5 float RegulatorPID::ki

Definicja w linii 43 pliku Regulator.cpp.

4.10.4.6 float RegulatorPID::kp

Definicja w linii 43 pliku Regulator.cpp.

4.10.4.7 float RegulatorPID::p

Definicja w linii 42 pliku Regulator.cpp.

4.10.4.8 float RegulatorPID::uchyb_aktualny

Definicja w linii 47 pliku Regulator.cpp.

4.10.4.9 float RegulatorPID::uchyb_stary

Definicja w linii 46 pliku Regulator.cpp.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Regulator.cpp

4.11 Dokumentacja klasy RegulatorProporcjonalny

Klasa RegulatorProporcjonalny. Klasa ta zawiera algorytm regulatora proporcjonalnego liczący wartość sygnału sterującego prędkością silników.

Diagram dziedziczenia dla RegulatorProporcjonalny

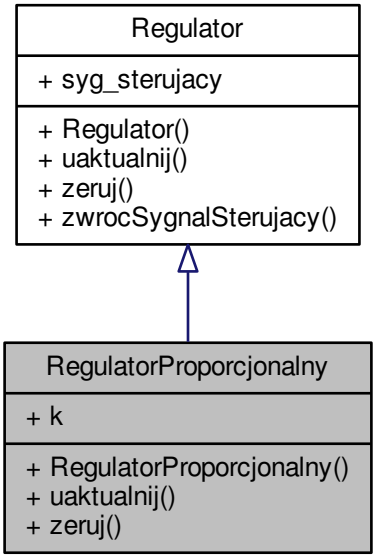
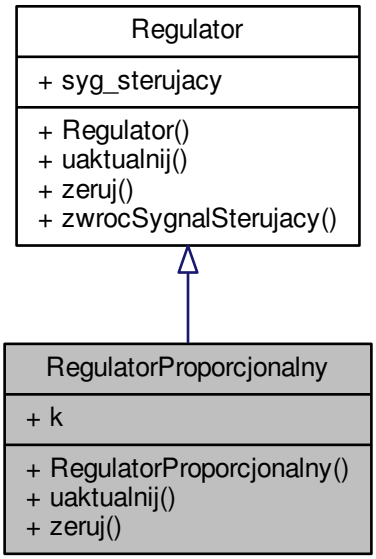


Diagram współpracy dla RegulatorProporcjonalny:



Metody publiczne

- RegulatorProporcjonalny (float k)

  Konstruktor przypisujący wartość otrzymanego parametru do zmiennej lokalnej.
- int uaktualnij (float uchyb)

  Metoda liczy wartość sygnału sterującego na podstawie otrzymanego parametru uchybu.
- int zeruj ()

  Metoda zerująca wartość wzmocnienia regulatora w celu przyjęcia kolejnej.

Atrybuty publiczne

- float k

4.11.1 Opis szczegółowy

Klasa RegulatorProporcjonalny. Klasa ta zawiera algorytm regulatora proporcjonalnego liczący wartość sygnału sterującego prędkością silników.

Definicja w linii 97 pliku Regulator.cpp.

4.11.2 Dokumentacja konstruktora i destruktora

4.11.2.1 RegulatorProporcjonalny::RegulatorProporcjonalny (float *k*) [inline]

Konstruktor przypisujący wartość otrzymanego parametru do zmiennej lokalnej.

Definicja w linii 101 pliku Regulator.cpp.

4.11.3 Dokumentacja funkcji składowych

4.11.3.1 int RegulatorProporcjonalny::uaktualnij (float *uchyb*) [inline], [virtual]

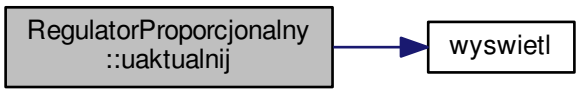
Metoda liczy wartość sygnału sterującego na podstawie otrzymanego parametru uchybu.

Implementuje Regulator.

Definicja w linii 107 pliku Regulator.cpp.

Odwołuje się do DEBUG_MODE, Regulator::syg_sterujacy i wyswietl().

Oto graf wywołań dla tej funkcji:



4.11.3.2 int RegulatorProporcjonalny::zeruj ( ) [inline], [virtual]

Metoda zerująca wartość wzmocnienia regulatora w celu przyjęcia kolejnej.

Implementuje Regulator.

Definicja w linii 118 pliku Regulator.cpp.

4.11.4 Dokumentacja atrybutów składowych

4.11.4.1 float RegulatorProporcjonalny::k

Definicja w linii 99 pliku Regulator.cpp.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Regulator.cpp

4.12 Dokumentacja klasy RegulatorTrojstawny

Klasa RegulatorTrojstawny. Klasa ta zawiera algorytm regulatora trójstawnego liczący wartość sygnału sterującego prędkością silników.

Diagram dziedziczenia dla RegulatorTrojstawny

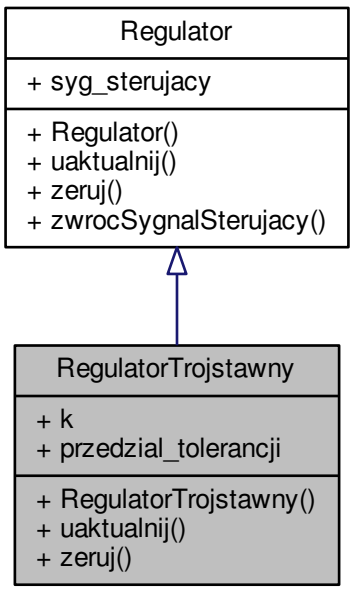
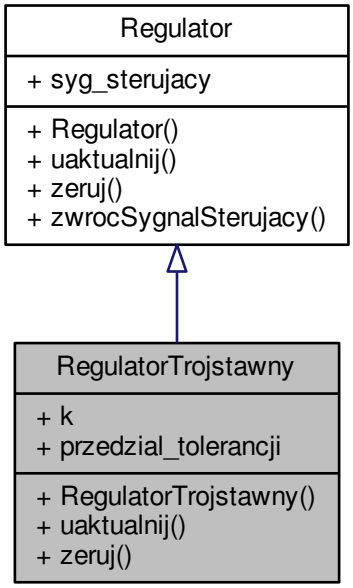


Diagram współpracy dla RegulatorTrojstawny:



Metody publiczne

- RegulatorTrojstawny (float k, float tolerancja)

  Metoda przypisująca wartości otrzymnaych parametrów do zmiennych lokalnych.
- int uaktualnij (float uchyb)

  Metoda uaktualniająca zmienne wewnętrzne obiektu tej klasy.
- int zeruj ()

  Metoda zerująca wartości regulatora w celu przyjęcia nowych.

Atrybuty publiczne

- float k
- float przedzial_tolerancji

4.12.1 Opis szczegółowy

Klasa RegulatorTrojstawny. Klasa ta zawiera algorytm regulatora trójstawnego liczący wartość sygnału sterującego prędkością silników.

Definicja w linii 128 pliku Regulator.cpp.

4.12.2 Dokumentacja konstruktora i destruktora

4.12.2.1 RegulatorTrojstawny::RegulatorTrojstawny (float *k*, float *tolerancja*) [inline]

Metoda przypisująca wartości otrzymnaych parametrów do zmiennych lokalnych.

Parametry

k	wartość maksymalna wmocnienienia
przedzial_tolerancji	przedzial tolerancji dla którego liczona ma być wartość uchybu i sygnał sterujący.

Definicja w linii 136 pliku Regulator.cpp.

4.12.3 Dokumentacja funkcji składowych

4.12.3.1 int RegulatorTrojstawny::uaktualnij (float *uchyb*) [inline], [virtual]

Metoda uaktualniająca zmienne wewnętrzne obiektu tej klasy.

Parametry

uchyb	parametr na podtawie której metoda ta wylicza wartość sygnału sterującego.

Implementuje Regulator.

Definicja w linii 144 pliku Regulator.cpp.

Odwołuje się do DEBUG_MODE i Regulator::syg_sterujacy.

4.12.3.2 int RegulatorTrojstawny::zeruj ( ) [inline], [virtual]

Metoda zerująca wartości regulatora w celu przyjęcia nowych.

Implementuje Regulator.

Definicja w linii 162 pliku Regulator.cpp.

4.12.4 Dokumentacja atrybutów składowych

4.12.4.1 float RegulatorTrojstawny::k

Definicja w linii 130 pliku Regulator.cpp.

4.12.4.2 float RegulatorTrojstawny::przedzial_tolerancji

Definicja w linii 131 pliku Regulator.cpp.

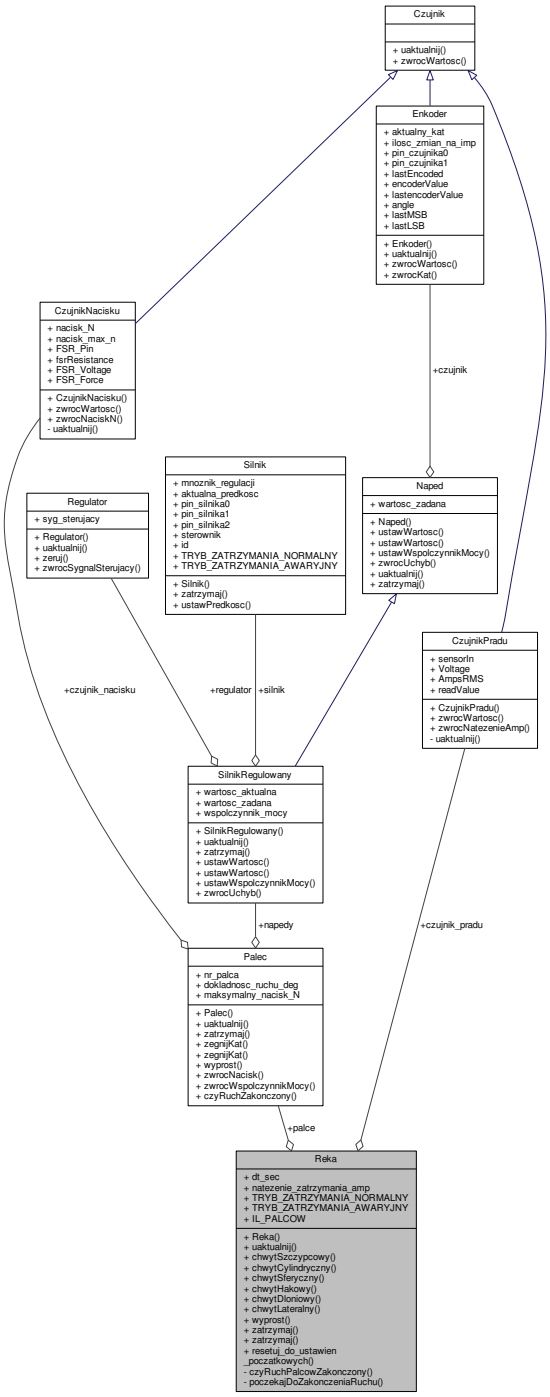
Dokumentacja dla tej klasy została wygenerowana z pliku:

- Regulator.cpp

4.13 Dokumentacja klasy Reka

Klasa Reka. Jest to główna klasa programu. Zarządza pracą wszystkich palców oraz czujnika prądu.

Diagram współpracy dla Reka:



Metody publiczne

- Reka (Palec ∗palce[ ], CzujnikPradu ∗czujnik_pradu, float nat_zatrz, float dt)

Konstruktor klasy Reka przypisujący wartości otrzymanych parametrów do zmiennych lokalnych.

- int uaktualnij ()

  Metoda uaktualniająca wszystkie elementy składowe ręki. Jest propagowana wgłąb kolejnych elementów składowych.
- int chwytSzczypcowy (bool pusty_chwyt, float max_sila)

  Metoda realizująca chwyt szczypcowy.
- int chwytCylindryczny (bool pusty_chwyt, float max_sila)

  Metoda realizująca chwyt cylindryczny.
- int chwytSferyczny (bool pusty_chwyt, float max_sila)

  Metoda realizująca chwyt sferyczny.
- int chwytHakowy (bool pusty_chwyt, float max_sila)

  Metoda realizująca chwyt hakowy.
- int chwytDloniowy (bool pusty_chwyt, float max_sila)

  Metoda realizująca chwyt dłoniowy.
- int chwytLateralny (bool pusty_chwyt, float max_sila)

  Metoda realizująca chwyt lateralny.
- int wyprost ()

  Metoda realizująca wyprostowanie palców ręki.
- int zatrzymaj (int tryb_zatrz)

  Metoda realizująca zatrzymanie w wybranym przez użytkownika trybie (Normalny, Awaryjny).
- int zatrzymaj ()

  Metoda realizująca zatrzymanie w trybie normalnym.
- int resetuj_do_ustawien_poczatkowych ()

  Metoda resetująca rękę do ustawien poczatkowych, oraz jej posiadajacych elementow.

Atrybuty publiczne

- float dt_sec = 1.0/100.0
- Palec * palce [IL_PALCOW]
- CzujnikPradu * czujnik_pradu
- float natezenie_zatrzymania_amp

Statyczne atrybuty publiczne

- static const char TRYB_ZATRZYMANIA_NORMALNY = 0
- static const char TRYB_ZATRZYMANIA_AWARYJNY = 1
- static const char IL_PALCOW = 5

Metody prywatne

- bool czyRuchPalcowZakonczony ()

  Metoda zwracająca wartość, która informuje, czy wszystkie palce zakończyły swój ruch.
- int poczekajDoZakonczeniaRuchu ()

  Metoda blokująca wykonywanie programu do momentu zakończenia wykonywania danego chwytu (ruchu).

4.13.1 Opis szczegółowy

Klasa Reka. Jest to główna klasa programu. Zarządza pracą wszystkich palców oraz czujnika prądu.

Definicja w linii 21 pliku Reka.cpp.

4.13.2 Dokumentacja konstruktora i destruktora

4.13.2.1 Reka::Reka (**Palec** ∗ *palce[ ],* **CzujnikPradu** ∗ *czujnik_pradu,* float *nat_zatrz,* float *dt*) [inline]

Konstruktor klasy Reka przypisujący wartości otrzymanych parametrów do zmiennych lokalnych.

Konstruktor klasy Reka.

Parametry

palce	Tablica wskaźników na obiekty palców.
czujnik_pradu	Wskaźnik do obiektu czujnika prądu.
nat_zatrz	Zmienna natężenia zatrzymania ruchu w trybie awaryjnym.
dt	Czas próbkowania - steruje ile razy na sekundę wykonywane są obliczenia.

Definicja w linii 74 pliku Reka.cpp.

Odwołuje się do czujnik_pradu.

4.13.3 Dokumentacja funkcji składowych

4.13.3.1 int Reka::chwytCylindryczny (bool *pusty_chwyt,* float *max_sila*) [inline]

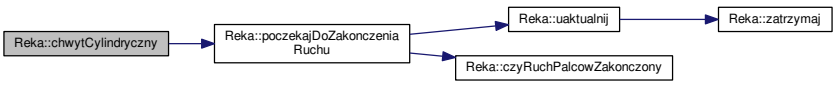
Metoda realizująca chwyt cylindryczny.

Metoda realizująca chwyt cylindryczny.

Definicja w linii 120 pliku Reka.cpp.

Odwołuje się do poczekajDoZakonczeniaRuchu().

Oto graf wywołań dla tej funkcji:



4.13.3.2 int Reka::chwytDloniowy (bool *pusty_chwyt,* float *max_sila*) [inline]

Metoda realizująca chwyt dłoniowy.

Metoda realizująca chwyt dłoniowy.

Definicja w linii 172 pliku Reka.cpp.

4.13.3.3 int Reka::chwytHakowy (bool *pusty_chwyt,* float *max_sila*) [inline]

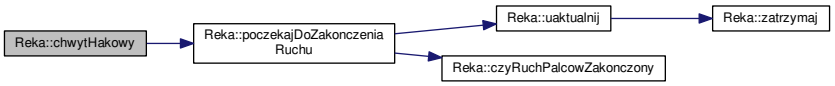
Metoda realizująca chwyt hakowy.

Metoda realizująca chwyt hakowy.

Definicja w linii 155 pliku Reka.cpp.

Odwołuje się do poczekajDoZakonczeniaRuchu().

Oto graf wywołań dla tej funkcji:



4.13.3.4 int Reka::chwytLateralny (bool *pusty_chwyt,* float *max_sila*) [inline]

Metoda realizująca chwyt lateralny.

Metoda realizująca chwyt lateralny.

Definicja w linii 181 pliku Reka.cpp.

4.13.3.5 int Reka::chwytSferyczny (bool *pusty_chwyt,* float *max_sila*) [inline]

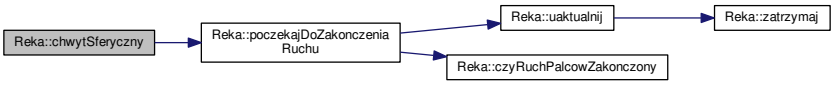
Metoda realizująca chwyt sferyczny.

Metoda realizująca chwyt sferyczny.

Definicja w linii 138 pliku Reka.cpp.

Odwołuje się do poczekajDoZakonczeniaRuchu().

Oto graf wywołań dla tej funkcji:



4.13.3.6 int Reka::chwytSzczypcowy (bool *pusty_chwyt,* float *max_sila*) [inline]

Metoda realizująca chwyt szczypcowy.

Metoda realizująca chwyt szczypcowy.

Parametry

pusty_chwyt	Zmienna określająca rodzaj wykonywanego chwytu (pusty lub chwyt przedmiotu)
zmienna	określająca maksywmalną siłę z jaką ma być chwytany dany przedmiot.

Definicja w linii 111 pliku Reka.cpp.

4.13.3.7 bool Reka::czyRuchPalcowZakonczony ( ) [inline], [private]

Metoda zwracająca wartość, która informuje, czy wszystkie palce zakończyły swój ruch.

Metoda zwracająca wartość, która informuje, czy wszystkie palce zakończyły swój ruch.

Zwraca

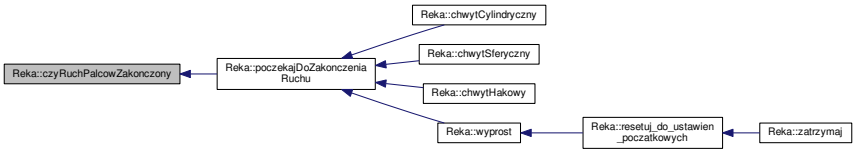
Wartość boolowska, informująca o zakończeniu ruchu wszystkich palców.

Definicja w linii 28 pliku Reka.cpp.

Odwołuje się do IL_PALCOW i palce.

Odwołania w poczekajDoZakonczeniaRuchu().

Oto graf wywoływań tej funkcji:



4.13.3.8 int Reka::poczekajDoZakonczeniaRuchu ( ) [inline], [private]

Metoda blokująca wykonywanie programu do momentu zakończenia wykonywania danego chwytu (ruchu).

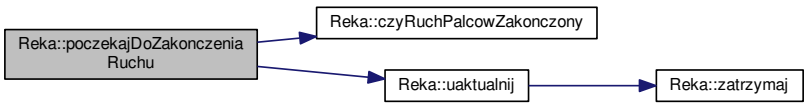
Metoda blokująca wykonywanie programu do momentu zakończenia wykonywania danego chwytu (ruchu).

Definicja w linii 40 pliku Reka.cpp.

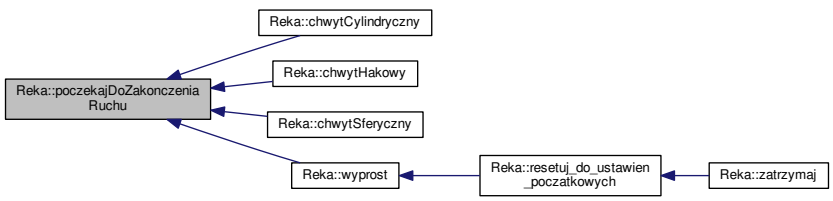
Odwołuje się do czyRuchPalcowZakonczony(), dt_sec i uaktualnij().

Odwołania w chwytCylindryczny(), chwytHakowy(), chwytSferyczny() i wyprost().

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:



4.13.3.9 int Reka::resetuj_do_ustawien_poczatkowych ( ) [inline]

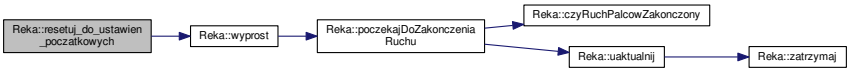
Metoda resetująca rękę do ustawien poczatkowych, oraz jej posiadajacych elementow.

Definicja w linii 236 pliku Reka.cpp.

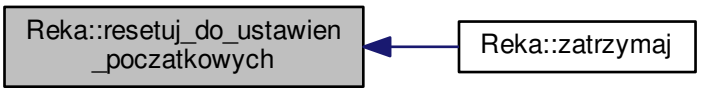
Odwołuje się do IL_PALCOW i wyprost().

Odwołania w zatrzymaj().

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:



4.13.3.10 int Reka::uaktualnij ( ) [inline]

Metoda uaktualniająca wszystkie elementy składowe ręki. Jest propagowana wgłąb kolejnych elementów składowych.

Metoda uaktualniająca wszystkie elementy składowe ręki. Jest propagowana wgłąb kolejnych elementów składowych.

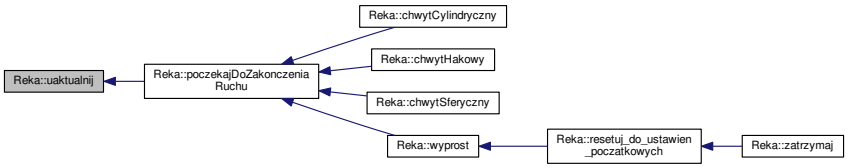
Definicja w linii 88 pliku Reka.cpp.

Odwołuje się do IL_PALCOW i zatrzymaj().

Odwołania w poczekajDoZakonczeniaRuchu().

Oto graf wywołań dla tej funkcji:

Oto graf wywoływań tej funkcji:



4.13.3.11 int Reka::wyprost ( ) [inline]

Metoda realizująca wyprostowanie palców ręki.

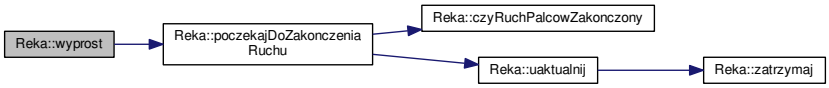
Metoda realizująca wyprostowanie palców ręki.

Definicja w linii 190 pliku Reka.cpp.

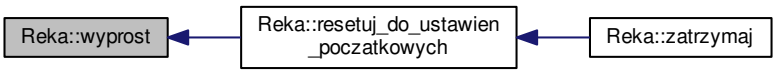
Odwołuje się do IL_PALCOW i poczekajDoZakonczeniaRuchu().

Odwołania w resetuj_do_ustawien_poczatkowych().

Oto graf wywołań dla tej funkcji:



Oto graf wywoływań tej funkcji:



4.13.3.12 int Reka::zatrzymaj (int *tryb_zatrz*) [inline]

Metoda realizująca zatrzymanie w wybranym przez użytkownika trybie (Normalny, Awaryjny).

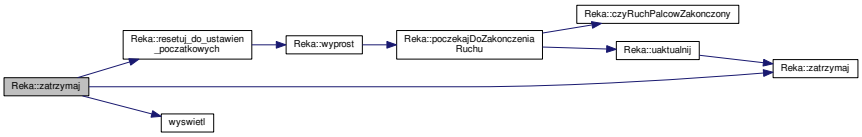
Parametry

tryb_zatrz	tryb zatrzymania programu.

Definicja w linii 209 pliku Reka.cpp.

Odwołuje się do IL_PALCOW, resetuj_do_ustawien_poczatkowych(), wyswietl() i zatrzymaj().

Oto graf wywołań dla tej funkcji:



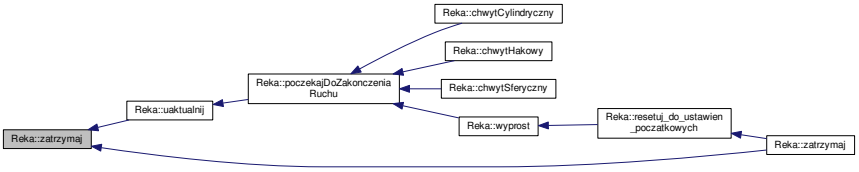
4.13.3.13 int Reka::zatrzymaj ( ) [inline]

Metoda realizująca zatrzymanie w trybie normalnym.

Definicja w linii 230 pliku Reka.cpp.

Odwołania w uaktualnij() i zatrzymaj().

Oto graf wywoływań tej funkcji:



4.13.4 Dokumentacja atrybutów składowych

4.13.4.1 CzujnikPradu* Reka::czujnik_pradu

Definicja w linii 62 pliku Reka.cpp.

Odwołania w Reka().

4.13.4.2 float Reka::dt_sec = 1.0/100.0

Definicja w linii 59 pliku Reka.cpp.

Odwołania w poczekajDoZakonczeniaRuchu().

4.13.4.3 const char Reka::IL_PALCOW = 5 [static]

Definicja w linii 57 pliku Reka.cpp.

Odwołania w czyRuchPalcowZakonczony(), resetuj_do_ustawien_poczatkowych(), uaktualnij(), wyprost() i zatrzymaj().

4.13.4.4 float Reka::natezenie_zatrzymania_amp

Definicja w linii 63 pliku Reka.cpp.

4.13.4.5 Palec* Reka::palce[IL_PALCOW]

Definicja w linii 61 pliku Reka.cpp.

Odwołania w czyRuchPalcowZakonczony().

4.13.4.6 const char Reka::TRYB_ZATRZYMANIA_AWARYJNY = 1 [static]

Definicja w linii 55 pliku Reka.cpp.

4.13.4.7 const char Reka::TRYB_ZATRZYMANIA_NORMALNY = 0 [static]

Parametry

TRYB_ZATRZYMANIA_NORMALNY	Zmienna określająca zatrzymanie ruchu i wznowienie pracy
TRYB_ZATRZYMANIA_AWARYJNY	Zmienna określająca zatrzymanie ruchu i wznawianie pracy ręki po uprzednim resecie do ustawien poczatkowych

Definicja w linii 54 pliku Reka.cpp.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Reka.cpp

4.14 Dokumentacja klasy Serwo

Klasa Serwo dziedziczy po klasie Naped. Zawiera w sobie obiekt klasy Servo i metody obsługującej stwerowanie serwomechanizmem.

Diagram dziedziczenia dla Serwo

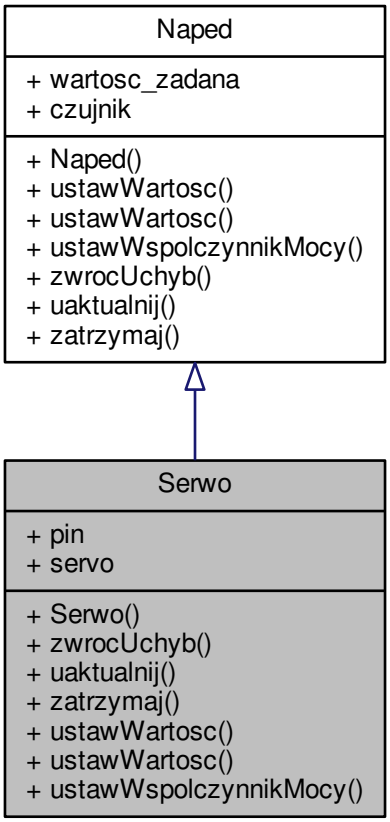
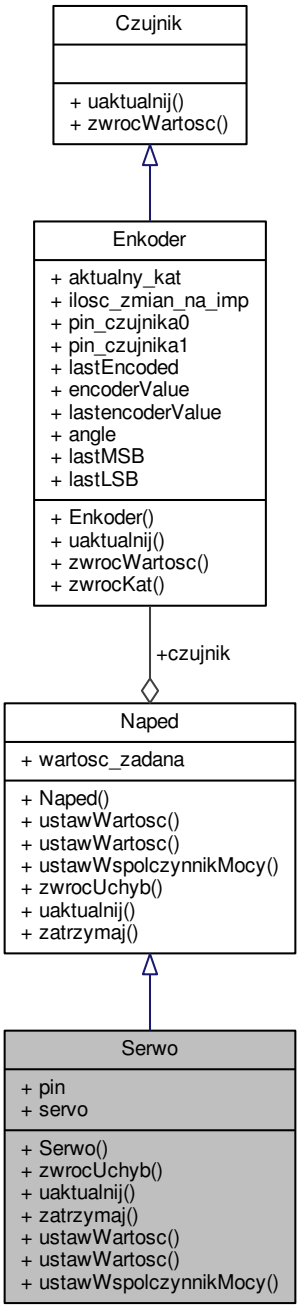


Diagram współpracy dla Serwo:



Metody publiczne

- Serwo (int pin)
- float zwrocUchyb ()
- int uaktualnij ()
- int zatrzymaj ()
- int ustawWartosc (float kat)

metoda ustawiajaca zadaną wartość kąta na wejście serwomechanizmu.

- int ustawWartosc (float kat, float wspolmocy)
- int ustawWspolczynnikMocy (float wspol_mocy)

Atrybuty publiczne

- int pin
- Servo servo

4.14.1 Opis szczegółowy

Klasa Serwo dziedziczy po klasie Naped. Zawiera w sobie obiekt klasy Servo i metody obsługującej stwerowanie serwomechanizmem.

Definicja w linii 49 pliku Naped.cpp.

4.14.2 Dokumentacja konstruktora i destruktora

4.14.2.1 Serwo::Serwo (int *pin*) [inline]

Definicja w linii 54 pliku Naped.cpp.

4.14.3 Dokumentacja funkcji składowych

4.14.3.1 int Serwo::uaktualnij ( ) [inline], [virtual]

Implementuje Naped.

Definicja w linii 64 pliku Naped.cpp.

4.14.3.2 int Serwo::ustawWartosc (float *kat*) [inline], [virtual]

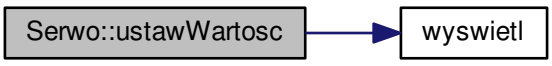
metoda ustawiajaca zadaną wartość kąta na wejście serwomechanizmu.

Implementuje Naped.

Definicja w linii 75 pliku Naped.cpp.

Odwołuje się do Naped::wartosc_zadana i wyswietl().

Oto graf wywołań dla tej funkcji:



4.14.3.3 int Serwo::ustawWartosc (float *kat,* float *wspolmocy*) [inline], [virtual]

Implementuje Naped.

Definicja w linii 83 pliku Naped.cpp.

Odwołuje się do Naped::ustawWartosc().

Oto graf wywołań dla tej funkcji:

4.14.3.4 int Serwo::ustawWspolczynnikMocy (float *wspol_mocy*) [inline], [virtual]

Implementuje Naped.

Definicja w linii 87 pliku Naped.cpp.

4.14.3.5 int Serwo::zatrzymaj ( ) [inline], [virtual]

Implementuje Naped.

Definicja w linii 69 pliku Naped.cpp.

4.14.3.6 float Serwo::zwrocUchyb ( ) [inline], [virtual]

Implementuje Naped.

Definicja w linii 60 pliku Naped.cpp.

4.14.4 Dokumentacja atrybutów składowych

4.14.4.1 int Serwo::pin

Definicja w linii 51 pliku Naped.cpp.

4.14.4.2 Servo Serwo::servo

Definicja w linii 52 pliku Naped.cpp.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Naped.cpp

4.15 Dokumentacja klasy Silnik

Klasa Silnik zawierająca metody sterujące pracą silników.

Diagram współpracy dla Silnik:

Metody publiczne

- Silnik (DualMC33926MotorShield ∗ster, int id, float mnoznik_reg)

  Konstruktor zawierający trzy parametry - rodzaj setrownika, id silnika i mnożnik regulacji.
- virtual int zatrzymaj ()

  wirtualna metoda bazowa zatrzumująca pracę silnika
- virtual int ustawPredkosc (float predkosc)

  wirtualna metoda bazowa ustawiająca zadaną prędkość dla silnika w zależności od nr ID silnika.

Atrybuty publiczne

- float mnoznik_regulacji
- float aktualna_predkosc
- int pin_silnika0
- int pin_silnika1
- int pin_silnika2
- DualMC33926MotorShield ∗ sterownik
- int id

Statyczne atrybuty publiczne

- static const char TRYB_ZATRZYMANIA_NORMALNY = 0
- static const char TRYB_ZATRZYMANIA_AWARYJNY = 1

4.15.1 Opis szczegółowy

Klasa Silnik zawierająca metody sterujące pracą silników.

Definicja w linii 21 pliku Silnik.cpp.

4.15.2 Dokumentacja konstruktora i destruktora

4.15.2.1 Silnik::Silnik (DualMC33926MotorShield * *ster,* int *id,* float *mnoznik_reg*) [inline]

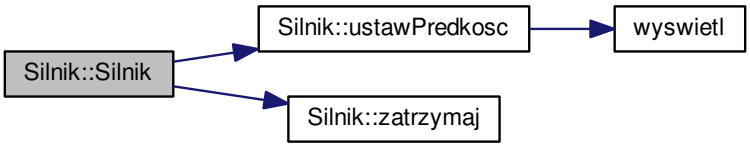
Konstruktor zawierający trzy parametry - rodzaj setrownika, id silnika i mnożnik regulacji.

Mnożnik regulacji stosuje się w celu zwiększenia wartości sygnału sterującego. Domyślnie przyjmuję wartość 1.

Definicja w linii 37 pliku Silnik.cpp.

Odwołuje się do id, ustawPredkosc() i zatrzymaj().

Oto graf wywołań dla tej funkcji:



4.15.3 Dokumentacja funkcji składowych

4.15.3.1 virtual int Silnik::ustawPredkosc (float *predkosc*) [inline], [virtual]

wirtualna metoda bazowa ustawiająca zadaną prędkość dla silnika w zależności od nr ID silnika.

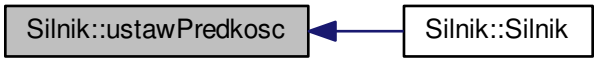
Definicja w linii 55 pliku Silnik.cpp.

Odwołuje się do mnoznik_regulacji i wyswietl().

Odwołania w Silnik().

Oto graf wywołań dla tej funkcji:

Oto graf wywoływań tej funkcji:



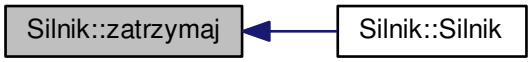
4.15.3.2 virtual int Silnik::zatrzymaj ( ) [inline],[virtual]

wirtualna metoda bazowa zatrzumująca pracę silnika

Definicja w linii 47 pliku Silnik.cpp.

Odwołania w Silnik().

Oto graf wywoływań tej funkcji:



4.15.4 Dokumentacja atrybutów składowych

4.15.4.1 float Silnik::aktualna_predkosc

Definicja w linii 28 pliku Silnik.cpp.

4.15.4.2 int Silnik::id

Definicja w linii 35 pliku Silnik.cpp.

Odwołania w Silnik().

4.15.4.3 float Silnik::mnoznik_regulacji

Definicja w linii 27 pliku Silnik.cpp.

Odwołania w ustawPredkosc().

4.15.4.4 int Silnik::pin_silnika0

Definicja w linii 30 pliku Silnik.cpp.

4.15.4.5 int Silnik::pin_silnika1

Definicja w linii 31 pliku Silnik.cpp.

4.15.4.6 int Silnik::pin_silnika2

Definicja w linii 32 pliku Silnik.cpp.

4.15.4.7 DualMC33926MotorShield* Silnik::sterownik

Definicja w linii 34 pliku Silnik.cpp.

4.15.4.8 const char Silnik::TRYB_ZATRZYMANIA_AWARYJNY = 1 [static]

Definicja w linii 25 pliku Silnik.cpp.

4.15.4.9 const char Silnik::TRYB_ZATRZYMANIA_NORMALNY = 0 [static]

Definicja w linii 24 pliku Silnik.cpp.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- Silnik.cpp

4.16 Dokumentacja klasy SilnikRegulowany

Klasa dzidzicząca po klasie Naped. Pozwala na wybór i ustawienie parametrów dla obiektów typu Naped.

Diagram dziedziczenia dla SilnikRegulowany

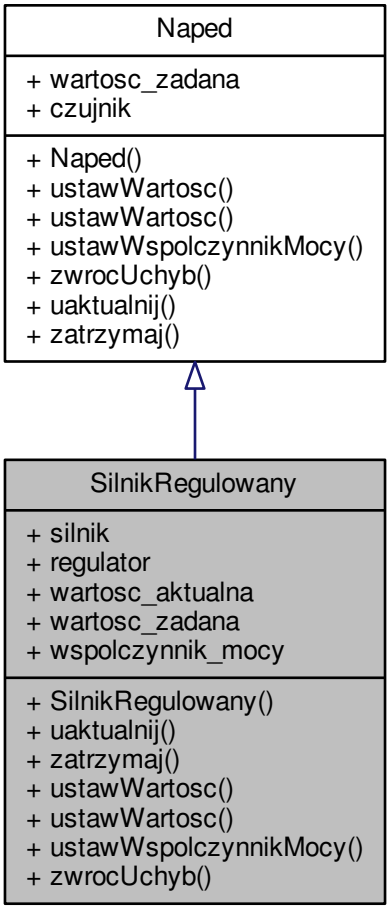
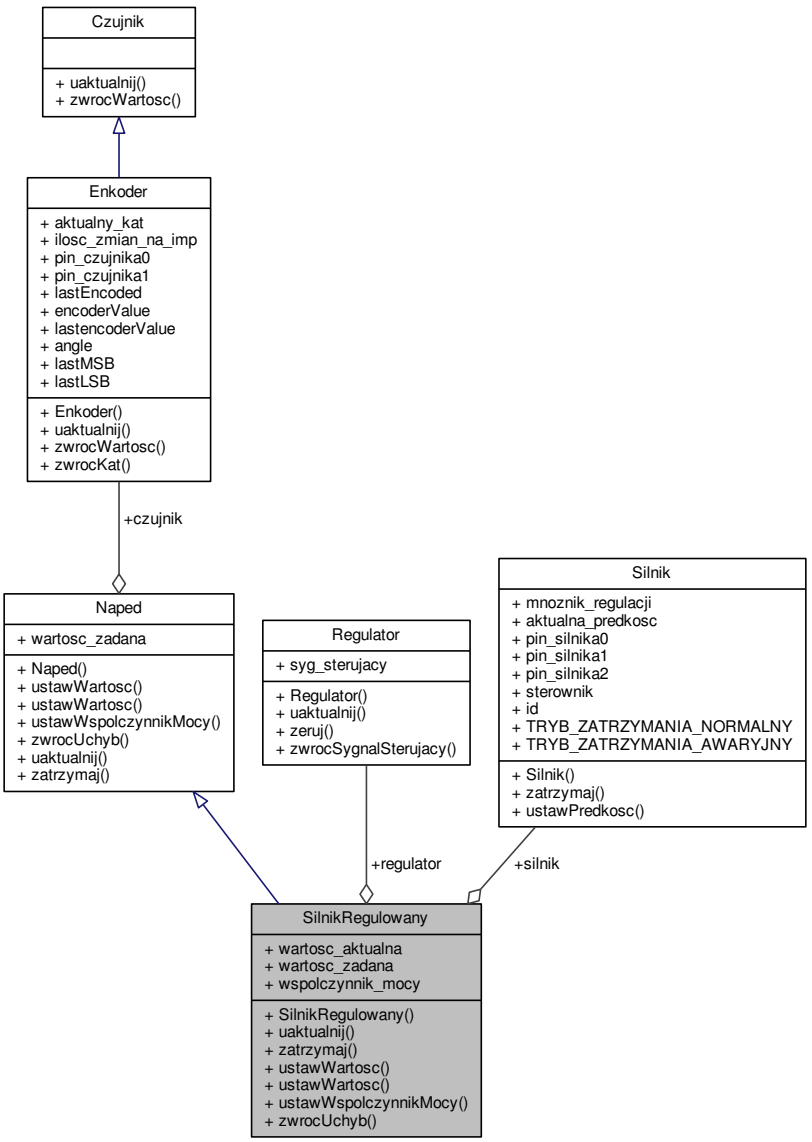


Diagram współpracy dla SilnikRegulowany:



Metody publiczne

- SilnikRegulowany (Regulator ∗reg, Enkoder ∗czu, Silnik ∗sil)

  Konstruktor przypisujący wartości otrzymanych parametrów do zmiennych lokalnych.
- int uaktualnij ()

  metoda odwołująca się do metod uaktualnij obiektu silnik, czujnik i regulator.
- int zatrzymaj ()

  metoda zatrzymująca pracę silnika.
- int ustawWartosc (float a)
- int ustawWartosc (float kat, float wspolmocy)
- int ustawWspolczynnikMocy (float wspol_mocy)

  Metoda zwracająca współczynnik mocy.
- float zwrocUchyb ()

  metoda liczy i zwraca wartość uchybu.

Atrybuty publiczne

- Silnik * silnik
- Regulator * regulator
- float wartosc_aktualna = 0.0
- float wartosc_zadana = 0.0
- float wspolczynnik_mocy = 1.0

4.16.1 Opis szczegółowy

Klasa dzidzicząca po klasie Naped. Pozwala na wybór i ustawienie parametrów dla obiektów typu Naped.

Definicja w linii 95 pliku Naped.cpp.

4.16.2 Dokumentacja konstruktora i destruktora

4.16.2.1 SilnikRegulowany::SilnikRegulowany (**Regulator** * *reg,* **Enkoder** * *czu,* **Silnik** * *sil*) [inline]

Konstruktor przypisujący wartości otrzymanych parametrów do zmiennych lokalnych.

Definicja w linii 106 pliku Naped.cpp.

Odwołuje się do Naped::czujnik i sil.

4.16.3 Dokumentacja funkcji składowych

4.16.3.1 int SilnikRegulowany::uaktualnij ( ) [inline], [virtual]

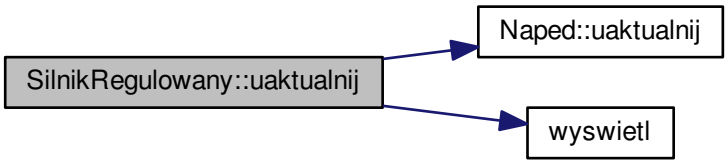
metoda odwołująca się do metod uaktualnij obiektu silnik, czujnik i regulator.

Implementuje Naped.

Definicja w linii 113 pliku Naped.cpp.

Odwołuje się do Naped::czujnik, DEBUG_MODE, Naped::uaktualnij() i wyswietl().

Oto graf wywołań dla tej funkcji:



4.16.3.2 int SilnikRegulowany::ustawWartosc (float *a*) [inline], [virtual]

Implementuje Naped.

Definicja w linii 136 pliku Naped.cpp.

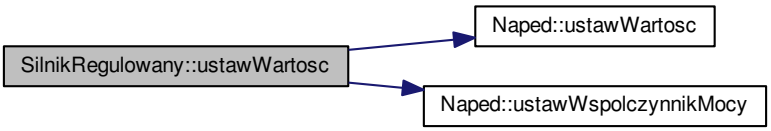
4.16.3.3 int SilnikRegulowany::ustawWartosc (float *kat*, float *wspolmocy*) [inline], [virtual]

Implementuje Naped.

Definicja w linii 143 pliku Naped.cpp.

Odwołuje się do Naped::ustawWartosc() i Naped::ustawWspolczynnikMocy().

Oto graf wywołań dla tej funkcji:



4.16.3.4 int SilnikRegulowany::ustawWspolczynnikMocy (float *wspol_mocy*) [inline], [virtual]

Metoda zwracająca współczynnik mocy.

Implementuje Naped.

Definicja w linii 149 pliku Naped.cpp.

4.16.3.5 int SilnikRegulowany::zatrzymaj ( ) [inline], [virtual]

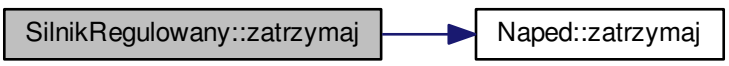
metoda zatrzymująca pracę silnika.

Implementuje Naped.

Definicja w linii 128 pliku Naped.cpp.

Odwołuje się do Naped::zatrzymaj().

Oto graf wywołań dla tej funkcji:



4.16.3.6 float SilnikRegulowany::zwrocUchyb ( ) [inline], [virtual]

metoda liczy i zwraca wartość uchybu.

Implementuje Naped.

Definicja w linii 154 pliku Naped.cpp.

4.16.4 Dokumentacja atrybutów składowych

4.16.4.1 Regulator* SilnikRegulowany::regulator

Definicja w linii 100 pliku Naped.cpp.

4.16.4.2 Silnik* SilnikRegulowany::silnik

Definicja w linii 98 pliku Naped.cpp.

4.16.4.3 float SilnikRegulowany::wartosc_aktualna = 0.0

Definicja w linii 102 pliku Naped.cpp.

4.16.4.4 float SilnikRegulowany::wartosc_zadana = 0.0

Definicja w linii 103 pliku Naped.cpp.

4.16.4.5 float SilnikRegulowany::wspolczynnik_mocy = 1.0

Definicja w linii 104 pliku Naped.cpp.

Dokumentacja dla tej klasy została wygenerowana z pliku:

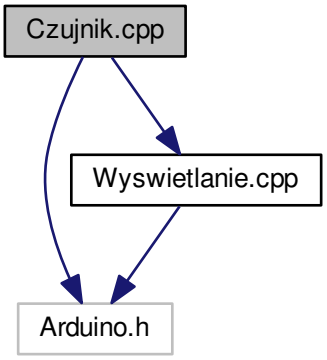
- Naped.cpp

5 Dokumentacja plików

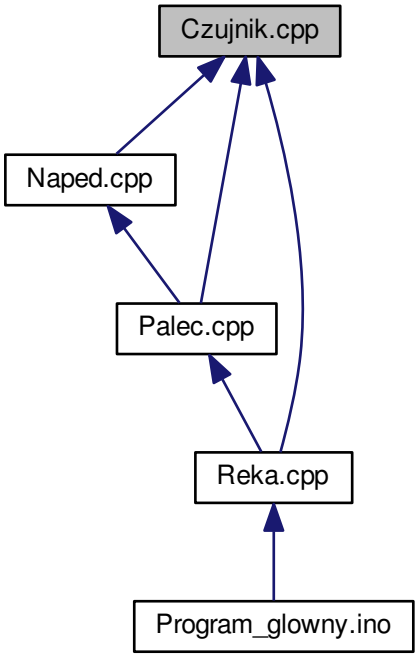
5.1 Dokumentacja pliku Czujnik.cpp

```
#include "Arduino.h"
#include "Wyswietlanie.cpp"
```

Wykres zależności załączania dla Czujnik.cpp:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class Czujnik

  Klasa abstrakcyjna Czujnik. Klasa zawierająca wirtualne metody które przy wywoływaniu nadpisywane są przez metody klasy dziedziczącej.
- class CzujnikNacisku

  Klasa ta zawiera metody liczące siłę nacisku i zwracające wartość tego nacisku do programu głównego.
- class CzujnikPradu

  Klasa CzujnikPradu dziedziczy metody uaktualnij() i zwrocWartosc() z abstrakcyjnej klasy Czujnik.
- class Enkoder

  Klasa Enkoder definiuje zachowanie Enkodera inkrementalnego, pozwala na podstawie ilości impulsów i sekwencji zmian na impuls określić kąt obrotu.

Definicje

- #define Czujnik_H

5.1.1 Opis szczegółowy

Autor

Violetta Munar Ernandes

Wersja

1.0

Definicja w pliku Czujnik.cpp.

5.1.2 Dokumentacja definicji

5.1.2.1 #define Czujnik_H

Definicja w linii 8 pliku Czujnik.cpp.

5.2 Czujnik.cpp

```
00001
00007 #ifndef Czujnik_H
00008 #define Czujnik_H
00009
00010 #include "Arduino.h"
00011
00012 #include "Wyswietlanie.cpp"
00013
00014
00018 class Czujnik {
00019   public:
00020
00021     virtual int   uaktualnij()    = 0;
00022     virtual float zwrocWartosc()  = 0;
00023 };
00024
00025
00028 class CzujnikNacisku : public Czujnik {
00029   private:
```

```
00030
00031     int uaktualnij() {
00032       if (DEBUG_MODE) Serial.println("CzujnikNacisku.uaktualnij()");
00033
00034       int naciskADC = analogRead(FSR_Pin);
00035       FSR_Voltage = (naciskADC/1023.0)* 3.3;
00036 //      Serial.print("Voltage reading in V = ");
00037 //      Serial.println(FSR_Voltage);
00038       fsrResistance = 3.3 - FSR_Voltage;
00039       fsrResistance *= 1.95;                // 1.95KOM resistor
00040       fsrResistance /= FSR_Voltage;
00041 //      Serial.print("FSR resistance in Kohms = ");
00042 //      Serial.println(fsrResistance);
00043
00044       FSR_Force = pow(fsrResistance, -1.1637385058);
00045       FSR_Force*= 1369.8967136477;
00046       FSR_Force*=9.81;
00047       FSR_Force /= 1000;
00048 //      Serial.print("Force in Newtons: ");
00049 //      Serial.println(FSR_Force);
00050
00051       nacisk_N = FSR_Force;
00052
00053       return 0;
00054     }
00055
00056
00057
00058   public:
00059     float nacisk_N;
00060     float nacisk_max_n;
00061     int FSR_Pin;
00062     float fsrResistance;
00063     float FSR_Voltage;
00064     float FSR_Force;
00065
00069     CzujnikNacisku(int pin_czuj) {
00070       FSR_Pin = pin_czuj;
00071       pinMode(FSR_Pin, INPUT);
00072       nacisk_N = 0.0;
00073       nacisk_max_n = 10.0;
00074     }
00075
00076       float zwrocWartosc() {
00077       return this -> zwrocNaciskN();
00078     }
00079
00084     float zwrocNaciskN() {
00085       uaktualnij();
00086
00087       return nacisk_N;
00088     }
00089
00094 };
00095
00098 class CzujnikPradu : public Czujnik {
00099   private:
00104   int uaktualnij() {
00105
00106       if (DEBUG_MODE) wyswietl("CzujnikPradu.uaktualnij()", AmpsRMS);
00107
00108       readValue = analogRead(sensorIn);
00109       Voltage = readValue/1023.0*3.3;
00110       AmpsRMS = (15.7895*Voltage)-39.47368;
00111
00112      return 0;
00113     }
00114
00115   public:
00116
00117     int sensorIn = A7;
00118     //int mVperAmp = 66; // use 100 for 20A Module and 185 for 5 Module
00119     double Voltage = 0;
00120     double AmpsRMS = 0;
00121     float readValue = 0;
00122
00126     CzujnikPradu(int pin_czujnika) {
00127       this -> sensorIn = pin_czujnika;
00128       this -> AmpsRMS = 0.0;
00129       pinMode(sensorIn, INPUT);
00130     }
00131
00134     float zwrocWartosc() {
00135       return this -> zwrocNatezenieAmp();
00136     }
00137
00141     float zwrocNatezenieAmp() {
```

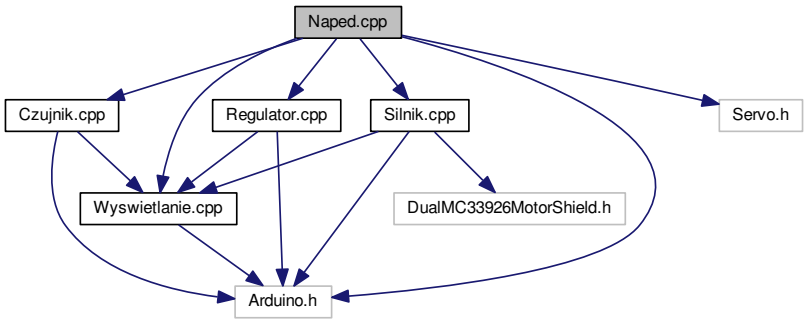
```
00142       uaktualnij();
00143
00144       if (DEBUG_MODE) wyswietl("CzujnikPradu.zwrocNatezenieAmp()", AmpsRMS);
00145
00146       Serial.print("\nCzujnikPradu: [A]: ");Serial.print(AmpsRMS);Serial.print("; [V]: ");Serial.println(
      Voltage);
00147
00148       return AmpsRMS;
00149     }
00150 };
00151
00156 class Enkoder : public Czujnik {
00157   public:
00158     float aktualny_kat;
00159     int ilosc_zmian_na_imp;
00160     int pin_czujnika0;
00161     int pin_czujnika1;
00162     volatile int lastEncoded = 0;
00163     volatile long encoderValue = 0;
00164     long lastencoderValue = 0;
00165     volatile float angle = 0;
00166
00167     int lastMSB = 0;
00168     int lastLSB = 0;
00169
00177     Enkoder(int pin0, int pin1, int il_zm_na_imp) {
00178
00179       this -> pin_czujnika0 = pin0;
00180       this -> pin_czujnika1 = pin1;
00181       this -> ilosc_zmian_na_imp = il_zm_na_imp;
00182
00183       pinMode(pin_czujnika0, INPUT);
00184       pinMode(pin_czujnika1, INPUT);
00185
00186       digitalWrite(pin_czujnika0, HIGH); // ustaiwnie pinu na stan wysoki
00187       digitalWrite(pin_czujnika1, HIGH); // ustaiwnie pinu na stan wysoki
00188
00189       this -> aktualny_kat = 0.0;
00190     }
00191
00196     int uaktualnij() {  // to ruszamy, wartosc kata przypisujemy do zmiennej 'aktualny_kat'
00197
00198       int MSB = digitalRead(pin_czujnika0); //MSB = most significant bit
00199       int LSB = digitalRead(pin_czujnika1); //LSB = least significant bit
00200
00201       int encoded = (MSB << 1) |LSB; //converting the 2 pin value to single number
00202       int sum  = (lastEncoded << 2) | encoded; //adding it to the previous encoded value
00203
00204       if(sum == 0b1101 || sum == 0b0100 || sum == 0b0010 || sum == 0b1011) encoderValue--;
00205       if(sum == 0b1110 || sum == 0b0111 || sum == 0b0001 || sum == 0b1000) encoderValue++;
00206
00207       // KAT ZAWIERA SIE W PRZEDZIALE (-180 - 180)
00208
00209       //---------DLA DUZEGO ENKODERA--------------------
00210
00211       if (encoderValue>ilosc_zmian_na_imp*12/2) encoderValue -= ilosc_zmian_na_imp*12; // 2*12 changes ->
      180*, 4*12ch -> 360*
00212       if (encoderValue<-ilosc_zmian_na_imp*12/2) encoderValue += ilosc_zmian_na_imp*12; // 2*12 changes ->
      180*, 4*12ch -> 360*
00213       angle = encoderValue/ilosc_zmian_na_imp/12.0*360; // dobra dzialajaca wersja
00214
00215       //---------DLA MNIEJSZEGO ENKODERA--------------------
00216       //  if (encoderValue>2*12) encoderValue -= 4*12; // 2*12 changes -> 180*, 4*12ch -> 360*
00217       //  if (encoderValue<-2*12) encoderValue += 4*12; // 2*12 changes -> 180*, 4*12ch -> 360*
00218       //  angle = encoderValue/4.0/12.0*360;
00219
00220       lastEncoded = encoded; //store this value for next time
00221       aktualny_kat = angle;
00222
00223       if (DEBUG_MODE) wyswietl("Enkoder.uaktualnij()", aktualny_kat);
00224
00225       return 0;
00226     }
00227
00228     //Metoda uaktualniająca zmienne wewnętrzne obiektu tej klasy.
00229
00234     float zwrocWartosc() {
00235      return this -> zwrocKat();
00236     }
00237
00242     float zwrocKat() {
00243       uaktualnij();
00244
00245       return aktualny_kat;
00246     }
00247 };
00248
```

```
00249 #endif
```

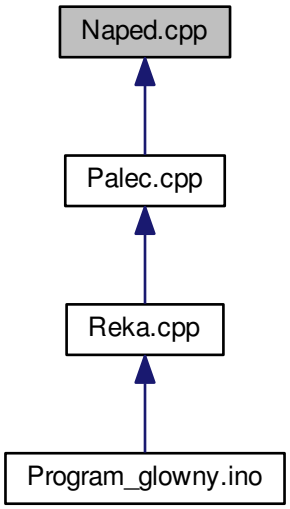
5.3 Dokumentacja pliku Naped.cpp

```
#include "Arduino.h"
#include "Wyswietlanie.cpp"
#include "Silnik.cpp"
#include "Czujnik.cpp"
#include "Regulator.cpp"
#include <Servo.h>
```

Wykres zależności załączania dla Naped.cpp:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class Naped

  Klasa Naped zawiera metody wirtualne które nadpisywane są przez klasy dziedziczące.
- class Serwo

  Klasa Serwo dziedziczy po klasie Naped. Zawiera w sobie obiekt klasy Servo i metody obsługującej stwerowanie serwomechanizmem.
- class SilnikRegulowany

  Klasa dzidziczą po klasie Naped. Pozwala na wybór i ustawienie parametrów dla obiektów typu Naped.

Definicje

- #define Naped_H
- #define DEBUG_MODE false

5.3.1 Opis szczegółowy

Autor

Violetta Munar Ernandes

Wersja

1.0

Definicja w pliku Naped.cpp.

5.3.2 Dokumentacja definicji

5.3.2.1 #define DEBUG_MODE false

Definicja w linii 10 pliku Naped.cpp.

Odwołania w CzujnikNacisku::uaktualnij(), CzujnikPradu::uaktualnij(), SilnikRegulowany::uaktualnij(), Enkoder↩ ::uaktualnij() i CzujnikPradu::zwrocNatezenieAmp().

5.3.2.2 #define Naped_H

Definicja w linii 8 pliku Naped.cpp.

5.4 Naped.cpp

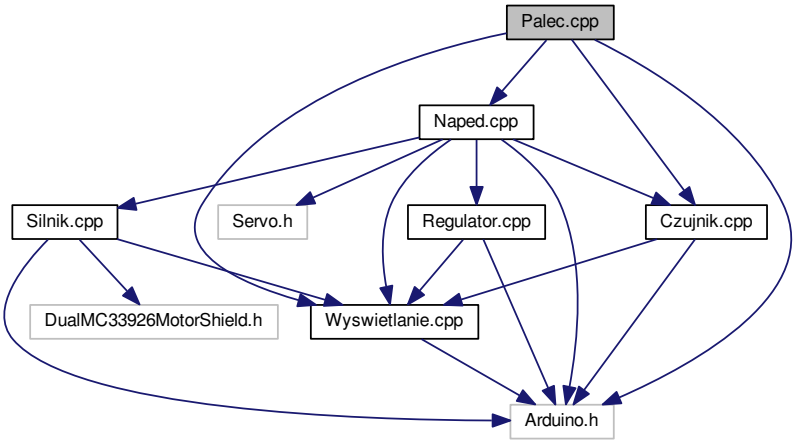
```
00001
00007 #ifndef Naped_H
00008 #define Naped_H
00009
00010 #define DEBUG_MODE false
00011
00012 #include "Arduino.h"
00013
00014 #include "Wyswietlanie.cpp"
00015 #include "Silnik.cpp"
00016 #include "Czujnik.cpp"
00017 #include "Regulator.cpp"
00018 #include <Servo.h>
00019
00025 class Naped {
00026   public:
00027     float wartosc_zadana = 0.0;
00028     Enkoder* czujnik; // obiekt typu Czujnik .
00029
00030     Naped () {
00031
00032     }
00033     virtual int ustawWartosc(float wartosc) = 0;
00034
00035     virtual int ustawWartosc(float wartosc, float wspolczynnik_mocy) = 0;
00036
00037     virtual int ustawWspolczynnikMocy(float wspol_mocy) = 0;
00038
00039     virtual float zwrocUchyb() = 0;
00040
00041     virtual int uaktualnij() = 0;
00042
00043     virtual int zatrzymaj() = 0;
00044 };
00045
00049 class Serwo : public Naped {
00050   public:
00051     int pin;
00052     Servo servo;
00053
00054     Serwo(int pin) {
00055       this -> pin = pin;
00056       servo.attach(this -> pin);
00057
00058     }
00059
00060     float zwrocUchyb() {
00061       return 0.0;
00062     }
00063
00064     int uaktualnij() {
00065
00066       return 0;
00067     }
00068
00069     int zatrzymaj() {
00070       Serial.println("TODO: Serwo.zatrzymaj()");
00071
00072       return 0;
00073     }
00074
00075     int ustawWartosc(float kat) {
00076       wartosc_zadana = kat;
00077       wyswietl("Kat serwo", wartosc_zadana);
00078       servo.write((int) wartosc_zadana);
00079
00080       return 0;
00081     }
00082
00083     int ustawWartosc(float kat, float wspolmocy) {
00084       ustawWartosc(kat);
00085     }
00086
00087      int ustawWspolczynnikMocy(float wspol_mocy) {
00088       return 0;
00089      }
00090
00091 };
00095 class SilnikRegulowany : public Naped {
00096   public:
00097
00098     Silnik* silnik;
00099 //    Czujnik*/ czujnik;
00100     Regulator* regulator;
```

```
00101
00102     float wartosc_aktualna = 0.0;
00103     float wartosc_zadana = 0.0;
00104     float wspolczynnik_mocy = 1.0;
00105
00106     SilnikRegulowany(Regulator* reg, Enkoder* czu,
      Silnik* sil) {
00107       wartosc_zadana = 0.0;
00108       silnik = sil;
00109       czujnik = czu;
00110       regulator = reg;
00111     }
00112
00113     int uaktualnij() {
00114       if (DEBUG_MODE) Serial.println("SilnikRegulowany.uaktualnij()");
00115
00116       wartosc_aktualna = czujnik -> zwrocWartosc();
00117       regulator -> uaktualnij(wartosc_zadana - wartosc_aktualna);
00118
00119       silnik -> ustawPredkosc(regulator -> zwrocSygnalSterujacy()*wspolczynnik_mocy);
00120       wyswietl("Enkoder: ", czujnik -> zwrocWartosc());
00121
00122       //Serial.println(napedy[i] -> czujnik -> zwrocWartosc());
00123       //  Serial.print("Reg U = "); Serial.print(i+1); Serial.print(": "); Serial.println(napedy[i] ->
       zwrocUchyb());
00124
00125       return 0;
00126     }
00127
00128     int zatrzymaj() {
00129       Serial.println("TODO: SilnikRegulowany.zatrzymaj()");
00130
00131       silnik -> zatrzymaj();
00132
00133       return 0;
00134     }
00135
00136     int ustawWartosc(float a) {
00137
00138       wartosc_zadana = a;
00139
00140       return 0;
00141     }
00142
00143     int ustawWartosc(float kat, float wspolmocy) {
00144       ustawWartosc(kat);
00145       ustawWspolczynnikMocy(wspolmocy);
00146       return 0;
00147     }
00148
00149      int ustawWspolczynnikMocy(float wspol_mocy) {
00150       wspolczynnik_mocy = wspol_mocy;
00151       return 0;
00152      }
00153
00154     float zwrocUchyb() {
00155       return wartosc_zadana - wartosc_aktualna;
00156     }
00157
00161 };
00162
00163
00164 #endif
```

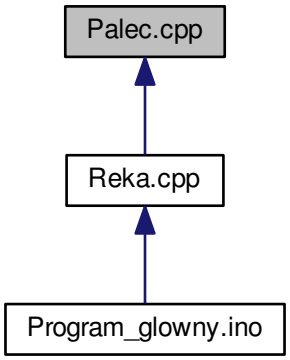
5.5 Dokumentacja pliku Palec.cpp

```
#include "Arduino.h"
#include "Wyswietlanie.cpp"
#include "Czujnik.cpp"
#include "Naped.cpp"
```

Wykres zależności załączania dla Palec.cpp:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class Palec

  Klasa Palec zawiera klasy dziedziczące PalecNorm i PalecKciuk. Klasa zaiwera metody wirtualne które nadpisywane są przez klasy dziedziczące.
- class PalecNorm

  Klasa PalecNorm Klasa zaiwera metody które realizują pracę poszczególnych palów. Metody te odnoszą sie do palców które w swojej budowie zawierają dwa silniki.
- class PalecKciuk

  Klasa PalecKciuk Klasa zaiwera metody które realizują pracę kciuka. Metody te odnoszą sie do palca który w swojej budowie zawiera dwa silniki i serwomechanizm.

Definicje

- #define Palec_H
- #define DEBUG_MODE false

5.5.1 Opis szczegółowy

Autor

Violetta Munar Ernandes

Wersja

1.0

Definicja w pliku Palec.cpp.

5.5.2 Dokumentacja definicji

5.5.2.1 #define DEBUG_MODE false

Definicja w linii 10 pliku Palec.cpp.

Odwołania w PalecKciuk::uaktualnij(), PalecNorm::wyprost(), PalecNorm::zatrzymaj(), Palec::zegnijKat(), Palec↩ Norm::zegnijKat() i PalecKciuk::zegnijKat().

5.5.2.2 #define Palec_H

Definicja w linii 8 pliku Palec.cpp.

5.6 Palec.cpp

```
00001
00007 #ifndef Palec_H
00008 #define Palec_H
00009
00010 #define DEBUG_MODE false
00011
00012 #include "Arduino.h"
00013
00014 #include "Wyswietlanie.cpp"
00015
00016 #include "Czujnik.cpp"
00017 #include "Naped.cpp"
00018
00022 class Palec {
00023   public:
00024     char nr_palca;
00025
00026     float dokladnosc_ruchu_deg = 15;
00027     float maksymalny_nacisk_N = 100;
00028
00029     CzujnikNacisku* czujnik_nacisku;
00030     SilnikRegulowany* napedy[2];
00031
00032     // konstruktor palec
00033     Palec() {
00034       nr_palca = 0;
00035     }
00036
00037     virtual int uaktualnij() {
```

```
00038       wyswietl("Nacisk [N]: ", zwrocNacisk());
00039
00040       for (int i=0; i<2; i++) {
00041         Serial.print("    NAPED "); Serial.println(i);
00042         napedy[i] -> uaktualnij();
00043
00044         float wspolczynnik_mocy = zwrocWspolczynnikMocy();
00045
00046         napedy[i] -> ustawWspolczynnikMocy(wspolczynnik_mocy);
00047       }
00048       return 0;
00049     }
00050
00051
00052     virtual int zatrzymaj() = 0;
00053
00054     virtual int zegnijKat(float c, float a, float b, float maxnacisk) {
00055       if (DEBUG_MODE) Serial.println("Palec: redirecting from zegnijKat(3zm) -> zegnijKat(2zm)");
00056       zegnijKat(a, b, maxnacisk);
00057
00058       return 0;
00059     }
00060
00061     virtual int zegnijKat(float a, float b, float maxnacisk) {
00062       if (DEBUG_MODE) Serial.println("Palec: redirecting from zegnijKat(2zm) -> zegnijKat(3zm)");
00063       zegnijKat(0.0, a, b, maxnacisk);
00064
00065       return 0;
00066     }
00067
00068     virtual int wyprost() = 0;
00069
00070     float zwrocNacisk() {
00071
00072       float nacisk = czujnik_nacisku->zwrocWartosc();
00073
00074       return nacisk;
00075     }
00076
00078     float zwrocWspolczynnikMocy() {
00079       float nacisk = this -> zwrocNacisk();
00080       float wspolczynnik_mocy = (maksymalny_nacisk_N - nacisk)/maksymalny_nacisk_N;
00081
00082       if (wspolczynnik_mocy > 0)
00083         return wspolczynnik_mocy;
00084       else
00085         return wspolczynnik_mocy*0.1;
00086     }
00087
00088
00089     bool czyRuchZakonczony() {
00090       for (int i=0; i<2; i++) {
00091         float uchyb = napedy[i] -> zwrocUchyb();
00092         if ((uchyb < -dokladnosc_ruchu_deg) || (uchyb > dokladnosc_ruchu_deg)) {
00093           return false;
00094         }
00095       }
00096       return true;
00097     }
00098 };
00099
00105 class PalecNorm : public Palec {
00106   public:
00107
00113     PalecNorm(char nr_palca, SilnikRegulowany*
      napedy[], CzujnikNacisku* czuj_nac) {
00114       this -> nr_palca = nr_palca;
00115       this -> napedy[0] = napedy[0];
00116       this -> napedy[1] = napedy[1];
00117       this -> czujnik_nacisku = czuj_nac;
00118     }
00119
00120
00121     int uaktualnij() {
00122 //      if (DEBUG_MODE) wyswietl("\n\nPalecNorm.uaktualnij()", nr_palca);
00123
00124       wyswietl("Nacisk [N]: ", zwrocNacisk());
00125
00126       for (int i=0; i<2; i++) {
00127         Serial.print("    NAPED "); Serial.println(i);
00128         napedy[i] -> uaktualnij();
00129
00130         float wspolczynnik_mocy = zwrocWspolczynnikMocy();
00131
00132         napedy[i] -> ustawWspolczynnikMocy(wspolczynnik_mocy);
00133       }
00134
```

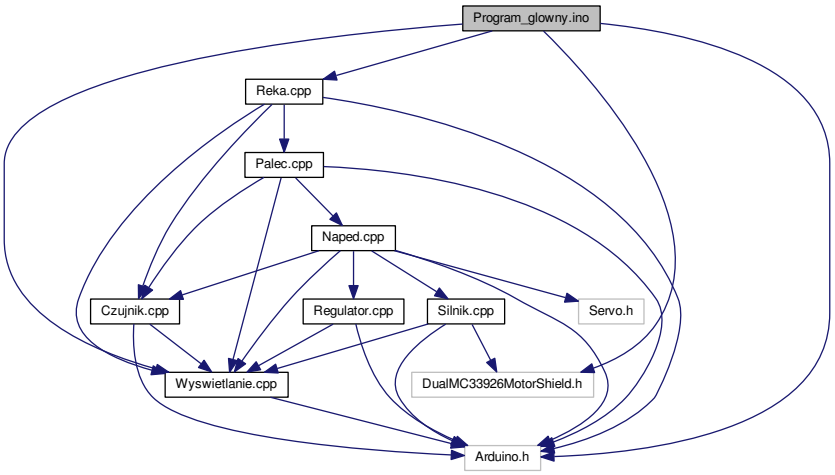
```
00135       return 0;
00136     }
00137
00138     int zatrzymaj() {
00139       if (DEBUG_MODE) wyswietl("\n\nPalecNorm.zatrzymaj()",
      nr_palca);
00140
00141       for (int i=0; i<2; i++) {
00142         napedy[i] -> zatrzymaj();
00143       }
00144
00145       return 0;
00146     }
00147
00148     int zegnijKat(float a, float b, float maxnacisk) {
00149       if (DEBUG_MODE) wyswietl("\n\nPalecNorm.zegnijKat(3zm)",
      nr_palca);
00150
00151       maksymalny_nacisk_N = maxnacisk;
00152       float wspolczynnik_mocy = zwrocWspolczynnikMocy();
00153
00154       napedy[0] -> ustawWartosc(a, wspolczynnik_mocy);
00155       napedy[1] -> ustawWartosc(b, wspolczynnik_mocy);
00156
00157       return 0;
00158     }
00159
00160     int wyprost() {
00161       if (DEBUG_MODE) wyswietl("\n\nPalecNorm.wyprost()",
      nr_palca);
00162
00163       zegnijKat(0.0, 0.0, 10);
00164
00165       return 0;
00166     }
00167 };
00168
00174 class PalecKciuk : public Palec {
00175   public:
00176     Serwo* serwo;
00177
00178     PalecKciuk(char nr_palca, SilnikRegulowany*
      napedy[], Serwo* serwo, CzujnikNacisku* czuj_nac) {
00179       this -> nr_palca = nr_palca;
00180       this -> napedy[0] = napedy[0];
00181       this -> napedy[1] = napedy[1];
00182       this -> serwo = serwo;
00183       this -> czujnik_nacisku = czuj_nac;
00184     }
00185
00192     int uaktualnij() {
00193       if (DEBUG_MODE) wyswietl("\n\nPalecKciuk.uaktualnij()",
      nr_palca);
00194
00195       wyswietl("Nacisk [N]: ", zwrocNacisk());
00196
00197       for (int i=0; i<2; i++) {
00198         Serial.print("    NAPED "); Serial.println(i);
00199         napedy[i] -> uaktualnij();
00200
00201         float wspolczynnik_mocy = zwrocWspolczynnikMocy();
00202
00203         napedy[i] -> ustawWspolczynnikMocy(wspolczynnik_mocy);
00204       }
00205
00206       return 0;
00207     }
00208
00209     int zatrzymaj() {
00210       wyswietl("\n\nPalecKciuk.zatrzymaj()", nr_palca);
00211
00212       for (int i=0; i<2; i++) {
00213         napedy[i] -> zatrzymaj();
00214       }
00215
00216       serwo -> zatrzymaj();
00217
00218       return 0;
00219     }
00220
00221     int zegnijKat(float c, float a, float b, float maxnacisk) {
00222       if (DEBUG_MODE) wyswietl("\n\nPalecKciuk.zegnijKat(3zm)",
      nr_palca);
00223
00224       maksymalny_nacisk_N = maxnacisk;
00225       float wspolczynnik_mocy = zwrocWspolczynnikMocy();
00226
```

```
00227       napedy[0] -> ustawWartosc(a, wspolczynnik_mocy);
00228       napedy[1] -> ustawWartosc(b, wspolczynnik_mocy);
00229       serwo -> ustawWartosc(c);
00230
00231       return 0;
00232     }
00233
00234
00235     int wyprost() {
00236       wyswietl("\n\nPalecKciuk.wyprost()", nr_palca);
00237
00238       zegnijKat(0.0, 0.0, 0.0, 10);
00239
00240       return 0;
00241     }
00242 };
00243
00244 #endif
```

5.7 Dokumentacja pliku Program_glowny.ino

```
#include <Arduino.h>
#include "Wyswietlanie.cpp"
#include "Reka.cpp"
#include "DualMC33926MotorShield.h"
```

Wykres zależności załączania dla Program_glowny.ino:



Definicje

- #define DEBUG_MODE false

Funkcje

- void setup ()
- void loop ()

  Pętla główna programu. Znajduje się tu cała logika działania programu.
- void updateEnkoder00 ()

  Funkcja nieprzyjmująca parametrow wywołuję funkcję updateEnkoder() o dwóch parametrach.
- void updateEnkoder01 ()

- void updateEnkoder10 ()
- void updateEnkoder11 ()
- void updateEnkoder20 ()
- void updateEnkoder21 ()
- void updateEnkoder30 ()
- void updateEnkoder31 ()
- void updateEnkoder40 ()
- void updateEnkoder41 ()
- void updateEnkoder (int palid, int enkid)

  funkcja realizująca odwolanie do metody uaktualnij

Zmienne

- CzujnikNacisku * czuj_nac [5]

  tablica wskaźników na obiekty typu Czujnik reprezentujących czujniki nacisku.
- Enkoder * enk [5][2]

  tablica wskaźników na obiekty typu Czujnik reprezentujących enkodery.
- CzujnikPradu * czuj_prad

  tablica wskaźników na obiekty typu Czujnik reprezentujących czujnik prądu.
- Silnik * sil [5][2]

  tablica wskaźników na obiekty typu Silnik reprezentujących silniki DC.
- SilnikRegulowany * napedy [5][2]

  tablica wskaźników na obiekty typu Naped reprezentujących napędy regulowane.
- Serwo * serwo_kciuk

  wskaźnik do obiektu typu Naped reprezentujących serwomechanizm.
- DualMC33926MotorShield * sterowniki [5]

  deklaracja tablicy wskaźników na obiekty typu DualMC33926MotorShield
- Palec * pal [5]

  tablica wskaźników na obiekty typu Palec reprezentujących palce ręki.
- Reka * reka

  wskaźnik do obiektu typu Reka reprezentujących cały mechanizm zawierający wszystkie elementy.
- float dt_sec = 4.0/100.0

  zmienna określająca czas próbkowania w sekundach
- float natezenie_zatrzymania = 4.0

  natezenie pradu przy ktorym reka przestanie sie zaciskac

5.7.1 Opis szczegółowy

Autor

Violetta Munar Ernandes

Wersja

1.0

Definicja w pliku Program_glowny.ino.

5.7.2 Dokumentacja definicji

5.7.2.1 #define DEBUG_MODE false

Definicja w linii 7 pliku Program_glowny.ino.

5.7.3 Dokumentacja funkcji

5.7.3.1 void loop ( )

Pętla główna programu. Znajduje się tu cała logika działania programu.

Pętla główna programu. Znajduje się tu cała logika działania programu.

Definicja w linii 151 pliku Program_glowny.ino.

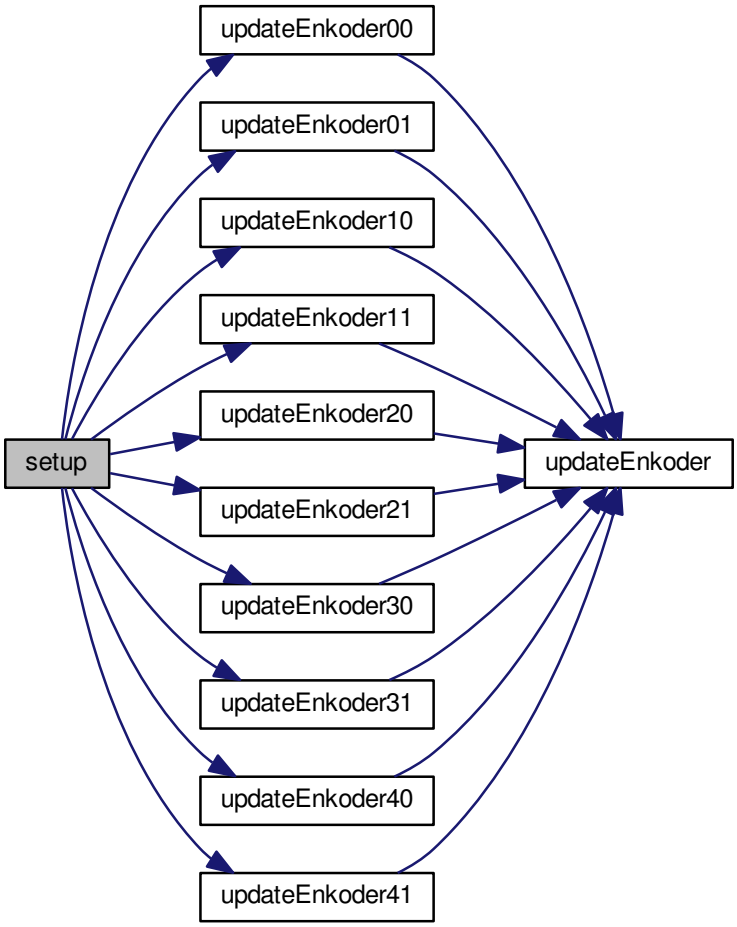
5.7.3.2 void setup ( )

Funkcja służąca do konfiguracji używanych zmiennych i obiektów. Kompletowany jest tutaj obiekt klasy Reka, składający się z wielu elementów składowych.

Definicja w linii 40 pliku Program_glowny.ino.

Odwołuje się do dt_sec, natezenie_zatrzymania, sterowniki, updateEnkoder00(), updateEnkoder01(), updateEnkoder10(), updateEnkoder11(), updateEnkoder20(), updateEnkoder21(), updateEnkoder30(), updateEnkoder31(), updateEnkoder40() i updateEnkoder41().

Oto graf wywołań dla tej funkcji:



5.7.3.3 void updateEnkoder (int *palid*, int *enkid*)

funkcja realizująca odwolanie do metody uaktualnij

Funkcja uruchamiana przez funckje void updateEnkoderXY(), które są funkcjami obsługi zdarzenia zmiany stanu na danym wyjściu cyfrowym.

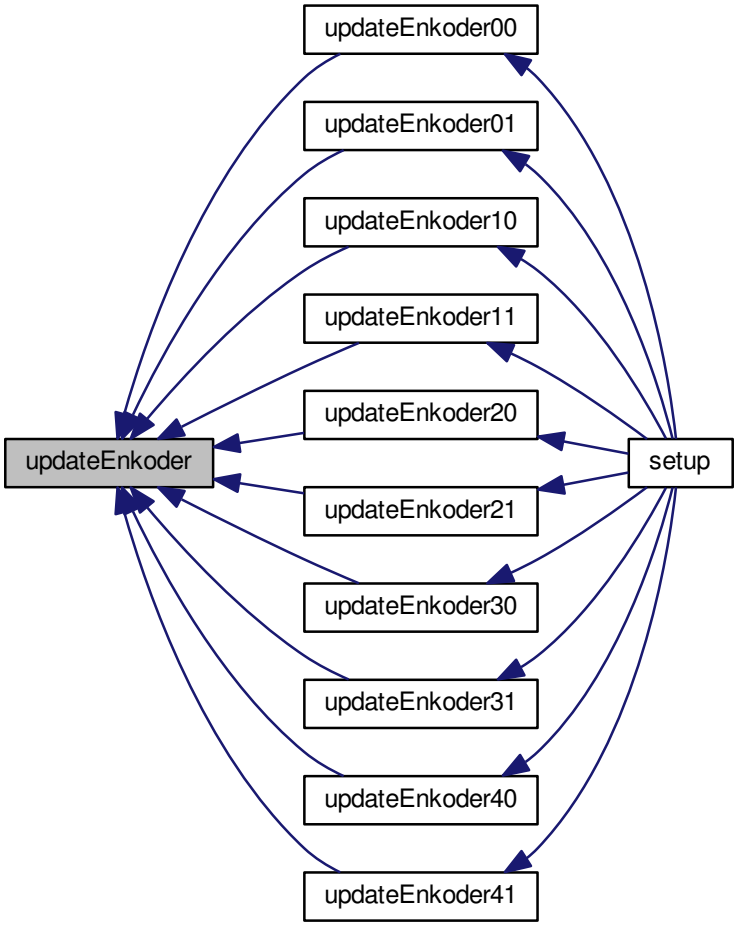
Parametry

palid	Zmienna oznaczająca numer palca.
enkid	Zmienna oznaczająca numer enkodera na danym palcu.

Definicja w linii 177 pliku Program_glowny.ino.

Odwołania w updateEnkoder00(), updateEnkoder01(), updateEnkoder10(), updateEnkoder11(), updateEnkoder20(), updateEnkoder21(), updateEnkoder30(), updateEnkoder31(), updateEnkoder40() i updateEnkoder41().

Oto graf wywoływań tej funkcji:



5.7.3.4 void updateEnkoder00 ( )

Funkcja nieprzyjmująca parametrow wywołuję funkcję updateEnkoder() o dwóch parametrach.

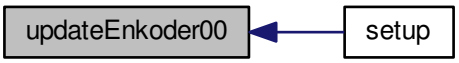
Definicja w linii 160 pliku Program_glowny.ino.

Odwołuje się do updateEnkoder().

Odwołania w setup().

Oto graf wywołań dla tej funkcji:

Oto graf wywoływań tej funkcji:



5.7.3.5 void updateEnkoder01 ( )

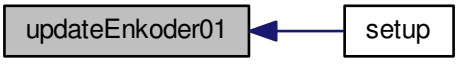
Definicja w linii 161 pliku Program_glowny.ino.

Odwołuje się do updateEnkoder().

Odwołania w setup().

Oto graf wywołań dla tej funkcji:

Oto graf wywoływań tej funkcji:



5.7.3.6 void updateEnkoder10 ( )

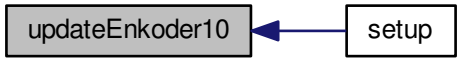
Definicja w linii 162 pliku Program_glowny.ino.

Odwołuje się do updateEnkoder().

Odwołania w setup().

Oto graf wywołań dla tej funkcji:

Oto graf wywoływań tej funkcji:



5.7.3.7 void updateEnkoder11 ( )

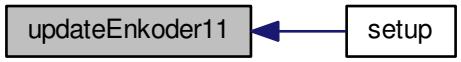
Definicja w linii 163 pliku Program_glowny.ino.

Odwołuje się do updateEnkoder().

Odwołania w setup().

Oto graf wywołań dla tej funkcji:

Oto graf wywoływań tej funkcji:



5.7.3.8 void updateEnkoder20 ( )

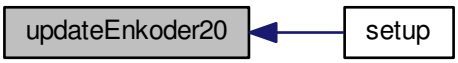
Definicja w linii 164 pliku Program_glowny.ino.

Odwołuje się do updateEnkoder().

Odwołania w setup().

Oto graf wywołań dla tej funkcji:

Oto graf wywoływań tej funkcji:



5.7.3.9 void updateEnkoder21 ( )

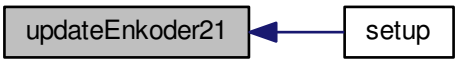
Definicja w linii 165 pliku Program_glowny.ino.

Odwołuje się do updateEnkoder().

Odwołania w setup().

Oto graf wywołań dla tej funkcji:

Oto graf wywoływań tej funkcji:



5.7.3.10 void updateEnkoder30 ( )

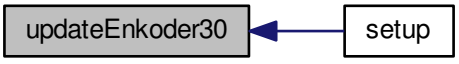
Definicja w linii 166 pliku Program_glowny.ino.

Odwołuje się do updateEnkoder().

Odwołania w setup().

Oto graf wywołań dla tej funkcji:

Oto graf wywoływań tej funkcji:



5.7.3.11 void updateEnkoder31 ( )

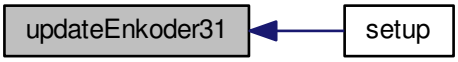
Definicja w linii 167 pliku Program_glowny.ino.

Odwołuje się do updateEnkoder().

Odwołania w setup().

Oto graf wywołań dla tej funkcji:

Oto graf wywoływań tej funkcji:



5.7.3.12 void updateEnkoder40 ( )

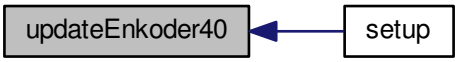
Definicja w linii 168 pliku Program_glowny.ino.

Odwołuje się do updateEnkoder().

Odwołania w setup().

Oto graf wywołań dla tej funkcji:

Oto graf wywoływań tej funkcji:



5.7.3.13 void updateEnkoder41 ( )

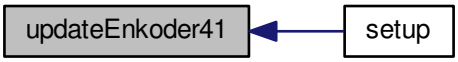
Definicja w linii 169 pliku Program_glowny.ino.

Odwołuje się do updateEnkoder().

Odwołania w setup().

Oto graf wywołań dla tej funkcji:

Oto graf wywoływań tej funkcji:



5.7.4 Dokumentacja zmiennych

5.7.4.1 CzujnikNacisku* czuj_nac[5]

tablica wskaźników na obiekty typu Czujnik reprezentujących czujniki nacisku.

Definicja w linii 16 pliku Program_glowny.ino.

Odwołania w PalecKciuk::PalecKciuk() i PalecNorm::PalecNorm().

5.7.4.2 CzujnikPradu* czuj_prad

tablica wskaźników na obiekty typu Czujnik reprezentujących czujnik prądu.

Definicja w linii 18 pliku Program_glowny.ino.

5.7.4.3 float dt_sec = 4.0/100.0

zmienna określająca czas próbkowania w sekundach

Definicja w linii 31 pliku Program_glowny.ino.

Odwołania w setup() i RegulatorPID::uaktualnij().

5.7.4.4 Enkoder* enk[5][2]

tablica wskaźników na obiekty typu Czujnik reprezentujących enkodery.

Definicja w linii 17 pliku Program_glowny.ino.

5.7.4.5 SilnikRegulowany* napedy[5][2]

tablica wskaźników na obiekty typu Naped reprezentujących napędy regulowane.

Definicja w linii 22 pliku Program_glowny.ino.

5.7.4.6 float natezenie_zatrzymania = 4.0

natezenie pradu przy ktorym reka przestanie sie zaciskac

Definicja w linii 33 pliku Program_glowny.ino.

Odwołania w setup().

5.7.4.7 Palec* pal[5]

tablica wskaźników na obiekty typu Palec reprezentujących palce ręki.

Definicja w linii 27 pliku Program_glowny.ino.

5.7.4.8 Reka* reka

wskaźnik do obiektu typu Reka reprezentujących cały mechanizm zawierający wszystkie elementy.

Definicja w linii 29 pliku Program_glowny.ino.

5.7.4.9 Serwo* serwo_kciuk

wskaźnik do obiektu typu Naped reprezentujących serwomechanizm.

Definicja w linii 23 pliku Program_glowny.ino.

5.7.4.10 Silnik* sil[5][2]

tablica wskaźników na obiekty typu Silnik reprezentujących silniki DC.

Definicja w linii 20 pliku Program_glowny.ino.

Odwołania w SilnikRegulowany::SilnikRegulowany().

5.7.4.11 DualMC33926MotorShield* sterowniki[5]

deklaracja tablicy wskaźników na obiekty typu DualMC33926MotorShield

Definicja w linii 25 pliku Program_glowny.ino.

Odwołania w setup().

5.8 Program_glowny.ino

```
00001
00007 #define DEBUG_MODE false
00008
00009 #include <Arduino.h>
00010 #include "Wyswietlanie.cpp"
00011
00012 #include "Reka.cpp"
00013 #include "DualMC33926MotorShield.h"
00014
00015
00016 CzujnikNacisku* czuj_nac[5];
00017 Enkoder* enk[5][2];
00018 CzujnikPradu* czuj_prad;
00019
00020 Silnik* sil[5][2];
00021
00022 SilnikRegulowany* napedy[5][2];
00023 Serwo* serwo_kciuk;
00024
00025 DualMC33926MotorShield* sterowniki[5];
00026
00027 Palec* pal[5];
00028
00029 Reka* reka;
00030
00031 float dt_sec = 4.0/100.0;
00032
00033 float natezenie_zatrzymania = 4.0;
00034
00040 void setup() {
00041
00042    for (int i = 2; i<=50; i++)
00043 {
00044    pinMode(i, OUTPUT);
00045    digitalWrite(i, LOW); //turn pullup resistor on
00046 }
00047
00048
00049
00050   Serial.begin(115200);
00051
00052   // tworzenie obiektow i przypisywanie do nich pinow
00053   czuj_nac[0] = new CzujnikNacisku(A0);
00054   czuj_nac[1] = new CzujnikNacisku(A1);
00055   czuj_nac[2] = new CzujnikNacisku(A2);
00056   czuj_nac[3] = new CzujnikNacisku(A3);
00057   czuj_nac[4] = new CzujnikNacisku(A4);
00058
00059   czuj_prad = new CzujnikPradu(A7);
00060
00061   enk[0][0] = new Enkoder(21, 22, 2);
00062   attachInterrupt(digitalPinToInterrupt(21), updateEnkoder00, CHANGE);
00063   attachInterrupt(digitalPinToInterrupt(22), updateEnkoder00, CHANGE);
00064
00065   enk[0][1] = new Enkoder(23, 24, 2);
00066   attachInterrupt(digitalPinToInterrupt(23), updateEnkoder01, CHANGE);
00067   attachInterrupt(digitalPinToInterrupt(24), updateEnkoder01, CHANGE);
00068
00069   enk[1][0] = new Enkoder(26, 27, 2);
00070   attachInterrupt(digitalPinToInterrupt(26), updateEnkoder10, CHANGE);
00071   attachInterrupt(digitalPinToInterrupt(27), updateEnkoder10, CHANGE);
00072
00073   enk[1][1] = new Enkoder(44, 45, 2);
00074   attachInterrupt(digitalPinToInterrupt(44), updateEnkoder11, CHANGE);
00075   attachInterrupt(digitalPinToInterrupt(45), updateEnkoder11, CHANGE);
00076
00077   enk[2][0] = new Enkoder(28, 29, 2);
00078   attachInterrupt(digitalPinToInterrupt(28), updateEnkoder20, CHANGE);
00079   attachInterrupt(digitalPinToInterrupt(29), updateEnkoder20, CHANGE);
00080
00081   enk[2][1] = new Enkoder(30, 31, 2); // (pin, pin, tryb zliczania impulsow -> maly enkoder - 4,
      duzy enkoder - 2)
00082   attachInterrupt(digitalPinToInterrupt(30), updateEnkoder21, CHANGE);
00083   attachInterrupt(digitalPinToInterrupt(31), updateEnkoder21, CHANGE);
00084
00085   enk[3][0] = new Enkoder(32, 33, 2);
00086   attachInterrupt(digitalPinToInterrupt(32), updateEnkoder30, CHANGE);
00087   attachInterrupt(digitalPinToInterrupt(33), updateEnkoder30, CHANGE);
00088
00089   enk[3][1] = new Enkoder(34, 35, 2);
00090   attachInterrupt(digitalPinToInterrupt(34), updateEnkoder31, CHANGE);
00091   attachInterrupt(digitalPinToInterrupt(35), updateEnkoder31, CHANGE);
00092
00093   enk[4][0] = new Enkoder(36, 37, 2);
```

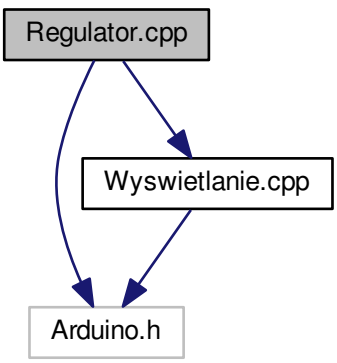
```
00094   attachInterrupt(digitalPinToInterrupt(36), updateEnkoder40, CHANGE);
00095   attachInterrupt(digitalPinToInterrupt(37), updateEnkoder40, CHANGE);
00096
00097   enk[4][1] = new Enkoder(38, 39, 2);
00098   attachInterrupt(digitalPinToInterrupt(38), updateEnkoder41, CHANGE);
00099   attachInterrupt(digitalPinToInterrupt(39), updateEnkoder41, CHANGE);
00100
00101   sterowniki[0] = new DualMC33926MotorShield(40, 12, 3, 13, 43);  // ( M1DIR - faza A, M1PWM -
       AENBL, M2DIR- faza B, M2PWM - BENBL, nD2 - MODE)
00102   sterowniki[1] = new DualMC33926MotorShield(14, 10, 25, 11, 50); // ( M1DIR - faza A, M1PWM -
       AENBL, M2DIR- faza B, M2PWM - BENBL, nD2 - MODE)
00103   sterowniki[2] = new DualMC33926MotorShield(16, 8, 15, 9, 50);   // ( M1DIR - faza A, M1PWM -
       AENBL, M2DIR- faza B, M2PWM - BENBL, nD2 - MODE)
00104   sterowniki[3] = new DualMC33926MotorShield(18, 6, 17, 7, 50);   // ( M1DIR - faza A, M1PWM -
       AENBL, M2DIR- faza B, M2PWM - BENBL, nD2 - MODE)
00105   sterowniki[4] = new DualMC33926MotorShield(20, 4, 19, 5, 50);   // ( M1DIR - faza A, M1PWM -
       AENBL, M2DIR- faza B, M2PWM - BENBL, nD2 - MODE)
00106
00107   //inicjacja sterownika silnika DC
00108   sterowniki[0]->init();
00109   sterowniki[1]->init();
00110   sterowniki[2]->init();
00111   sterowniki[3]->init();
00112   sterowniki[4]->init();
00113
00114  // tworzenie obiektów o typie Silnik
00115   sil[0][0] = new Silnik(sterowniki[0], 0, 1.0); // (sterownik, id, mnoznik)
00116   sil[0][1] = new Silnik(sterowniki[0], 1, 1.0);
00117   sil[1][0] = new Silnik(sterowniki[1], 0, 1.0);
00118   sil[1][1] = new Silnik(sterowniki[1], 1, 1.0);
00119   sil[2][0] = new Silnik(sterowniki[2], 0, 1.0);
00120   sil[2][1] = new Silnik(sterowniki[2], 1, 1.0);
00121   sil[3][0] = new Silnik(sterowniki[3], 0, 1.0);
00122   sil[3][1] = new Silnik(sterowniki[3], 1, 1.0);
00123   sil[4][0] = new Silnik(sterowniki[4], 0, 1.0);
00124   sil[4][1] = new Silnik(sterowniki[4], 1, 1.0);
00125
00126   napedy[0][0] = new SilnikRegulowany(new RegulatorTrojstawny(200, 15),
       enk[0][0], sil[0][0]);
00127   napedy[0][1] = new SilnikRegulowany(new RegulatorTrojstawny(200, 15),
       enk[0][1], sil[0][1]);
00128   napedy[1][0] = new SilnikRegulowany(new RegulatorTrojstawny(200, 15),
       enk[1][0], sil[1][0]);
00129   napedy[1][1] = new SilnikRegulowany(new RegulatorTrojstawny(200, 15),
       enk[1][1], sil[1][1]);
00130   napedy[2][0] = new SilnikRegulowany(new RegulatorTrojstawny(200, 15),
       enk[2][0], sil[2][0]);
00131   napedy[2][1] = new SilnikRegulowany(new RegulatorTrojstawny(200, 15),
       enk[2][1], sil[2][1]);
00132   napedy[3][0] = new SilnikRegulowany(new RegulatorTrojstawny(200, 15),
       enk[3][0], sil[3][0]);
00133   napedy[3][1] = new SilnikRegulowany(new RegulatorTrojstawny(200, 15),
       enk[3][1], sil[3][1]);
00134   napedy[4][0] = new SilnikRegulowany(new RegulatorTrojstawny(200, 15),
       enk[4][0], sil[4][0]);
00135   napedy[4][1] = new SilnikRegulowany(new RegulatorTrojstawny(200, 15),
       enk[4][1], sil[4][1]);
00136
00137   serwo_kciuk = new Serwo(2);
00138
00139   pal[0] = new PalecKciuk(0, napedy[0], serwo_kciuk, czuj_nac[0]);
00140   pal[1] = new PalecNorm(1, napedy[1], czuj_nac[1]);
00141   pal[2] = new PalecNorm(2, napedy[2], czuj_nac[2]);
00142   pal[3] = new PalecNorm(3, napedy[3], czuj_nac[3]);
00143   pal[4] = new PalecNorm(4, napedy[4], czuj_nac[4]);
00144
00145   reka = new Reka(pal, czuj_prad, natezenie_zatrzymania,
       dt_sec);
00146 }
00147
00151 void loop() {
00152   reka -> chwytCylindryczny(false, 1);
00153   Serial.println("zakonczono chwytCylindryczny");
00154   delay(6000);
00155   reka -> wyprost();
00156   Serial.println("zakonczono wyprost");
00157   delay(6000);
00158 }
00159
00160 void updateEnkoder00() { updateEnkoder(0, 0); }
00161 void updateEnkoder01() { updateEnkoder(0, 1); }
00162 void updateEnkoder10() { updateEnkoder(1, 0); }
00163 void updateEnkoder11() { updateEnkoder(1, 1); }
00164 void updateEnkoder20() { updateEnkoder(2, 0); }
00165 void updateEnkoder21() { updateEnkoder(2, 1); }
00166 void updateEnkoder30() { updateEnkoder(3, 0); }
00167 void updateEnkoder31() { updateEnkoder(3, 1); }
```

```
00168 void updateEnkoder40() { updateEnkoder(4, 0); }
00169 void updateEnkoder41() { updateEnkoder(4, 1); }
00170
00171
00177 void updateEnkoder(int palid, int enkid) {
00178   enk[palid][enkid] -> uaktualnij();
00179 }
```

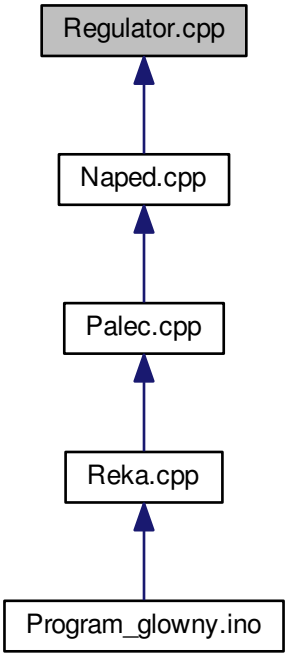
5.9 Dokumentacja pliku Regulator.cpp

```
#include "Arduino.h"
#include "Wyswietlanie.cpp"
```

Wykres zależności załączania dla Regulator.cpp:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class Regulator

  Bazowa klasa Regulator zawierająca metody wirtualne które są nadpisywane przez metody klas dziedziczących.
- class RegulatorPID

  Klasa RegulatorPID. Klasa ta zawiera algorytm liczący wartość sygnału sterującego prędkością silników.
- class RegulatorProporcjonalny

  Klasa RegulatorProporcjonalny. Klasa ta zawiera algorytm regulatora proporcjonalnego liczący wartość sygnału sterującego prędkością silników.
- class RegulatorTrojstawny

  Klasa RegulatorTrojstawny. Klasa ta zawiera algorytm regulatora trójstawnego liczący wartość sygnału sterującego prędkością silników.

Definicje

- #define Regulator_H
- #define DEBUG_MODE false

5.9.1 Opis szczegółowy

Autor

Violetta Munar Ernandes

Wersja

1.0

Definicja w pliku Regulator.cpp.

5.9.2 Dokumentacja definicji

5.9.2.1 #define DEBUG_MODE false

Definicja w linii 10 pliku Regulator.cpp.

Odwołania w RegulatorPID::uaktualnij(), RegulatorProporcjonalny::uaktualnij() i RegulatorTrojstawny::uaktualnij().

5.9.2.2 #define Regulator_H

Definicja w linii 8 pliku Regulator.cpp.

5.10 Regulator.cpp

```
00001
00007 #ifndef Regulator_H
00008 #define Regulator_H
00009
00010 #define DEBUG_MODE false
00011
00012 #include "Arduino.h"
00013
00014 #include "Wyswietlanie.cpp"
00015
00019 class Regulator {
00020   public:
00021     float syg_sterujacy;
00022
00023     Regulator () {
00024       syg_sterujacy = 0.0;
00025     }
00026
00027     virtual int uaktualnij(float uchyb) = 0;
00028
00029     virtual int zeruj() = 0;
00030
00031     virtual float zwrocSygnalSterujacy() {
00032
00033       return syg_sterujacy;
00034     }
00035
00036 };
00040 class RegulatorPID : public Regulator {
00041   public:
00042     float p, i, d;
00043     float kp, ki, kd;
00044     float dt_sec;
00045
00046     float uchyb_stary;
00047     float uchyb_aktualny;
00048
00049     RegulatorPID (float kp, float ki, float kd, float dt_s) {
00050       p = 0;
00051       i = 0;
00052       d = 0;
00053
00054       this -> kp = 0;
00055       this -> ki = 0;
```

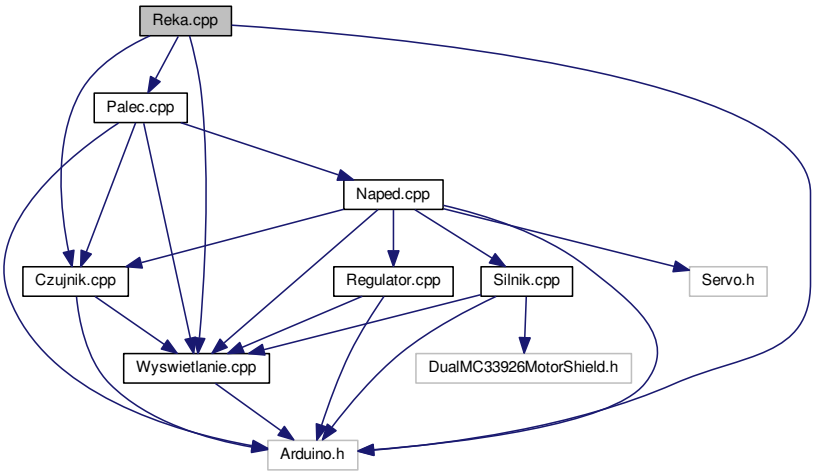
```
00056       this -> kd = 0;
00057
00058       this -> dt_sec = dt_s;
00059
00060       uchyb_stary = 0.0;
00061
00062       Serial.println("utworzono RegulatorPID");
00063     }
00064
00065
00066     int uaktualnij(float uchyb) {
00067       if (DEBUG_MODE) Serial.println("TODO: RegulatorPID.uaktualnij()");
00068
00069       // liczenie p, i, d
00070       p  = kp * uchyb;
00071       i += ki/2*dt_sec * uchyb;
00072       d  = (kd/dt_sec) * (uchyb - uchyb_stary);
00073
00074       // liczenie nowego syg_sterujacego
00075       syg_sterujacy = p + i + d;
00076
00077       wyswietl("uchyb RegulatorPID =", uchyb);
00078       if (DEBUG_MODE) wyswietl("sterowanie RegulatorPID =",
    syg_sterujacy);
00079
00080       return 0;
00081     }
00082
00083     int zeruj() {
00084       Serial.println(" RegulatorPID.zeruj()");
00085
00086       p = 0;
00087       i = 0;
00088       d = 0;
00089
00090       return 0;
00091     }
00092 };
00093
00097 class RegulatorProporcjonalny : public Regulator {
00098   public:
00099     float k;
00100
00101     RegulatorProporcjonalny (float k) {
00102       this -> k = k;
00103
00104       Serial.println("utworzono RegulatorProporcjonalny");
00105     }
00106
00107     int uaktualnij(float uchyb) {
00108       if (DEBUG_MODE) Serial.println("TODO: RegulatorProporcjonalny.uaktualnij()");
00109
00110       syg_sterujacy = uchyb*k;    // liczenie nowego syg_sterujacego
00111
00112       wyswietl("uchyb RegulatorProporcjonalny =", uchyb);
00113       if (DEBUG_MODE) wyswietl("sterowanie RegulatorProporcjonalny =",
    syg_sterujacy);
00114
00115       return 0;
00116     }
00117
00118     int zeruj() {
00119       k = 0;
00120       return true;
00121     }
00122
00123 };
00124
00128 class RegulatorTrojstawny : public Regulator {
00129   public:
00130     float k;
00131     float przedzial_tolerancji;
00136     RegulatorTrojstawny (float k, float tolerancja) {
00137       this -> k = k;
00138       this -> przedzial_tolerancji = tolerancja;
00139     }
00140
00144     int uaktualnij(float uchyb) {
00145       if (DEBUG_MODE) Serial.println("TODO: RegulatorTrojstawny.uaktualnij()");
00146
00147       // liczenie nowego syg_sterujacego
00148       if (uchyb >= -przedzial_tolerancji/2 && uchyb <= przedzial_tolerancji/2) { // jesli uchyb miesci sie
     w przedziale tolerancji
00149         syg_sterujacy = 0.0;
00150       } else if (uchyb < 0) {
00151         syg_sterujacy = -k; // jesli uchyb mniejszy od zera
00152       } else {
```

```
00153         syg_sterujacy =  k; // jesli uchyb wiekszy od zera
00154       }
00155
00156      // wyswietl("Reg Trojstawny uchyb  = ", uchyb);
00157      // if (DEBUG_MODE) wyswietl("sterowanie RegulatorTrojstawny =", syg_sterujacy);
00158
00159       return 0;
00160     }
00161
00162     int zeruj() {
00163       k = 0;
00164       przedzial_tolerancji = 0;
00165       return true;
00166     }
00167 };
00168
00169 #endif
```

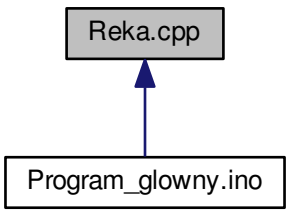
5.11 Dokumentacja pliku Reka.cpp

```
#include "Arduino.h"
#include "Wyswietlanie.cpp"
#include "Palec.cpp"
#include "Czujnik.cpp"
```

Wykres zależności załączania dla Reka.cpp:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class Reka

  Klasa Reka. Jest to główna klasa programu. Zarządza pracą wszystkich palców oraz czujnika prądu.

Definicje

- #define Reka_H

5.11.1 Opis szczegółowy

Autor

Violetta Munar Ernandes

Wersja

1.0

Definicja w pliku Reka.cpp.

5.11.2 Dokumentacja definicji

5.11.2.1 #define Reka_H

Definicja w linii 8 pliku Reka.cpp.

5.12 Reka.cpp

```
00001
00007 #ifndef Reka_H
00008 #define Reka_H
00009
00010 #include "Arduino.h"
00011
00012 #include "Wyswietlanie.cpp"
00013
00014 #include "Palec.cpp"
00015 #include "Czujnik.cpp"
00016
00017
00021 class Reka {
00022   private:
00023
00028     bool czyRuchPalcowZakonczony () {
00029       for (int i=0; i<IL_PALCOW; i++) {
00030         if (palce[i] -> czyRuchZakonczony() == false) {
00031           return false;
00032         }
00033       }
00034       return true;
00035     }
00036
00040    int poczekajDoZakonczeniaRuchu() {
00041       do {
00042         uaktualnij();
00043         delay( (int) (dt_sec*1000) );
00044       } while (czyRuchPalcowZakonczony() == false);
00045
00046       return 0;
00047    }
```

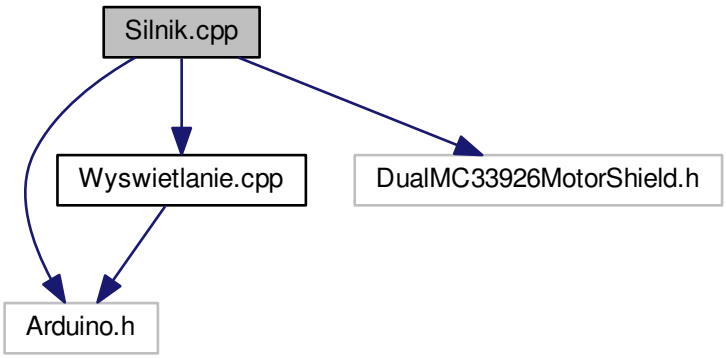
```
00048
00053   public:
00054     static const char TRYB_ZATRZYMANIA_NORMALNY = 0;
00055     static const char TRYB_ZATRZYMANIA_AWARYJNY = 1;
00056
00057     static const char IL_PALCOW = 5;
00058
00059     float dt_sec = 1.0/100.0;
00060
00061     Palec* palce[IL_PALCOW];
00062     CzujnikPradu* czujnik_pradu;
00063     float natezenie_zatrzymania_amp;
00064
00065
00074     Reka(Palec* palce[], CzujnikPradu* czujnik_pradu, float nat_zatrz, float dt) {
00075       this -> palce[0] = palce[0];
00076       this -> palce[1] = palce[1];
00077       this -> palce[2] = palce[2];
00078       this -> palce[3] = palce[3];
00079       this -> palce[4] = palce[4];
00080       this -> dt_sec = dt;
00081       this -> czujnik_pradu = czujnik_pradu;
00082       this -> natezenie_zatrzymania_amp = nat_zatrz;
00083     }
00084
00088     int uaktualnij() {
00089
00090       Serial.println("\n\nREKA");
00091
00092       if (czujnik_pradu -> zwrocWartosc() > natezenie_zatrzymania_amp) {
00093         zatrzymaj(TRYB_ZATRZYMANIA_NORMALNY);
00094       }
00095
00096       for (int i=0; i<IL_PALCOW; i++) {
00097         Serial.print("\n  PALEC "); Serial.println(i);
00098         palce[i] -> uaktualnij();
00099       }
00100
00101         return 0;
00102     }
00103
00104     // CHWYTY
00105
00111     int chwytSzczypcowy(bool pusty_chwyt, float max_sila) {
00112       Serial.println("\n\n\nTODO: Reka.chwytSzczypcowy()");
00113
00114       return 0;
00115     }
00116
00120     int chwytCylindryczny(bool pusty_chwyt, float max_sila) {
00121       Serial.println("\n\n\nTODO: Reka.chwytCylind()");
00122
00123 //      for (int i=0; i<IL_PALCOW; i++) {
00124 //        palce[i] -> zegnijKat(30, 30, 30, max_sila);
00125 //      }
00126
00127 //      palce[0] -> zegnijKat(60, 60, 60, max_sila);
00128       palce[1] -> zegnijKat(60, 60, max_sila);
00129
00130       this -> poczekajDoZakonczeniaRuchu();
00131
00132       return 0;
00133     }
00134
00138     int chwytSferyczny(bool pusty_chwyt, float max_sila) {
00139       Serial.println("\n\n\nTODO: Reka.chwytSferyczny()");
00140
00141        palce[0] -> zegnijKat(55, 90, 60, max_sila);
00142        palce[1] -> zegnijKat(0, 90, 60, max_sila);
00143        palce[2] -> zegnijKat(0, 60, 60, max_sila);
00144        palce[3] -> zegnijKat(0, 60, 60, max_sila);
00145        palce[4] -> zegnijKat(0, 90, 60, max_sila);
00146
00147        poczekajDoZakonczeniaRuchu();
00148
00149       return 0;
00150     }
00151
00155     int chwytHakowy(bool pusty_chwyt, float max_sila) {
00156       Serial.println("\n\n\nTODO: Reka.chwytHakowy()");
00157
00158        palce[0] -> zegnijKat(55,90, 30, max_sila);
00159        palce[1] -> zegnijKat(0, 20, 90, max_sila);
00160        palce[2] -> zegnijKat(0, 20, 90, max_sila);
00161        palce[3] -> zegnijKat(0, 20, 90, max_sila);
00162        palce[4] -> zegnijKat(0, 20, 90, max_sila);
00163
```

```
00164         poczekajDoZakonczeniaRuchu();
00165
00166       return 0;
00167     }
00168
00172     int chwytDloniowy(bool pusty_chwyt, float max_sila) {
00173       Serial.println("\n\n\nTODO: Reka.chwytDloniowy()");
00174
00175       return 0;
00176     }
00177
00181     int chwytLateralny(bool pusty_chwyt, float max_sila) {
00182       Serial.println("\n\n\nTODO: Reka.chwytLateralny()");
00183
00184       return 0;
00185     }
00186
00190     int wyprost() {
00191       Serial.println("\n\n\nTODO: Reka.wyprost()");
00192
00193       for (int i=0; i<IL_PALCOW; i++) {
00194         palce[i] -> wyprost();
00195       }
00196
00197       poczekajDoZakonczeniaRuchu();
00198
00199       return 0;
00200     }
00201
00202     // JEDEN PALEC
00203     // palce[nr].zegnijKat([d], a, b, c);
00204
00209     int zatrzymaj(int tryb_zatrz) {
00210       wyswietl("\n\n\nReka.zatrzymaj(tryb)", tryb_zatrz);
00211
00212       switch (tryb_zatrz) {
00213
00214         case TRYB_ZATRZYMANIA_NORMALNY:
00215           for (int i=0; i<IL_PALCOW; i++) {
00216             palce[i] -> zatrzymaj();
00217           }
00218           break;
00219
00220         case TRYB_ZATRZYMANIA_AWARYJNY:
00221           resetuj_do_ustawien_poczatkowych();
00222           for (int i=0; i<IL_PALCOW; i++) {
00223             palce[i] -> zatrzymaj();
00224           }
00225           break;
00226       }
00227       return 0;
00228     }
00229
00230     int zatrzymaj() {
00231       Serial.println("\n\n\nReka.zatrzymaj()");
00232
00233       return zatrzymaj(TRYB_ZATRZYMANIA_NORMALNY);
00234     }
00235
00236     int resetuj_do_ustawien_poczatkowych() {
00237
00238      for (int i=0; i<IL_PALCOW; i++) {
00239         palce[i] -> wyprost();
00240
00241
00242       return 0;
00243
00244     }
00245   }
00246 };
00247
00248 #endif
```

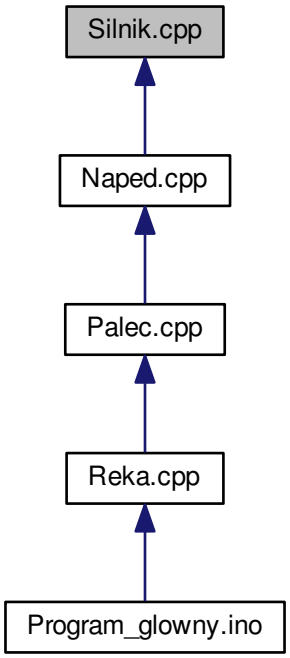
5.13 Dokumentacja pliku Silnik.cpp

```
#include "Arduino.h"
#include "Wyswietlanie.cpp"
#include "DualMC33926MotorShield.h"
```

Wykres zależności załączania dla Silnik.cpp:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class Silnik

  Klasa Silnik zawierająca metody sterujące pracą silników.

Definicje

- #define Silnik_H
- #define DEBUG_MODE false

5.13.1 Opis szczegółowy

Autor

Violetta Munar Ernandes

Wersja

1.0

Definicja w pliku Silnik.cpp.

5.13.2 Dokumentacja definicji

5.13.2.1 #define DEBUG_MODE false

Definicja w linii 10 pliku Silnik.cpp.

5.13.2.2 #define Silnik_H

Definicja w linii 8 pliku Silnik.cpp.

5.14 Silnik.cpp

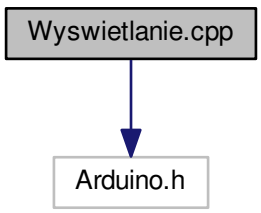
```
00001
00007 #ifndef Silnik_H
00008 #define Silnik_H
00009
00010 #define DEBUG_MODE false
00011
00012 #include "Arduino.h"
00013
00014 #include "Wyswietlanie.cpp"
00015 #include "DualMC33926MotorShield.h"
00016
00021 class Silnik {
00022   public:
00023
00024     static const char TRYB_ZATRZYMANIA_NORMALNY = 0; // mozna wznowic od razu;
       ustawianie predkosci na 0
00025     static const char TRYB_ZATRZYMANIA_AWARYJNY = 1; // wznawianie po uprzednim
       resecie do ustawien poczatkowych
00026
00027     float mnoznik_regulacji;
00028     float aktualna_predkosc;
00029
00030     int pin_silnika0;
00031     int pin_silnika1;
00032     int pin_silnika2;
00033
00034     DualMC33926MotorShield* sterownik;
00035     int id;
00036
00037     Silnik(DualMC33926MotorShield* ster, int id, float mnoznik_reg) {
00038       this -> sterownik = ster;
00039       this -> id = id;
00040
```

```
00041       this -> mnoznik_regulacji = mnoznik_reg;
00042       this -> aktualna_predkosc = 0.0;
00043
00044      // wyswietl("utworzono Silnik", (unsigned int) this);
00045     }
00046
00048     virtual int zatrzymaj() {
00049      // wyswietl("TODO: Silnik.zatrzymaj()", (unsigned int) this);
00050
00051       ustawPredkosc(0.0);
00052
00053       return 0;
00054     }
00055
00056     virtual int ustawPredkosc(float predkosc) {
00057       this -> aktualna_predkosc = predkosc*mnoznik_regulacji;
00058
00059       // ustawianie silnika fizycznego na predkosc
00060       if (this -> id == 0) {
00061         this -> sterownik -> setM1Speed((int) predkosc);
00062         wyswietl("Predkosc M1: ", aktualna_predkosc);
00063       } else if (this -> id == 1) {
00064         this -> sterownik -> setM2Speed((int) predkosc);
00065         wyswietl("Predkosc M2: ", aktualna_predkosc);
00066       }
00067
00068       return 0;
00069     }
00070 };
00071
00072 #endif
```

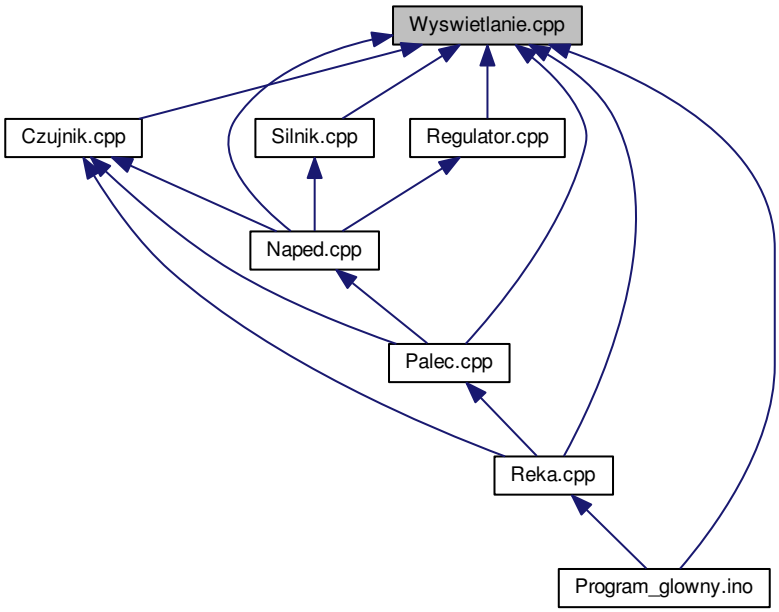
5.15 Dokumentacja pliku Wyswietlanie.cpp

```
#include "Arduino.h"
```

Wykres zależności załączania dla Wyswietlanie.cpp:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Definicje

- #define Wyswietlanie_H
- #define DEBUG_MODE false

Funkcje

- int wyswietl (String str, float i)

  Metoda wysyłająca opis i wartość zmiennej poprez interfejs szeregowy układu Arduino Due, które mogą być wświetlane w polu monitora szeregowego.

5.15.1 Opis szczegółowy

Autor

Violetta Munar Ernandes

Wersja

1.0

Definicja w pliku Wyswietlanie.cpp.

5.15.2 Dokumentacja definicji

5.15.2.1 #define DEBUG_MODE false

Definicja w linii 10 pliku Wyswietlanie.cpp.

5.15.2.2 #define Wyswietlanie_H

Definicja w linii 8 pliku Wyswietlanie.cpp.

5.15.3 Dokumentacja funkcji

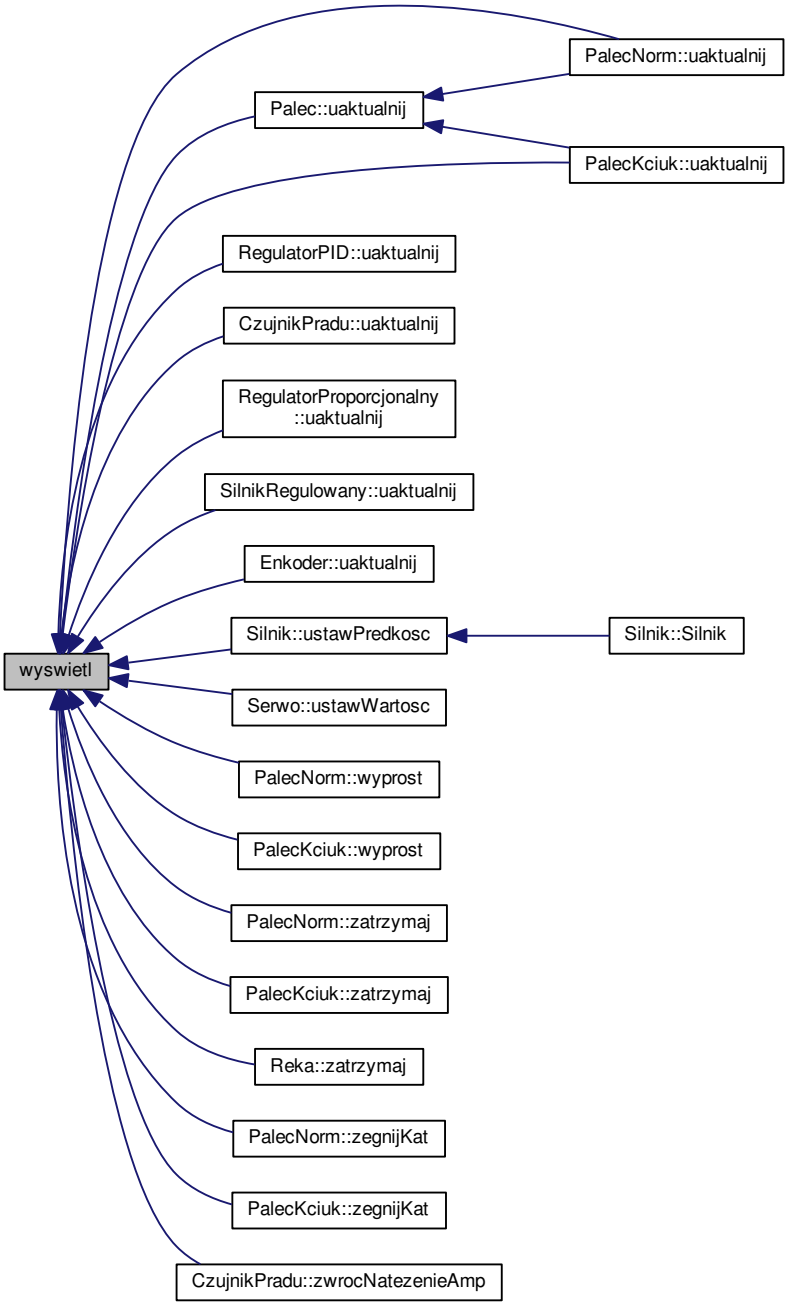
5.15.3.1 int wyswietl (String *str,* float *i*) [inline]

Metoda wysyłająca opis i wartość zmiennej poprez interfejs szeregowy układu Arduino Due, które mogą być wświetlane w polu monitora szeregowego.

Definicja w linii 14 pliku Wyswietlanie.cpp.

Odwołania w Palec::uaktualnij(), RegulatorPID::uaktualnij(), CzujnikPradu::uaktualnij(), RegulatorProporcjonalny::uaktualnij(), SilnikRegulowany::uaktualnij(), PalecNorm::uaktualnij(), PalecKciuk::uaktualnij(), Enkoder::uaktualnij(), Silnik::ustawPredkosc(), Serwo::ustawWartosc(), PalecNorm::wyprost(), PalecKciuk::wyprost(), PalecNorm::zatrzymaj(), PalecKciuk::zatrzymaj(), Reka::zatrzymaj(), PalecNorm::zegnijKat(), PalecKciuk::zegnijKat() i CzujnikPradu::zwrocNatezenieAmp().

Oto graf wywoływań tej funkcji:



5.16 Wyswietlanie.cpp

```
00001
00007 #ifndef Wyswietlanie_H
00008 #define Wyswietlanie_H
00009
00010 #define DEBUG_MODE false
00011
00012 #include "Arduino.h"
00013
```

```
00014 inline int wyswietl(String str, float i) {
00015  // cout << "  >> " << str  << " " << i << endl;
00016   Serial.print(str); Serial.print(" "); Serial.println(i);
00017
00018   return 0;
00019 }
00020
00021 #endif
```

Skorowidz