Spis treści

[1. Komunikacja z symulatorem Laparo 2](#_Toc515699898)

[1.1. Komendy wysyłane do symulatora 2](#_Toc515699899)

[1.2. Suma kontrolna 2](#_Toc515699900)

[1.3. Identyfikatory komend 2](#_Toc515699901)

[1.4. Identyfikatory komend o niepotwierdzonym działaniu 2](#_Toc515699902)

[1.5. Pingowanie urządzenia 3](#_Toc515699903)

[1.6. Dane wysyłane przez symulator 3](#_Toc515699904)

[1.6.1. Ramka wysyłana przez urządzenie ma następującą postać: 3](#_Toc515699905)

[1.6.2. Bajt nr 3 może mieć następujące wartości: 3](#_Toc515699906)

[1.6.3. Postać danych 4](#_Toc515699907)

[2. Biblioteka służąca do komunikacji z symulatorem Laparo 4](#_Toc515699908)

[2.1. Wymagania uruchomienia 4](#_Toc515699909)

[2.2. Interfejs publiczny 5](#_Toc515699910)

[2.2.1. Interejs ILaparoComunnicator 5](#_Toc515699911)

[2.2.2. Klasa CommunicatorFactory 5](#_Toc515699912)

[2.2.3. Używanie plików w zastępstwie urządzenia 5](#_Toc515699913)

[2.2.4. Klasy- modele danych 6](#_Toc515699914)

[2.2.5. Klasa LaparoCommunicatorImpl 6](#_Toc515699915)

[2.2.6. Klasa IntrernalData 7](#_Toc515699916)

[2.2.7. Klasa OutCommand 7](#_Toc515699917)

[3. Dostarczony program LaparoTester 8](#_Toc515699918)

[4. Program LaparoTester2 8](#_Toc515699919)

[4.1. Schemat blokowy działania programu 9](#_Toc515699920)

# Komunikacja z symulatorem Laparo

Komunikacja z urządzeniem odbywa się za pomocą danych wysyłanych po interfejsie USB. Mikroprocesor zamontowany w urządzeniu zarówno interpretuje dane przychodzące, jak i wysyła własne.

## Komendy wysyłane do symulatora

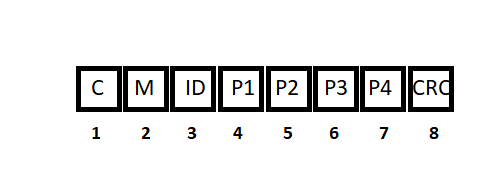
Urządzenie potrafi interpretować szereg komend. Muszą one zostać wysłane w postaci 8 bajtów, dokładna struktura ma postać:

1. C - Bajt 0x43, będący częścią protokołu, konsekwencje jego braku nie są jeszcze znane
2. M - Bajt 0x4D, będący częścią protokołu, konsekwencje jego braku nie są jeszcze znane
3. ID - Bajt stanowiący identyfikator konkretnej komendy
4. P1 do P4 - Cztery bajty stanowiące parametry komendy, w przypadku ich braku - zera
5. CRC - Bajt stanowiący sumę kontrolną, będący częścią protokołu, obecnie wygląda, że jego brak nie powoduje błędów

## Suma kontrolna

Każda komenda wysłana do urządzenia musi być zakończona bajtem z obliczoną sumą kontrolną. Jest ona liczona jako wynik dodania wszystkich pozostałych bajtów w komendzie (bajty 0 do 6). Obecnie nie wiadomo, co dzieje się w przypadku wartości większej niż 255, która nie zmieści się na jednym bajcie.

Poniżej znajduje się graficzne przedstawienie komendy



Rysunek 1 Skład ramki generowanej przez symulator

## Identyfikatory komend

* Bajt 0x50 reprezentuje komendę odpowiedzialną za pingowanie, nie przyjmuje żadnych parametrów
* Bajt 0x53 reprezentuje komendę odpowiedzialną za rozpoczęcie nadawania danych przez urządzenie, nie przyjmuje żadnych parametrów

## Identyfikatory komend o niepotwierdzonym działaniu

* Bajt 0x54 reprezentuje komendę nazwaną OUT\_SET\_TOOL\_TYPE
* Bajt 0x49  reprezentuje komendę nazwaną OUT\_SET\_STATE\_IDLE
* Bajt 0x42 reprezentuje komendę nazwaną OUT\_SET\_STATE\_BOOTLOADER
* Bajt 0x4D reprezentuje komendę nazwaną OUT\_SET\_STATE\_MEMS\_CALIB
* Bajt 0x43 reprezentuje komendę nazwaną OUT\_SET\_STATE\_PROX\_CALIB

## Pingowanie urządzenia

Po zainicjowaniu komunikacji z urządzeniem, aby ją utrzymać, należy wysyłać komendy 0x50. Jeżeli urządzenie nie otrzyma komendy 0x50 w czasie 1 s od ostatniej komendy 0x53 lub 0x50, zamknie swój koniec kanału komunikacyjnego.

## Dane wysyłane przez symulator

Zgodnie z obecnym stanem wiedzy, urządzenie wysyła wszystkie dane z czujników w równych odstępach czasu wynoszących 10 ms. Dane mają postać ciągu bajtów zakończonego sumą kontrolną. Program testowy dowodzi jednak, że dane wysyłane są rzadziej - mniej więcej co 30ms. Ponadto w ramach jednego komunikatu wysyłanych jest wiele danych.

### Ramka wysyłana przez urządzenie ma następującą postać:

* C - Bajt 0x43
* M - Bajt 0x4D
* Bajt symbolizujący charakter wysłanych danych (patrz dalej)
* Bajty składające się na konkretne dane, ich ilość jest zależna od poprzedniego bajtu
* Suma kontrolna

### Bajt nr 3 może mieć następujące wartości:

* 0x00 - brak komendy
* 0x52 - dostarczone dane dotyczą położenia końcówki jednego ze szczypiec
* 0x4D - w dostarczonym kodzie źródłowym oznaczone jako “IN\_RESULTS\_CAM”
* 0x41 - dostarczone dane dotyczą przyśpieszenia
* 0x56 - dostarczone dane dotyczą prędkości
* 0x4F - w dostarczonym kodzie źródłowym oznaczone jako “IN\_RESULTS\_OSC”
* 0x44 - dostarczone dane dotyczą odległości
* 0x53 - w dostarczonym kodzie źródłowym oznaczone jako “IN\_RESULTS\_CLAMPS\_S”
* 0x43 - w dostarczonym kodzie źródłowym oznaczone jako “IN\_RESULTS\_CLAMPS”
* 0x49 - w dostarczonym kodzie źródłowym oznaczone jako “IN\_RESULTS\_INFO”

### Postać danych

W przypadku danych o położeniu (bajt 0x52 - R) - dane te mają postać:

* Bajt wskazujący, którego ze szczypiec dotyczą dane, ma możliwe wartości: 0x4C(L) - lewy szczypiec, 0x52(R) - prawy szczypiec, 0x41(A) - w dostarczonym kodzie źródłowym oznaczony jako “SIDE\_ASSIST”
* 4 bajty symbolizujące głębokość końcówki szczypca
* 16 bajtów reprezentujące 4-liczbowy kwaternion oznaczający położenie końcówki
* 4 bajty reprezentujące rotację końcówki
* 4 bajty reprezentujące kąt pod jakim znajduje się końcówka

W przypadku danych symbolizowanych przez A, V, D, S, C lub O, dane mają postać:

* Bajt, którego znaczenia nie udało się poznać ze statycznej analizy dostarczonego kodu źródłowego
* 4 bajty reprezentujące konkretne dane

W wszystkich powyższych przypadkach zestawy 4 bajtów należy odczytywać jako liczbę zmiennoprzecinkową pojedynczej precyzji.

# Biblioteka służąca do komunikacji z symulatorem Laparo

Biblioteka udostępniona jest za pomocą pliku .dll, który należy dodać do swojej aplikacji jako referencję.

Biblioteka odpowiada za całość komunikacji z urządzeniem, udostępniając użytkownikowi interfejs umożliwiający pobieranie danych z symulatora oraz testowanie aplikacji w przypadku braku dostępu do urządzenia.

## Wymagania uruchomienia

W trybie korzystania z symulatora, biblioteka wymaga, aby w folderze jej uruchomienia znajdował się plik ‘portName.txt’, w którym zapisana ma być nazwa portu, do którego podłączony jest symulator. Przykładowa zawartość pliku:

„COM3”

### Znajdowanie odpowiedniej nazwy portu

* 1. Otwórz menadżer urządzeń
  2. Otwórz zakładkę Porty (COM i LPT)
  3. Odnajdź urządzenie Laparo (prawdopodobnie nazwane „urządzenie szeregowe”, nie ma możliwości podania konkretnej nazwy)
  4. Skopiuj nazwę portu podaną w nawiasie

Takie rozwiązanie, zostało wybrane ze względu na nieobecność biblioteki systemowej System.Managment w oprogramowaniu Unity.

## Interfejs publiczny

Biblioteka udostępnia interfejs ILaparoComunnicator, która odpowiada za całość komunikacji z urządzeniem. Ponadto dostępne są również klasy będące modelami danych.

### Interejs ILaparoComunnicator

Stanowi interfejs pomiędzy użytkownikiem biblioteki, a symulatorem.

#### Implementowane interfejsy:

IDisposable() : umożliwia używanie interfejsu w blokach using

#### Publiczne metody

* GetDataInEuler() : EulerData - zwraca najnowsze dane z urządzenia w formie kątów Eulera
* GetDataInQuaternion() : QuaternionData - zwraca najnowsze dane z urządzenia w formie kwaternionu

### Klasa CommunicatorFactory

Klasa służąca do otrzymywania odpowiedniej implementacji interfejsu ILaparoCommunicator.

#### Publiczne metody statyczne:

* GetMock(string path) - tworzy implementację, która będzie pobierać dane ze wcześniej przygotowanego pliku.
* GetCommunicator - tworzy implementację komunikującą się z symulatorem.

### Używanie plików w zastępstwie urządzenia

Klasa CommunicatorFactory umożliwia stworzenie implementacji interfejsu ILaparoCommunicator, która czyta dane z plików przez co umożliwia tworzenie i testowanie aplikacji bez dostępu do symulatora.

Metodzie GetMock należy przekaż ścieżkę do folderu, który zawiera trzy pliki:

* cartesian.tsv
* euler.tsv
* quaternion.tsv

Odczytywane będą tylko pliki w formacie .tsv. Pojedyncze liczby zmiennoprzecinkowe oddzielone są od siebie za pomocą znaku ‘/t’. Linie naprzemiennie odczytywane są jako dane z lewej i prawej końcówki.

Przykładowa linijka z pliku z danymi:

„12,3 11,3 4,5”

Pozycja ostatniej wczytanej linii jest wspólna, więc po wczytaniu dwóch linijek z danych kartezjańskich, czytanie danych w kwaternionach rozpocznie się od linii nr 3. Dane będą czytane w pętli zapewniając niezależność od długości pliku. Ponadto klasa wczytuje wszystkie dane podczas tworzenia obiektu, więc zmiany w pliku nie będą od razu widoczne w programie.

### Klasy- modele danych

#### Klasa EulerData

Reprezentuje położenia końcówek obu szczypiec za pomocą kątów Eulera.

##### Publiczne właściwości:

* LeftAngles : double[] - tablica kątów alfa, beta, gamma lewej końcówki
* RightAngles : double[] - tablica kątów alfa, beta, gamma prawej końcówki

#### Klasa QuaternionData

Reprezentuje położenia końcówek obu szczypiec za kwaternionów.

##### Publiczne właściwości

* LeftQuaternion : double[] - kwaternion reprezentujący położenie lewej końcówki
* RightQuaternion : double[] - kwaternion reprezentujący położenie prawej końcówki

Wszystkie powyższe klasy przeciążają metodę ToString(), aby zapewnić łatwe wypisanie danych.

### Klasa LaparoCommunicatorImpl

Klasa wyszukuje urządzenie Laparo wśród portów seryjnych komputera, inicjalizuje komunikację z nim, a następnie tworzy dwa wątki, z których jeden zajmuje się pingowaniem urządzenia, a drugi odczytywaniem danych przez nie wysłanych.

#### Implementowane interfejsy

ILaparoCommunicator

#### Pola prywatne

* port : System.IO.SerialPort - służy do komunikacji z urządzeniem
* pingerThread : System.Threading.Thread - reprezentuje wątek odpowiedzialny za pingowanie urządzenia
* receiverThread : System.Thread.Thread - reprezentuje wątek odpowiedzialny za odczytywanie danych otrzymanych od urządzenia

#### Metody publiczne:

* Dispose() - funkcja interfejsu IDisposable, zatrzymuje oba wątki stworzone podczas inicjalizacji oraz zamyka port
* GetEulerData() : EulerData - zwraca odpowiednio wypełnione dane dla prawej i lewej końcówki w postaci kątów Eulera
* GetQuaterionData : QuaterionData - zwraca odpowiednio wypełnione dane dla prawej i lewej końcówki w postaci kwaternionu

#### Metody prywatne

* SetPort() – szuka pliku „portName.txt” w folderze, z którego biblioteka została uruchomiona, a następnie próbuje odczytać z niego nazwę portu do otwarcia. Rzuca IOException, jeżeli nie znajdzie urządzenia lub pliku.
* PingDevice() - w nieskończonej pętli wysyła komendę PING do urządzenia i usypia wątek na 1s
* ReceiveData() - w nieskończonej pętli odczytuje wszystkie bajty z pola port, wypisuje je na konsolę i usypia wątek na 10ms
* SendCommand(OutCommand command) - tworzy tablicę 8 bajtów z podanego obiektu komendy, uzyskuje blokadę na pole port i wysyła dane
* InitializeThreads() - inicjalizuje pole pingerThread wątkiem wykonującym metodę PingDevice() oraz pole receiverThread wątkiem wykonującym metodę ReceiveData()

### Klasa IntrernalData

Klasa jest odpowiedzialna za interpretację danych otrzymanych od symulatora.

#### Klasa wewnętrzna: Data

Jest odpowiedzialna za przechowywanie danych otrzymanych od symulatora o konkretnej końcówce. Posiada pola reprezentujące odległość, oscylację, prędkość i przyspieszenie oraz kwaternion i kąty Eulera, które wysyła symulator.

#### Metody publiczne:

* ProccessData(byte[] bytes ) : intepretuje otrzymaną tablice bajtów na podstawie znanego schematu ramki danych. Wypełnia odpowiedni pola klasy Data.
* GetQuad(Side s) : zwraca tablicę liczb zmiennoprzecinkowych, które reprezentują kwaternion mówiący o położeniu danej końcówki.
* GetAngles(Side s) : zwraca tablicę liczb zmiennoprzecinkowych, które reprezentują kąty Eulera konkretnej końcówki.

### Klasa OutCommand

Reprezentuje komendę wysyłaną do urządzenia. Obecnie nie umożliwia podawania parametrów komendy.

#### Publiczne konstruktory

* OutCommand(OutCommandId ID) - inicjalizuje pole id argumentem ID oraz wywołuje metodę CalculateCrc()

#### Prywatne pola

* Id : byte - reprezentuje id komendy
* crc : byte - reprezentuje sumę kontrolną komendy

#### Prywatne metody

* CalculateCrc() - oblicza sumę kontrolną komendy i ustawia na nią pole crc

#### Typ wyliczeniowy OutCommandId

Typ wyliczeniowy typu byte reprezentuje id komendy.

##### Publiczne pola:

* + PING - wartość 0x50
  + SET\_STATE\_SENDING - wartość 0x53

# Testowanie biblioteki

# Dostarczony program LaparoTester

Program ten korzysta z biblioteki LaproCommunicator, aby pokazać jej działanie w przypadku korzystania z gotowego zestawu danych.

Uruchomienie programu wymaga skopiowania dostarczonego folderu „data” do lokalizacji „C:/”.

Program po uruchomieniu wyświetli dane ze wszystkich plików w folderze „data” (niemożliwe jest rozróżnienie formatu danych).

# Program LaparoTester2

Uwaga: obecnie wykonanie programu jest powstrzymane przez wyjątek podczas parsowania danych otrzymanych od symulatora.

Uwaga: program wymaga uprawnień administratora! W przypadku debugowania, środowisko programistyczny musi zostać uruchomione jako administrator.

Po uruchomieniu programu pojawia się konsola. W przypadku braku urządzenia wypisany zostanie odpowiedni komunikat.

W przypadku odnalezienia podłączonego symulatora, program zaczyna czytać dane otrzymane od niego. Dane nie są wypisywane, ale sposób jego działania można zobaczyć czytając plik „C;/laparo\_logs.txt”. Próba pobrania danych jest wykonywana co 10ms.

Program wstrzymuje swoje działanie po krótkim czasie (około minuty).

## Schemat blokowy działania programu

