Na prawach rękopisu

INSTYTUT INFORMATYKI, AUTOMATYKI I ROBOTYKI
POLITECHNIKI WROCŁAWSKIEJ
Raport serii SPR nr 3/05

Programowanie i konstrukcja kulistego robota społecznego wspomagającego terapię dzieci autystycznych

Krzysztof Arent
Marek Kabała
Marek Wnuk

Słowa kluczowe:
robot społeczny,
logika rozmyta,
zachowanie,
sterownik,
czujnik,
interfejs.

Wrocław 2005

Spis treści

1 Wprowadzenie

Roboty społeczne są stosunkowo młodą gałęzią robotyki. Sformułowana w [10] definicja wymienia cztery podstawowe cechy takiego robota. Robot dysponuje zdolnością rozpoznawania członków grupy, w której się znajduje (obejmującej zarówno inne roboty jak i ludzi), umie postrzegać i interpretować otaczający go świat poprzez swoje doświadczenie, bezpośrednio komunikować się z ludźmi oraz innymi robotami, a także uczyć się od nich. We wspomnianej pracy pokazano bardzo szerokie zastosowanie robotów społecznych. Wykorzystywane są jako obiekty eksperymentów (przy weryfikacji, ocenie i korygowaniu nowych teorii w psychologii, neurobiologii, badaniach nad rozwojem społecznym i biologicznym, etc.), w usługach (roboty asystenci, partnerzy), do rozrywki (roboty towarzyskie, zabawki), w terapii (autyzm u dzieci), edukacji (zespołowe konstruowanie robotów, zawody robotów), humanoidy.

Niniejszy projekt związany jest z badaniami nad zastosowaniem robotów społecznych w terapii dzieci autystycznych. Szerokie badania w tym zakresie prowadzone są od 1998 roku w ramach projektu Aurora (AUtonomous RObotic platform as a Remedial tool for children with Autism), [4]. Uwzględniając specyfikę choroby (zob. [15]) i doświadczenia terapeutów, zbudowano szereg robotów, dających możliwość oddziaływania na dzieci poprzez ruch, mowę, muzykę, kolory, efekty świetlne, i rozmaitych kształtach, począwszy od kuli a skończywszy na urządzeniach z tułowiami i ramionami (lalka Robota). Wnioski z dotychczas uzyskanych wyników ([8]) wskazują na celowość i potrzebę kontynuowania badań w tej dziedzinie. W szczególności, należy dalej rozszerzać spektrum robotów dedykowanych terapii autyzmu i intensywnie pracować nad nowymi metodami terapeutycznymi bazującymi na robotach społecznych.

Zgodnie z założeniami ustalonymi przy współudziale terapeutów i specjalistów z Dolnośląskiej Szkoły Wyższej Edukacji (Instytut Pedagogiki Specjalnej), Uniwersytetu Wrocławskiego i Centrum Rehabilitacji i Neuropsychiatrii w Mikoszowie, robot kulisty reaguje na dotyk, ruch, siłę rzutu, natężenie i wysokość dźwięku przy pomocy kolorowego światła o różnym natężeniu i modulacji, jednorodnego dźwięku o różnym charakterze, melodii lub komunikatów słownych, wibracji o różnym natężeniu.

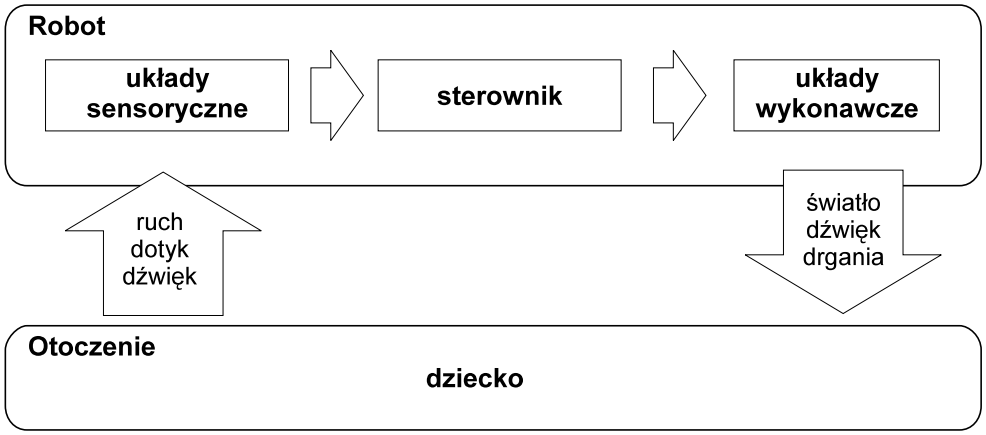
Oprócz czujników zbliżeniowych, żyroskopów i akcelerometrów, różnokolorowych źródeł światła oraz układów do rejestracji i odtwarzania dźwięku robot jest wyposażony w sterownik pozwalający programować jego zachowanie (reakcje na bodźce).

W specyfikacji robota bardzo istotne znaczenie ma postulat pełnej obsługi i programowania robota przez terapeutę, bez udziału robotyka. Realizacja tego postulatu wymaga opracowania i stworzenia interfejsu terapeuty. Interfejs powinien dawać możliwość definowania rozmaitych zachowań robota (w bardzo szerokim zakresie), stosownie do indywidualnych cech dziecka i celów terapeutycznych, w sposób, który dla terapeuty jest naturalny i intuicyjny. Ponadto, interfejs powinien być otwarty na modyfikacje wynikające z przebiegu eksperymentów terapeutycznych i zrealizowany przy użyciu darmowego oprogramowania, dostępnego dla wszystkich najważniejszych systemów operacyjnych.

Wstępne koncepcje opisywanego poniżej terapeutycznego robota kulistego zostały przedstawione w pracach [3], [13], [23].

Kulisty robot interaktywny przeznaczony do terapii autyzmu u dzieci młodszych został zaprojektowany jako reaktywna zabawka umożliwiająca terapeucie dostosowywanie zachowania do indywidualnych cech dziecka na podstawie dotychczasowych wyników prowadzonych ćwiczeń.

Zachowanie robota jako reakcja na określony stan otoczenia jest definiowane przez terapeutę i zapisane w pamięci sterownika (rys. 1). Na podstawie reguł zachowań i stanu otoczenia sterownik robota określa działania, które za pomocą układów wykonawczych wpływają na stan otoczenia i mają prowokować dziecko do interakcji.



Rysunek 1: Schemat blokowy układu robot – otoczenie

1.1 Układy sesoryczne

Prawidłowe określenie aktualnego stanu otoczenia i określenie przez terapeutę reguł zachowania robota jest możliwe przy założeniu, że za pośrednictwem układów sensorycznych można rozpoznać stan emocjonalny dziecka. Pociąga to za sobą założenie, że dziecko ujawnia swój stan emocjonalny poprzez działania, oraz że działania te mogą być zarejestrowane przez układy sensoryczne. Robot o zwartej konstrukcji, zawierający w sobie wszystkie układy sensoryczne wymaga by działania dziecka, które można rejestrować albo dotyczyły bezpośrednio robota (rzucanie, potrząsanie, toczenie), albo miały taką formę, że mogą być zarejestrowane przez sensor umieszczony w robocie (np. krzyk).
Parametry stanu robota i jego otoczenia, które dają się rejestrować przez dostępne sensory to:

- zmiana pozycji i orientacji robota,
- nacisk na obudowę robota,
- dotyk powierzchni obudowy robota,
- natężenie dźwięku w otoczeniu robota.

Do określenia aktualnych wartości wspomnianych wcześniej parametrów stanu otoczenia wykorzystane zostały:

- sensory zbliżeniowe,
- akcelerometry,
- żyroskopy,
- czujnik ciśnienia,
- mikrofon.

Sensory zbliżeniowe rozmieszczono równomiernie na całej powierzchni robota w celu określania liczby oraz powierzchni obszarów styku obudowy robota z elementami zewnętrznymi. Możliwe jest również badanie zmian tych parametrów w czasie (interpretacja działań dziecka – chwytanie robota). Jako sensory zbliżeniowe zastosowano układy wykorzystujące pole magnetyczne, które mają zasięg działania pozwalający na zmniejszenie ich liczby do ośmiu. Gdyby doświadczenia wykazały potrzebę

umieszczenia na powierzchni robota większej liczby sensorów, w celu dokładniejszego odtworzenia kształtu powierzchni styku, przewidziano wykorzystanie sensorów optycznych działających na zasadzie światła odbitego.
Akcelerometry i żyroskopy umożliwiają rejestrację zmiany położenia i orientacji robota, czyli postrzeganie takich zdarzeń jak toczenie robota, potrząsanie nim czy rzucanie.
Czujnik ciśnienia wewnątrz szczelnej i elastycznej obudowy robota umożliwia rejestrację takich zdarzeń jak odbijanie się robota od podłoża i przeszkód oraz nacisk na obudowę robota. Badanie ciśnienia wewnątrz obudowy umożliwia również odróżnienie od siebie takich zdarzeń jak swobodne odbijanie się robota od podłoża i potrząsania robotem (w obu przypadkach przyspieszenia mierzone akcelerometrami są podobne, natomiast ciśnienie wewnątrz obudowy zachowuje się różnie).
Rejestracja dźwięku w otoczeniu robota odbywa się przy użyciu mikrofonów elektretowych umieszczonych przy powierzchni obudowy robota.

1.2 Sterownik

Sterownik robota pozwala na łatwe przeprogramowywanie algorytmu sterowania obiektem, którym jest opisywana kula. Następujące zadania są w sterowniku wykonywane cyklicznie:

- odczytywanie sygnałów z czujników przyspieszenia, prędkości kątowej, natężenia i wysokości dźwięku, dotyku (zbliżeniowych),
- wstępne przetwarzanie sygnałów na wielkości pośrednie (bodźce) stosowane w opisie zachowania robota,
- realizowanie zadanego przez terapeutę zachowania (wyliczanie parametrów definiujących reakcje na bodźce),
- wtórne przetwarzanie wyliczonych parametrów na synały sterujące urządzeniami wykonawczymi.

Poza tym, do zadań sterownika należy zapewnienie możliwości przeprogramowywania zachowania robota, nadzór podstawowych parametrów pracy (np. zasilania) i komunikacja z komputerem nadrzędnym w celu rejestrowania danych pomiarowych.

1.3 Układy wykonawcze

Odpowiedzią robota na działania dziecka są akcje robota (określone przez terapeutę). Powinny one być na tyle interesujące z punktu widzenia dziecka, by skłonić je do podjęcia interakcji. Możliwości robota są ograniczone w takim stopniu, by zachować prostotę i niski koszt konstrukcji i jednocześnie nie stracić możliwości osiągnięcia celu terapeutycznego. Jako elementy wykonawcze zastosowano:

- różnokolorowe diody świecące (LED),
- silnik elektryczny (wibrator),
- głośnik.

Diody świecące umieszczone w ośmiu gniazdach rozłożonych równomiernie na całej powierzchni obudowy robota umożliwiają emitowanie światła czerwonego, zielonego i niebieskiego. Modulacja prądu diod umożliwia zmianę jasności świecenia i odcienia barwy wypadkowej, co pozwala na

uzyskanie rozmaitych efektów świetlnych. Silnik elektryczny z niewielkim, mimośrodowo umieszczonym obciążnikiem (znany z telefonów komórowych wibrator) umożliwia uzyskanie drgań o amplitudzie regulowanej prądem zasilania. Głośnik jest wykorzystywany do emitowania dźwięków i komunikatów słownych wprowadzonych wcześniej do pamięci rejestratora (odtwarzacz/dyktafon USB). Część radiową tego modułu można w przyszłości wykorzystać do przekazywania dźwięku od terapeuty.

2 Sterownik robota

Ważną zaletą opracowanego dla robota sterownika [23] jest łatwość programowania zachowań robota przez osoby nie zajmujące się robotyką. Wykorzystano koncepcję wnioskowania rozmytego w oparciu o bazę wiedzy utworzoną przez eksperta (terapeutę), co pozwoliło zbudować odpowiedni interfejs użytkownika [3].

2.1 Rozmyty model zachowania robota

Definiujac rozmyty model robota autystycznego przyjęto ujednoliconą notację nazw zmiennych lingwistycznych opartą na mnemotechnicznych skrótach:

- t - Touch
- d - Derivative
- x - (często tak się oznacza położenie)
- a - Acceleration
- r - Rotation
- A - Amplitude
- p - Pressure
- f - Frequency
- s - Sound
- l - Low
- m - Middle
- h - High, Hue
- i - Intensity
- fl - Flash
- q - Quaver
- o - Order
- v - Volume

Zgodnie z nią określono następujące bodźce – zmienne wejściowe:

- DOTYK:

  t - powierzchnia dotyku
  td - pochodna powierzchni dotyku
  txd - pochodna zmiany miejsca dotyku

- PRZYSPIESZENIE LINIOWE:

  ad - pochodna przyspieszenia
  af - czestotliwość zmian przyspieszenia

- PRĘDKOŚĆ WIROWANIA:

  rA - amplituda prędkości wirowania
  rAd - pochodna amplitudy prędkości wirowania

- CIŚNIENIE:

  p - ciśnienie wewnątrz kuli
  pd - pochodna ciśnienia wewnętrznego
  pf - czestotliwość zmian ciśnienia wewnętrznego

- DŹWIĘK:

  slA - amplituda częstotliwości niskich
  smA - amplituda częstotliwości średnich
  shA - amplituda częstotliwości wysokich

Podobnie zdefiniowano reakcje – zmienne wyjściowe:

- ŚWIECENIE:

  h - odcień barwy świecenia
  hd - pochodna odcienia barwy świecenia
  i - intensywność świecenia
  id - pochodna intensywności świecenia
  flf - czestotliwość migotania
  flfd - pochodna czestotliwości migotania

- WIBRACJE:

  qA - amplituda wibracji

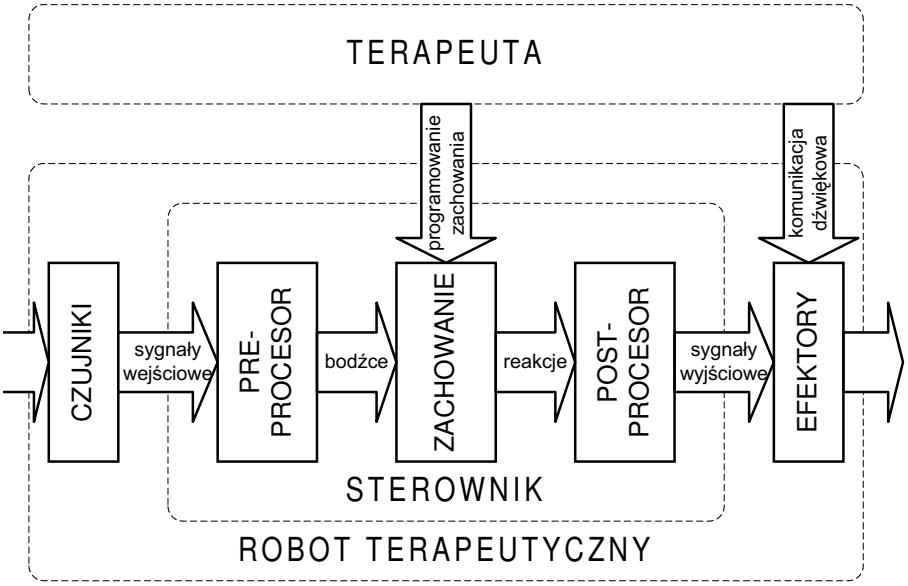
- SYGNAŁY DŹWIĘKOWE:

  so - porządek sygnałów dźwiękowych
  sv - głośność generowanego dźwięku

Krótkie nazwy zmiennych są podyktowane wygodą pracy w edytorze Matlaba i przejrzystości plików z modelami robota terapeutycznego.

2.2 Struktura układu sterowania

Struktura sterownika została przedstawiona na rys. 2. Jednostka centralna obsługuje układy wejściowe i wyjściowe za pośrednictwem różnych interfejsów (porty równoległe, interfejs szeregowy SPI, układ czasowo–licznikowy -*timer*).
Cyklicznie wykonywany jest następujący ciąg przetwarzania sygnałów wejściowych na wyjściowe:



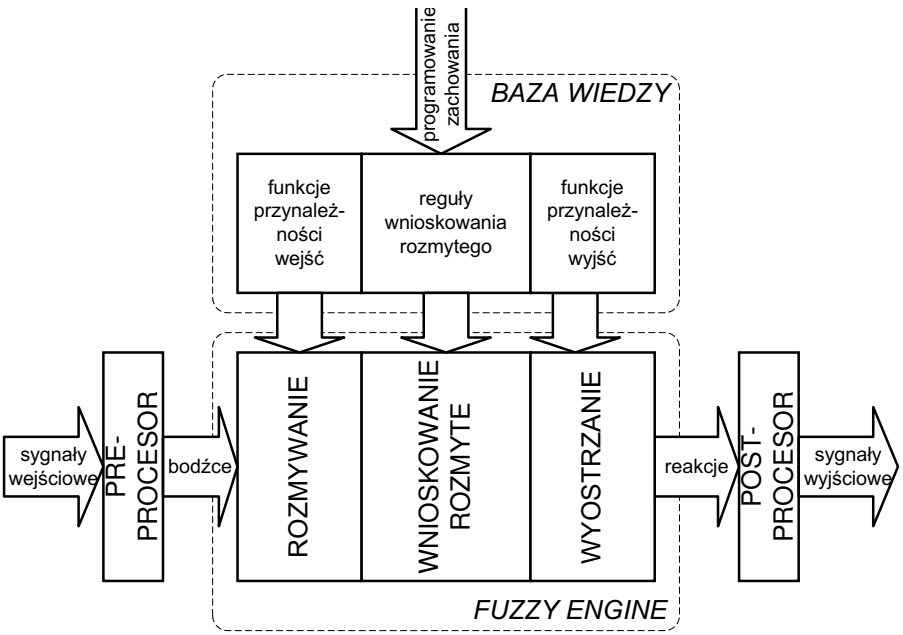
Rysunek 2: Struktura układu sterowania.

- preprocesor przetwarza bieżące wartości pomiarów (przyspieszenia, predkości obrotowej, dotyku, dźwięku itd.) na bodźce;
- programowalny blok realizujący zachowanie robota przetwarza bodźce na reakcje;
- postprocesor wytwarza bieżące wartości sygnałów wyjściowych (prądu zasilającego diody LED, częstotliwości migotania, amplitudy i częstotliwości wibracji itd.) na podstawie reakcji.

Działanie sterownika zostało zdekomponowane na trzy bloki, z których dwa (preprocesor i postprocesor) mają charakter dynamiczny, a jeden (zachowanie) jest blokiem statycznym o wielu wejściach i wielu wyjściach. Bloki dynamiczne są silnie powiązane ze sprzętem realizującym pomiary i generującym sygnały wyjściowe. Blok zachowania jest praktycznie niezależny od sprzętu, dzięki czemu można go przeprogramowywać bez specjalnego przygotowania technicznego. Programowanie zachowania wymaga jedynie opisania zależności pomiędzy reakcjami a bodźcami.

2.3 Realizacja zachowania

Zachowanie robota interaktywnego jest definiowane przez podanie statycznej funkcji wielu zmiennych. Ze względu na konieczność dostosowywania zachowania kuli do indywidualnych cech dziecka, w trakcie procesu terapii musi istnieć możliwość określania tej funkcji przez terapeutę. Co więcej, nie jest znany model obiektu, z którym robot pozostaje w interakcji (dziecka) a zatem nie jest możliwe dokonanie syntezy sterownika na tej podstawie.
Koncepcja sterowania rozmytego opartego na bazie wiedzy zbudowanej na podstawie doświadczenia ekspertów wydaje się znajdować w naszym przypadku szczególnie dobre zastosowanie. Sterownik rozmyty w sensie Mamdaniego [5] składa się z dynamicznego preprocesora, dynamicznego postprocesora oraz statycznego odwzorowania zrealizowanego przy pomocy bloku wnioskowania rozmytego (rys. 3).



Rysunek 3: Sterownik rozmyty wedlug Mamdaniego.

Algorytm sterowania jest pamiętany w postaci zestawu reguł o charakterze **if–then**. Reguły te operują na zmiennych lingwistycznych wartościowanych przy pomocy rozmytych funkcji przynależności, którym nadaje się nazwy potoczne odpowiadające intuicji eksperta.
Wykonanie jednego kroku algorytmu wymaga:

- rozmycia (*fuzzification*) ostrych wartości bodźców (określenia stopnia przynależności dla poszczególnych wartości zmiennych lingwistycznych opisujących bodźce),
- wnioskowania (*inference*) na podstawe reguł zawartych w bazie wiedzy i uzyskanych rozmytych wartości bodźców, co prowadzi do rozmytych wartości reakcji,
- wyostrzenia (*defuzzification*) wartości reakcji (określenia ostrych wartości reprezentujących wyniki wnioskowania).

Informacje niezbędne do przeprowadzenia wszystkich powyższych etapów znajdują się w bazie wiedzy. Rozmywanie bodźców i wyostrzanie reakcji są zdefiniowane przez podanie funkcji przynależności dla odpowiednich zmiennych lingwistycznych i mają ścisły związek ze sprzętową realizacją akwizycji sygnałów i programowym przetwarzaniem odpowiednich zmiennych (nie są przewidziane do definiowania przez terapeutę). Możliwe jest jedynie pewne uelastycznienie tych etapów przez udostępnienie terapeucie, w ograniczonym zakresie, skalowania wartości sygnałów.

2.4 Język FCL

Język opisu programu sterownika rozmytego został ujednolicony w ramach standardu IEC 1131 – PROGRAMMABLE CONTROLLERS [12]. W części 7. tej normy, zatytułowanej “Fuzzy Control Programming”, opisano składnię i zasady stosowania języka FCL (ang. *Fuzzy Control Language*). Dostosowanie się do tego standardu zapewnia ujednolicenie nazewnictwa, standaryzację składni i przenośność oprogramowania. Składnia języka FCL jest zdefiniowana następująco:

function_block_declaration ::= FUNCTION_BLOCK function_block_name
{fb_io_var_declarations}
{other_var_declarations}
function_block_body
END_FUNCTION_BLOCK

fb_io_var_declarations ::= input_declarations | output_declarations

other_var_declarations ::= var_declarations

function_block_body ::= {fuzzify_block}
{defuzzify_block}
{rule_block}
{option_block}

fuzzify_block ::= FUZZIFY variable_name
{linguistic_term}
END_FUZZIFY

defuzzify_block ::= DEFUZZIFY f_variable_name
{linguistic_term}
defuzzification_method
default_value
[range]
END_DEFUZZIFY

rule_block ::= RULEBLOCK rule_block_name
operator_definition
[activation_method]
accumulation_method
{rule}
END_RULEBLOCK

option_block ::= OPTION
any manufacturer specific parameter
END_OPTION

linguistic_term ::= TERM term_name := membership_function ;

membership_function ::= singleton | points

singleton ::= numeric_literal | variable_name

points ::= { (numeric_literal | variable_name ,
numeric_literal) }

defuzzification_method ::= METHOD : COG | COGS | COA | LM | RM ;

default_value ::= DEFAULT := numeric_literal | NC ;

range ::= RANGE := (numeric_literal .. numeric_literal) ;

operator_definition ::= (OR : MAX | ASUM | BSUM) |
(AND : MIN | PROD | BDIF) ;

activation_method ::= ACT : PROD | MIN ;

accumulation_method ::= ACCU : MAX | BSUM | NSUM ;

rule ::= RULE integer_literal :
IF condition THEN conclusion [WITH weighting_factor] ;

condition ::= (subcondition | variable_name) {AND | OR
(subcondition | variable_name)}

subcondition ::= (NOT (variable_name IS [NOT]) term_name)) |
(variable_name IS [NOT] term_name)

conclusion ::= { (variable_name | (variable_name IS term_name)) , }
(variable_name | variable_name IS term_name)

weighting_factor ::= variable | numeric_literal

2.5 Model zachowania robota w FCL

W celu zamodelowania zachowania terapeutycznego robota kulistego w języku FCL zadeklarowano wejściowe i wyjściowe zmienne lingwistyczne (por. 2.1):

```
FUNCTION_BLOCK ra
VAR_INPUT
t: REAL;
td: REAL;
txd: REAL;
ad: REAL;
af: REAL;
rA: REAL;
rAd: REAL;
p: REAL;
pd: REAL;
pf: REAL;
slA: REAL;
smA: REAL;
shA: REAL;
END_VAR
VAR_OUTPUT
h: REAL;
hd: REAL;
i: REAL;
id: REAL;
flf: REAL;
flfd: REAL;
qA: REAL;
so: REAL;
sv: REAL;
END_VAR
```

W dalszym ciągu następują bloki rozmywania (definicje termów dla poszczególnych zmiennych wejściowych):

```
FUZZIFY t
TERM medium := (  0.10,0.0) (  0.50,1.0) (  0.90,0.0);
TERM large := (  0.60,0.0) (  1.00,1.0) (  1.00,0.0);
TERM small := (  0.00,0.0) (  0.00,1.0) (  0.40,0.0);
END_FUZZIFY
FUZZIFY td
TERM n_large := ( -1.00,0.0) ( -1.00,1.0) ( -0.60,0.0);
TERM zero := ( -0.40,0.0) (  0.00,1.0) (  0.40,0.0);
TERM large := (  0.60,0.0) (  1.00,1.0) (  1.00,0.0);
TERM n_medium := ( -0.90,0.0) ( -0.50,1.0) ( -0.10,0.0);
TERM medium := (  0.10,0.0) (  0.50,1.0) (  0.90,0.0);
END_FUZZIFY
```

```
FUZZIFY txd
TERM small := ( -0.40,0.0) (  0.00,1.0) (  0.40,0.0);
TERM medium := (  0.10,0.0) (  0.50,1.0) (  0.90,0.0);
TERM large := (  0.60,0.0) (  1.00,1.0) (  1.40,0.0);
END_FUZZIFY
FUZZIFY ad
TERM n_large := ( -1.00,0.0) ( -1.00,1.0) ( -0.60,0.0);
TERM zero := ( -0.40,0.0) (  0.00,1.0) (  0.40,0.0);
TERM large := (  0.60,0.0) (  1.00,1.0) (  1.00,0.0);
TERM n_medium := ( -0.90,0.0) ( -0.50,1.0) ( -0.10,0.0);
TERM medium := (  0.10,0.0) (  0.50,1.0) (  0.90,0.0);
END_FUZZIFY
FUZZIFY af
TERM small := ( -0.40,0.0) (  0.00,1.0) (  0.40,0.0);
TERM medium := (  0.10,0.0) (  0.50,1.0) (  0.90,0.0);
TERM large := (  0.60,0.0) (  1.00,1.0) (  1.40,0.0);
END_FUZZIFY
FUZZIFY rA
TERM small := ( -0.40,0.0) (  0.00,1.0) (  0.40,0.0);
TERM medium := (  0.10,0.0) (  0.50,1.0) (  0.90,0.0);
TERM large := (  0.60,0.0) (  1.00,1.0) (  1.40,0.0);
END_FUZZIFY
FUZZIFY rAd
TERM n_large := ( -1.00,0.0) ( -1.00,1.0) ( -0.60,0.0);
TERM zero := ( -0.40,0.0) (  0.00,1.0) (  0.40,0.0);
TERM medium := (  0.10,0.0) (  0.50,1.0) (  0.90,0.0);
TERM n_medium := ( -0.90,0.0) ( -0.50,1.0) ( -0.10,0.0);
TERM large := (  0.60,0.0) (  1.00,1.0) (  1.00,0.0);
END_FUZZIFY
FUZZIFY p
TERM small := (  0.00,0.0) (  0.00,1.0) (  0.40,0.0);
TERM medium := (  0.10,0.0) (  0.50,1.0) (  0.90,0.0);
TERM large := (  0.60,0.0) (  1.00,1.0) (  1.00,0.0);
END_FUZZIFY
FUZZIFY pd
TERM n_large := ( -1.00,0.0) ( -1.00,1.0) ( -0.60,0.0);
TERM zero := ( -0.40,0.0) (  0.00,1.0) (  0.40,0.0);
TERM medium := (  0.10,0.0) (  0.50,1.0) (  0.90,0.0);
TERM n_medium := ( -0.90,0.0) ( -0.50,1.0) ( -0.10,0.0);
TERM large := (  0.60,0.0) (  1.00,1.0) (  1.00,0.0);
END_FUZZIFY
FUZZIFY pf
TERM small := ( -0.40,0.0) (  0.00,1.0) (  0.40,0.0);
TERM medium := (  0.10,0.0) (  0.50,1.0) (  0.90,0.0);
TERM large := (  0.60,0.0) (  1.00,1.0) (  1.40,0.0);
END_FUZZIFY
```

```
FUZZIFY slA
TERM small := ( -0.40,0.0) (  0.00,1.0) (  0.40,0.0);
TERM medium := (  0.10,0.0) (  0.50,1.0) (  0.90,0.0);
TERM large := (  0.60,0.0) (  1.00,1.0) (  1.40,0.0);
END_FUZZIFY
FUZZIFY smA
TERM smal := ( -0.40,0.0) (  0.00,1.0) (  0.40,0.0);
TERM medium := (  0.10,0.0) (  0.50,1.0) (  0.90,0.0);
TERM large := (  0.60,0.0) (  1.00,1.0) (  1.40,0.0);
END_FUZZIFY
FUZZIFY shA
TERM small := ( -0.40,0.0) (  0.00,1.0) (  0.40,0.0);
TERM medium := (  0.10,0.0) (  0.50,1.0) (  0.90,0.0);
TERM large := (  0.60,0.0) (  1.00,1.0) (  1.40,0.0);
END_FUZZIFY
```

Bloki wyostrzania definiują termy (w postaci singletonów), metodę wyostrzania, domyślną wartość zmiennej i zakres jej zmienności:

```
DEFUZZIFY h
TERM zero  :=  0.00;
TERM medium  :=  0.50;
TERM large  :=  1.00;
TERM big  :=  0.75;
TERM small  :=  0.25;
METHOD : COG;
DEFAULT : NC;
RANGE:=(  0.00..  1.00);
END_DEFUZZIFY
DEFUZZIFY hd
TERM n_large  := -1.00;
TERM zero  :=  0.00;
TERM large  :=  1.00;
TERM big  :=  0.75;
TERM n_big  := -0.75;
TERM n_medium  := -0.50;
TERM n_small  := -0.25;
TERM small  :=  0.25;
TERM medium  :=  0.50;
METHOD : COG;
DEFAULT : NC;
RANGE:=( -1.00..  1.00);
END_DEFUZZIFY
DEFUZZIFY i
TERM zero  :=  0.00;
TERM medium  :=  0.50;
TERM large  :=  1.00;
```

```
TERM small  :=  0.25;
TERM big  :=  0.75;
METHOD : COG;
DEFAULT : NC;
RANGE:=(  0.00..  1.00);
END_DEFUZZIFY
DEFUZZIFY id
TERM n_large  := -1.00;
TERM zero  :=  0.00;
TERM large  :=  1.00;
METHOD : COG;
DEFAULT : NC;
RANGE:=( -1.00..  1.00);
END_DEFUZZIFY
DEFUZZIFY flf
TERM zero  :=  0.00;
TERM medium  :=  0.50;
TERM large  :=  1.00;
TERM big  :=  0.75;
TERM small  :=  0.25;
METHOD : COG;
DEFAULT : NC;
RANGE:=(  0.00..  1.00);
END_DEFUZZIFY
DEFUZZIFY flfd
TERM n_large  := -1.00;
TERM zero  :=  0.00;
TERM large  :=  1.00;
TERM n_big  := -0.75;
TERM n_medium  := -0.50;
TERM n_small  := -0.25;
TERM small  :=  0.25;
TERM medium  :=  0.50;
TERM big  :=  0.75;
METHOD : COG;
DEFAULT : NC;
RANGE:=( -1.00..  1.00);
END_DEFUZZIFY
DEFUZZIFY qA
TERM zero  :=  0.00;
TERM medium  :=  0.50;
TERM large  :=  1.00;
TERM small  :=  0.25;
TERM big  :=  0.75;
METHOD : COG;
DEFAULT : NC;
```

```
RANGE:=(  0.00..  1.00);
END_DEFUZZIFY
DEFUZZIFY so
TERM 0  :=  0.00;
TERM 1  :=  0.25;
TERM 2  :=  0.50;
TERM 4  :=  1.00;
TERM 3  :=  0.75;
METHOD : COG;
DEFAULT : NC;
RANGE:=(  0.00..  1.00);
END_DEFUZZIFY
DEFUZZIFY sv
TERM zero  :=  0.00;
TERM medium  :=  0.50;
TERM large  :=  1.00;
TERM big  :=  0.75;
TERM small  :=  0.25;
METHOD : COG;
DEFAULT : NC;
RANGE:=(  0.00..  1.00);
END_DEFUZZIFY
```

Powyższa część modelu jest ustalona dla robota i nie podlega ingerencji użytkownika (wynika z konstrukcji sensorów i układów wykonawczych).
Blok reguł jest tworzony przez użytkownika robota (terapeutę). Podane poniżej reguły należy traktować jedynie jako ilustrację składni języka FCL:

```
RULEBLOCK No1
AND : MIN;
OR : MAX;
ACCU : MAX;
ACT : MIN;
RULE 1 : IF t IS large AND shA IS medium  THEN ( h IS medium ),( hd IS zero ),
       ( i IS medium ),( id IS zero ),( flf IS zero ) WITH   1.00;
RULE 2 : IF t IS NOT large OR shA IS NOT medium THEN ( h IS large ),( hd IS
       NOT zero ),( i IS zero ),( id IS zero ),( flf IS NOT zero ) WITH 1.00;
RULE 3 : IF p IS medium THEN ( sv IS large ) WITH   1.00;
RULE 4 : IF p IS NOT medium THEN ( sv IS small ) WITH   1.00;
END_RULEBLOCK
OPTION

END_OPTION
END_FUNCTION_BLOCK
```

W celu uzyskania zachowania robota terapeutycznego zamodelowanego powyżej, należy przetworzyć model z języka FCL na struktury danych dopasowane do zaimplementowanej w sterowniku rozmytej maszyny wnioskującej (*Fuzzy Engine*) i załadować je do pamięci sterownika.

3 Implementacja rozmytej maszyny wnioskującej z wykorzystaniem CPU12

Rodzina MC9S12 jest dobrze przystosowana do implementowania sterowników rozmytych. Na uwagę zasługują zwłaszcza specjalne instrukcje maszynowe dostępne w jednostce centralnej CPU12 [7]:

`MEM` (*MEMbership function evaluation*) pozwala w 5 taktach zegara określić stopień przynależności dla funkcji trapezowej określonej przez dwa punkty i dwa nachylenia zboczy.

`REV` (*Rule EValuation*) służy do ewaluacji reguł na zasadzie min-max (koniunkcja jest realizowana przez minimum, a alternatywa – przez maksimum funkcji przynależności poprzedników reguły). Czas wykonania instrukcji zależy od liczby poprzedników i następników (3 takty na poprzednik i następnik). Dostępny jest również jej wariant `REVW` pozwalający zadawać wagi dla poszczególnych reguł.

`WAV` (*Weighted AVerage*) realizuje wyostrzanie wartości wyjściowych. Wylicza ona *SOP (Sum Of Products)* i *SOW (Sum Of Weights)* i umieszcza je w rejestrach tak, by następująca po niej instrukcja dzielenia (`EDIV`) pozwalała uzyskać ostateczny wynik.

Ich wykorzystanie pozwala efektywnie oprogramować rozmywanie zmiennych wejściowych, wnioskowanie rozmyte i wyostrzanie zmiennych wyjściowych, pod warunkiem odpowiedniego zorganizowania bazy wiedzy oraz zmiennych wejściowych, wyjściowych i roboczych.

3.1 Rozmywanie (*Fuzzification*)

Instrukcja `MEM` opiera się na założeniu, że wejściowe zmienne lingwistyczne są opisane w bazie wiedzy trapezoidalnymi funkcjami przynależności określonymi dla każdego termu (rys. 4). Dane wejściowe dla tej instrukcji są następujące:

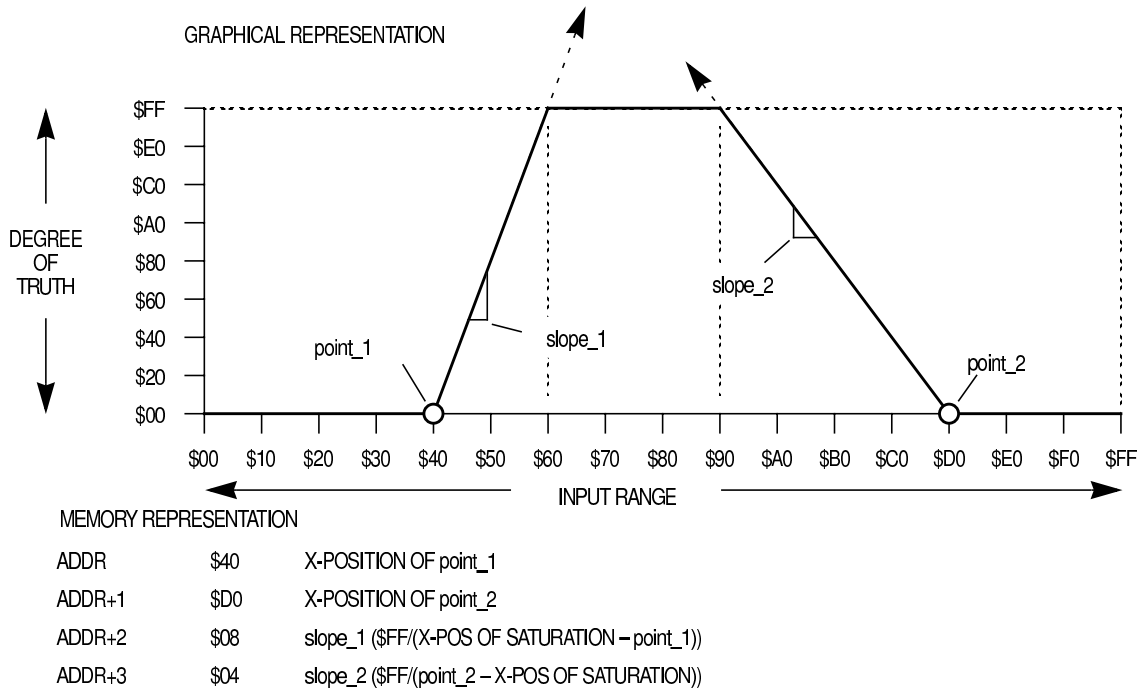
- (ACCA) - wartość wejściowa (ostra) aktualnej zmiennej lingwistycznej;
- (IX) - adres czterobajtowej struktury zawierającej dwa skrajne punkty i dwa nachylenia[1] zboczy trapezu opisującego funkcję przynależności dla aktualnie rozważanego termu aktualnej zmiennej lingwistycznej (w bazie wiedzy, FLASH/EEPROM);
- (IY) - adres bieżącej wartości aktualnie rozważanego termu aktualnej zmiennej lingwistycznej (RAM);

Przy wykonywaniu instrukcji wskaźniki są odpowiednio zwiększane (IX+=4, IY+=1). Naturalne zorganizowanie danych wejściowych (tablica struktur opisujących termy i tablica bieżących wartości termów) pozwala bezpośrednio po sobie powtórzyć `MEM` dla wszystkich kolejnych termów danej zmiennej. Cały proces rozmywania wejść da się zrealizować w postaci dwóch zagnieżdżonych pętli (dla wszystkich zmiennych wejściowych i dla wszystkich ich termów).

3.2 Wnioskowanie (*Rule Evaluation*)

Przy zastosowaniu instrukcji `REV` opis reguły:
if a_1 **is** T_{1i} **and** a_2 **is** T_{2j} **and** ... **then** p_1 **is** T_{1k} **and** p_2 **is** T_{2l} **and** ... ,
musi mieć w bazie wiedzy następującą strukturę:

[1] Zerowa wartość tego parametru oznacza nachylenie nieskończone (zbocze pionowe).



Rysunek 4: Reprezentacja trapezoidalnej funkcji przynależności.

$O_{1i}, O_{2i}, ..., 0xFE, O_{1k}, O_{2l}, ...,$
gdzie O_{mn} oznacza jednobajtowy adres względny (*offset*) w tablicy roboczej dla bieżącej wartości n-tego termu m-tej zmiennej lingwistycznej. Poszczególne reguły są oddzielone bajtem $0xFE$, podobnie jak poprzedniki od następników w jednej regule. Bajt $0xFF$ jest znacznikiem końca bazy reguł. Każda reguła zajmuje więc $N_a + N_p + 2$ bajty (N_a - liczba poprzedników, N_p - liczba następników użytych w regule). Liczba reguł w bazie jest ograniczona tylko wielkością dostępnej pamięci. Przykładowo, w 16kB można zmieścić 1000 reguł, z których każda ma 8 poprzedników i 8 następników. Wykonanie `REV` dla takiej bazy reguł trwa (przy częstotliwości zegara magistrali mikrokontrolera 24MHz) około 2ms[2]. Dane wejściowe dla `REV`:

- (ACCA) = $0xFF$ - robocza wartość min;
- (CCR.V) = 0 - przetwarzanie poprzedników;
- (IX) - adres początku listy reguł w bazie wiedzy (FLASH/EEPROM);
- (IY) - adres początku tablicy bieżących wartości termów wejściowych i wyjściowych (RAM);

Przetwarzanie całej bazy reguł (do znacznika $0xFF$) wykonuje się w trakcie jednokrotnego wywołania `REV`. Wyniki są wpisywane do tablicy chwilowych wartości termów funkcji wyjściowych[3].

UWAGA:
Ze względu na opisany sposób adresowania tablicy bieżących wartości termów, łączna liczba termów w bazie nie może przekroczyć 254 ($0xFE, 0xFF$ są zastrzeżone).

[2] Instrukcja `REV` może być przerywana wielokrotnie w trakcie jej wykonywana, nie powodując opóźnień w pracy układów wejściowo-wyjściowych.
[3] Wartości początkowe termów wyjściowych muszą być wyzerowane.

Instrukcja REVW (*Rule EValuation Weighted*) pozwala wprowadzić wagi reprezentujące "istotność" reguł w bazie. Struktura opisująca regułę dla tej instrukcji zawiera dwubajtowe adresy bezwzględne bieżących wartości poprzedników i następników. Separatorami są odpowiednio słowa: $0xFFFE$, $0xFFFF$. Każdej regule przyporządkowuje się wagę z przedziału $[0,1]$ reprezentowaną na jedym bajcie. W czasie oceniania reguły jej następniki są wartościowane z uwzględnieniem tej wagi. Liczba bajtów opisujących reguły z bazy wiedzy jest około dwukrotnie większa niż w przypadku REV ($2N_a + 2N_p + 4 + 1$). Dane wejściowe dla REV:

- (ACCA) = $0xFF$ - robocza wartość min;
- (CCR.V) = 0 - przetwarzanie poprzedników;
- (CCR.C) = 0/1 - zakaz/zezwolenie używania wag dla reguł;
- (IX) - adres początku listy reguł w bazie wiedzy (FLASH/EEPROM);
- (IY) - adres początku tablicy wag dla reguł w bazie wiedzy (FLASH/EEPROM);

Przetwarzanie całej bazy reguł (do znacznika $0xFF$) wykonuje się w trakcie jednokrotnego wywołania REV. Wyniki są wpisywane do tablicy chwilowych wartości termów funkcji wyjściowych. Podobnie jak dla REV, wartości początkowe termów wyjściowych muszą być wyzerowane.

> **UWAGA:**
> Zastosowanie REVW pozwala zaimplementować dowolną liczbę termów (jedynym ograniczeniem jest wielkość dostępnej pamięci RAM).

3.3 Wyostrzanie (*Defuzzification*)

Zastosowanie instrukcji WAV wymaga reprezentacji wyjściowych zmiennych lingwistycznych w postaci termów o wartościach jednopunktowych (singletonów). Wyliczone w trakcie wnioskowania wartości bieżące dla wszystkich termów danej zmiennej muszą być zorganizowane jako tablica bajtów. Dane wejściowe:

- (ACCB) - liczba termów funkcji wyjściowej;
- (IX) - adres tablicy singletonów (S_i) dla funkcji wyjściowej w bazie wiedzy (FLASH/EEPROM);
- (IY) - adres tablicy wartości bieżących (F_i) dla wszystkich termów danej zmiennej (RAM);

Wynikiem istrukcji WAV są dwie sumy:

$$\sum_{i=1}^{n} S_i F_i$$

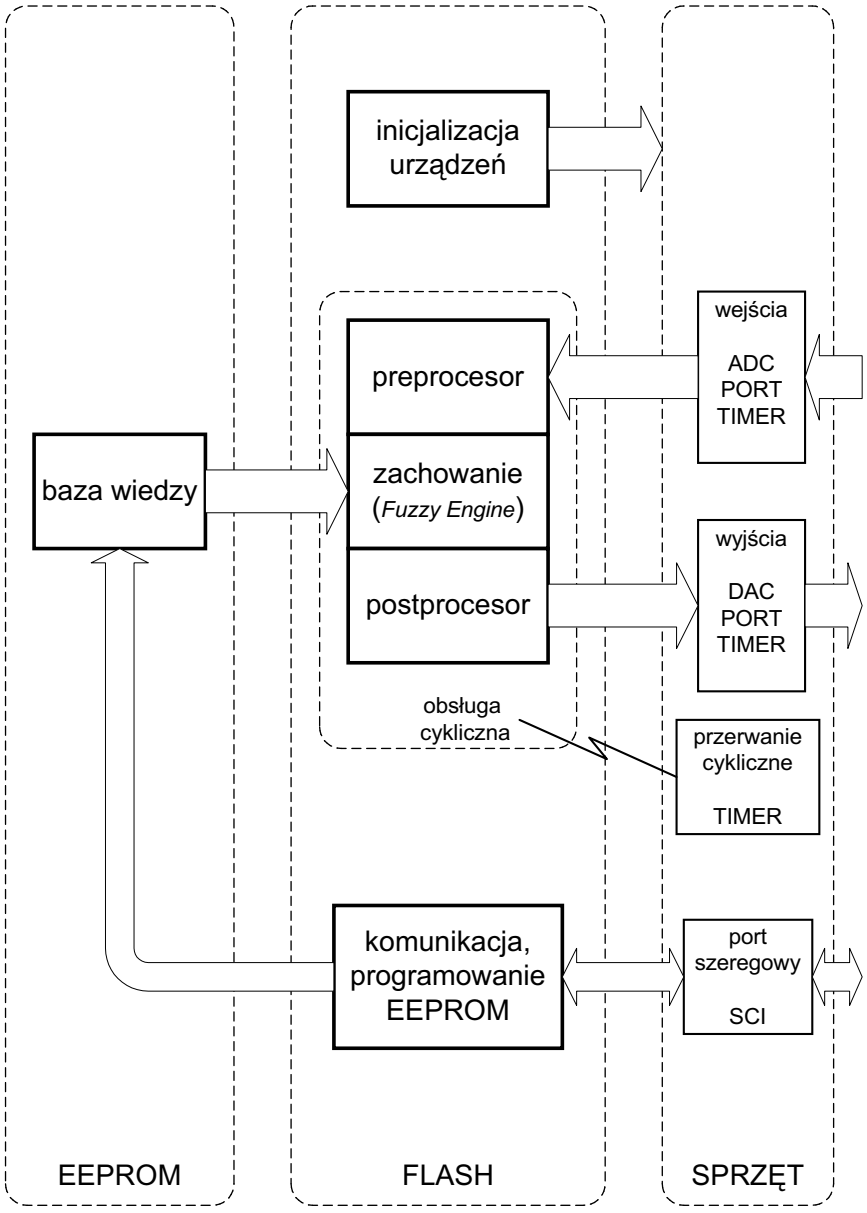
(24-bitowa wartość w rejestrach IY:ACCD),

$$\sum_{i=1}^{n} F_i$$

(16-bitowa wartość w IX), pozwalające uzyskać wynik wyostrzania metodą COGS (*Center Of Gravity for Singletons*) przez bezpośrednie wywołanie instrukcji dzielenia EDIV dającej wynik całkowity w IY i resztę w ACCD.

3.4 Oprogramowanie sterownika

Podstawowe bloki funkcjonalne sterownika kuli interaktywnej przedstawiono na rys. 5. W pamięci FLASH umieszczone jest oprogramowanie i stałe dane, nie podlegające modyfikacjom w trakcie eksploatacji robota.



Rysunek 5: Struktura oprogramowania sterownika.

Procedury inicjalizacji sprzętu zapewniają ustawienie parametrów pracy jednostki centralnej i układów peryferyjnych (wektory przerwań, parametry transmisji, kierunki portów równoległych itd.).
Zasadnicza praca sterownika odbywa się przez cykliczne wykonywanie procedur realizujących od-

czyt i wstępne przetwarzanie sygnałów wejściowych na bodźce (preprocesor), realizację zadanego zachowania (z wykorzystaniem *Fuzzy Engine* i bazy wiedzy) oraz wtórnego przetwarzania reakcji na sygnały wyjściowe (postprocesor). Wywoływanie tych funkcji jest realizowane w procedurze obsługi przerwania od *timer*-a.
Istotnym blokiem jest komunikacja z terapeutą umożliwiająca zmianę definicji zachowania robota (wprowadzenie nowej bazy reguł wnioskowania). Odbywa się ona przez przewodowe łącze szeregowe, które trzeba dołączyć do robota na czas programowania. Reguły wnioskowania znajdują się w pamięci oznaczonej jako EEPROM (w rzeczywistości jest to wydzielony blok pamięci FLASH o adresie 0x8000 i wielkości 16kB), którą można przeprogramowywać do 100.000 razy.

3.5 Generowanie bazy wiedzy na podstawie FCL

Definicja zachowania robota opisana w języku FCL (plik *ra.fcl*) jest konwertowana na postać binarną zgodną z wymogami maszyny wnioskującej opartej na CPU12 w dwóch etapach:

- translacji do języka C (plik *ra.c*):

```
fcl2kb12 ra.fcl ra.c > ra.1 2> ra.2
```

- kompilacji do postaci S-rekordów (plik *ra.s19*):

```
m6811-elf-gcc -mshort -c ra.c
m6811-elf-objcopy -O srec ra.o ra.s19
```

Opracowany translator (*fcl2kb12*) przetwarza postać źródłową rozmytej bazy wiedzy w języku FCL (por. 2.5) na moduł w języku C zawierający:

- definicje stałych:

```
#define USE_WEIGHTS    0 /* WITH flag */
#define USE_NOTS       1 /* NOT flag */
#define IN_VAR_N      13 /* number of input variables */
#define OUT_VAR_N      9 /* number of output variables */
#define RULES_N        5 /* number of rules */
#define IN_TERMS_N    47 /* number of input terms */
#define OUT_TERMS_N   51 /* number of output terms */
```

- pomocnicze parametry rozmytej bazy wiedzy:

```
const t_base_param BaseParam =
     {
       IN_VAR_N,      /* InVarN     */
       OUT_VAR_N,     /* OutVarN    */
       IN_TERMS_N,    /* InTermsN   */
       OUT_TERMS_N,   /* OutTermsN  */
       RULES_N,       /* RulesN     */
       USE_WEIGHTS,   /* UseWeights */
       USE_NOTS,      /* UseNots    */
      };
```

- tablicę parametrów wejść:

```
const t_var_info InpVar[IN_VAR_N] =
      {
        {    3,     0,     1 },    /* t            */
        {    5,    -1,     1 },    /* td           */
        {    3,     0,     1 },    /* txd          */
        {    5,    -1,     1 },    /* ad           */
        {    3,     0,     1 },    /* af           */
        {    3,     0,     1 },    /* rA           */
        {    5,    -1,     1 },    /* rAd          */
        {    3,     0,     1 },    /* p            */
        {    5,    -1,     1 },    /* pd           */
        {    3,     0,     1 },    /* pf           */
        {    3,     0,     1 },    /* slA          */
        {    3,     0,     1 },    /* smA          */
        {    3,     0,     1 }     /* shA          */
      };
```

- tablice definicji trapezoidalnych funkcji przynależności (termów) dla wejść:

```
/* t */
const t_term_info InVar0 [3] = {
         {  26, 230,   3,   3 },    /* medium       */
         { 153, 255,   3,   0 },    /* large        */
         {   0, 102,   0,   3 }     /* small        */
      };

/* td */
const t_term_info InVar1 [5] = {
         {   0,  51,   0,   5 },    /* n_large      */
         {  76, 179,   5,   5 },    /* zero         */
         { 204, 255,   5,   0 },    /* large        */
         {  13, 115,   5,   5 },    /* n_medium     */
         { 140, 242,   5,   5 }     /* medium       */
      };

/* txd */
const t_term_info InVar2 [3] = {
         {   0, 113,   4,   4 },    /* small        */
         {  71, 184,   4,   4 },    /* medium       */
         { 142, 255,   4,   4 }     /* large        */
      };

/* ad */
const t_term_info InVar3 [5] = {
         {   0,  51,   0,   5 },    /* n_large      */
```

```
         {  76, 179,   5,   5 },    /* zero         */
         { 204, 255,   5,   0 },    /* large        */
         {  13, 115,   5,   5 },    /* n_medium     */
         { 140, 242,   5,   5 }     /* medium       */
      };

/* af */
const t_term_info InVar4 [3] = {
         {   0, 113,   4,   4 },    /* small        */
         {  71, 184,   4,   4 },    /* medium       */
         { 142, 255,   4,   4 }     /* large        */
      };

/* rA */
const t_term_info InVar5 [3] = {
         {   0, 113,   4,   4 },    /* small        */
         {  71, 184,   4,   4 },    /* medium       */
         { 142, 255,   4,   4 }     /* large        */
      };

/* rAd */
const t_term_info InVar6 [5] = {
         {   0,  51,   0,   5 },    /* n_large      */
         {  76, 179,   5,   5 },    /* zero         */
         { 140, 242,   5,   5 },    /* medium       */
         {  13, 115,   5,   5 },    /* n_medium     */
         { 204, 255,   5,   0 }     /* large        */
      };

/* p */
const t_term_info InVar7 [3] = {
         {   0, 102,   0,   3 },    /* small        */
         {  26, 230,   3,   3 },    /* medium       */
         { 153, 255,   3,   0 }     /* large        */
      };

/* pd */
const t_term_info InVar8 [5] = {
         {   0,  51,   0,   5 },    /* n_large      */
         {  76, 179,   5,   5 },    /* zero         */
         { 140, 242,   5,   5 },    /* medium       */
         {  13, 115,   5,   5 },    /* n_medium     */
         { 204, 255,   5,   0 }     /* large        */
      };

/* pf */
```

```
const t_term_info InVar9 [3] = {
         {   0, 113,   4,   4 },    /* small        */
         {  71, 184,   4,   4 },    /* medium       */
         { 142, 255,   4,   4 }     /* large        */
      };

/* slA */
const t_term_info InVar10 [3] = {
         {   0, 113,   4,   4 },    /* small        */
         {  71, 184,   4,   4 },    /* medium       */
         { 142, 255,   4,   4 }     /* large        */
      };

/* smA */
const t_term_info InVar11 [3] = {
         {   0, 113,   4,   4 },    /* smal         */
         {  71, 184,   4,   4 },    /* medium       */
         { 142, 255,   4,   4 }     /* large        */
      };

/* shA */
const t_term_info InVar12 [3] = {
         {   0, 113,   4,   4 },    /* small        */
         {  71, 184,   4,   4 },    /* medium       */
         { 142, 255,   4,   4 }     /* large        */
      };
```

- tablicę parametrów wyjść:

```
const t_var_info OutVar[OUT_VAR_N] =
      {
        {    5,     0,     1 },    /* h            */
        {    9,    -1,     1 },    /* hd           */
        {    5,     0,     1 },    /* i            */
        {    3,    -1,     1 },    /* id           */
        {    5,     0,     1 },    /* flf          */
        {    9,    -1,     1 },    /* flfd         */
        {    5,     0,     1 },    /* qA           */
        {    5,     0,     1 },    /* so           */
        {    5,     0,     1 }     /* sv           */
      };
```

- tablice definicji singletonowych funkcji przynależności (termów) dla wejść:

```
/* h */
const unsigned char OutVar0 [5] = {
          0,    /* zero         */
```

```
         128,    /* medium       */
         255,    /* large        */
         191,    /* big          */
          64     /* small        */
      };

/* hd */
const unsigned char OutVar1 [9] = {
           0,    /* n_large      */
         128,    /* zero         */
         255,    /* large        */
         223,    /* big          */
          32,    /* n_big        */
          64,    /* n_medium     */
          96,    /* n_small      */
         159,    /* small        */
         191     /* medium       */
      };

/* i */
const unsigned char OutVar2 [5] = {
           0,    /* zero         */
         128,    /* medium       */
         255,    /* large        */
          64,    /* small        */
         191     /* big          */
      };

/* id */
const unsigned char OutVar3 [3] = {
           0,    /* n_large      */
         128,    /* zero         */
         255     /* large        */
      };

/* flf */
const unsigned char OutVar4 [5] = {
           0,    /* zero         */
         128,    /* medium       */
         255,    /* large        */
         191,    /* big          */
          64     /* small        */
      };

/* flfd */
const unsigned char OutVar5 [9] = {
```

```
          0,    /* n_large      */
        128,    /* zero         */
        255,    /* large        */
         32,    /* n_big        */
         64,    /* n_medium     */
         96,    /* n_small      */
        159,    /* small        */
        191,    /* medium       */
        223     /* big          */
      };

/* qA */
const unsigned char OutVar6 [5] = {
          0,    /* zero         */
        128,    /* medium       */
        255,    /* large        */
         64,    /* small        */
        191     /* big          */
      };

/* so */
const unsigned char OutVar7 [5] = {
          0,    /* 0            */
         64,    /* 1            */
        128,    /* 2            */
        255,    /* 4            */
        191     /* 3            */
      };

/* sv */
const unsigned char OutVar8 [5] = {
          0,    /* zero         */
        128,    /* medium       */
        255,    /* large        */
        191,    /* big          */
         64     /* small        */
      };
```

- listę reguł w jednej z dwóch form (w zależności od wartości `USE_WEIGHTS`):
  - dla instrukcji `REVW`:

```
const unsigned char Weights[RULES_N] = {
      255,
      255,
      255,
      255,
      255
```

```
    };

const unsigned int Rules[] = {
             /* IF */
      0x8002, /* t IS large */
      0x805A, /* shA IS medium */
      0xFFFE, /* THEN */
      0x805F, /* h IS medium */
      0x8064, /* hd IS zero */
      0x806D, /* i IS medium */
      0x8072, /* id IS zero */
      0x8074, /* flf IS zero */
      0xFFFE,
             /* IF */
      0x8003, /* t IS NOT large */
      0xFFFE, /* THEN */
      0x8060, /* h IS large */
      0x8064, /* hd IS zero */
      0x806C, /* i IS zero */
      0x8072, /* id IS zero */
      0x8074, /* flf IS zero */
      0xFFFE,
             /* IF */
      0x805B, /* shA IS NOT medium */
      0xFFFE, /* THEN */
      0x8060, /* h IS large */
      0x8064, /* hd IS zero */
      0x806C, /* i IS zero */
      0x8072, /* id IS zero */
      0x8074, /* flf IS zero */
      0xFFFE,
             /* IF */
      0x8038, /* p IS medium */
      0xFFFE, /* THEN */
      0x808E, /* sv IS large */
      0xFFFE,
             /* IF */
      0x8039, /* p IS NOT medium */
      0xFFFE, /* THEN */
      0x8090, /* sv IS small */

      0xFFFF
    };
```

– dla instrukcji `REV`:

```
const unsigned char Rules[] = {
            /* IF */
```

```
  0x02, /* t IS large */
  0x5A, /* shA IS medium */
  0xFE, /* THEN */
  0x5F, /* h IS medium */
  0x64, /* hd IS zero */
  0x6D, /* i IS medium */
  0x72, /* id IS zero */
  0x74, /* flf IS zero */
  0xFE,
        /* IF */
  0x03, /* t IS NOT large */
  0xFE, /* THEN */
  0x60, /* h IS large */
  0x64, /* hd IS zero */
  0x6C, /* i IS zero */
  0x72, /* id IS zero */
  0x74, /* flf IS zero */
  0xFE,
        /* IF */
  0x5B, /* shA IS NOT medium */
  0xFE, /* THEN */
  0x60, /* h IS large */
  0x64, /* hd IS zero */
  0x6C, /* i IS zero */
  0x72, /* id IS zero */
  0x74, /* flf IS zero */
  0xFE,
        /* IF */
  0x38, /* p IS medium */
  0xFE, /* THEN */
  0x8E, /* sv IS large */
  0xFE,
        /* IF */
  0x39, /* p IS NOT medium */
  0xFE, /* THEN */
  0x90, /* sv IS small */
  0xFF  /* End Of Rules */
};
```

Pomocnicze struktury danych są zdefiniowane w stałym pliku nagłówkowym *kb12.h* zawierającym:

- definicję struktury opisującej zmienną (we lub wy):

```
typedef struct
{
  unsigned char terms_nbr;
  short val_min;
```

```
  short val_max;
} t_var_info;
```

- definicję struktury opisującej trapezoidalną funkcję przynależności:

```
typedef struct
{
  unsigned char point1;
  unsigned char point2;
  unsigned char slope1;
  unsigned char slope2;
} t_term_info;
```

  złożonej z dwóch punktów podstawy i dwóch nachyleń boków trapezu

- definicję struktury opisującej parametry bazy wiedzy niezbędne do pracy maszyny wnioskującej:

```
typedef struct
{
  const short  InVarN;
  const short  OutVarN;
  const short  InTermsN;
  const short  OutTermsN;
  const short  RulesN;
  const char   UseWeights;
  const char   UseNots;
} t_base_param;
```

  gdzie `InVarN` i `OutVarN` są liczbami wejść i wyjść, `InTermsN` i `OutTermsN` są łącznymi liczbami termów wejściowych i termów wyjściowych, `RulesN` jest liczbą reguł w wynikowej bazie wiedzy, `UseWeights` jest flagą decydującą o wyborze formatu reguł (dla `REV` lub `REVW`), `UseNots` oznacza występowanie w poprzednikach negacji.

Plik ten jest wspólny dla *fcl2kb12* i maszyny wnioskującej.
Do kompilacji wytworzonego przez *fcl2kb12* pliku źródłowego można wykorzystać dowolny kompilator ANSI C pozwalający wytworzyć moduł binarny w postaci szesnastkowej (tzw. S-rekordy). W opisywanym przypadku uzyto pakietu *m6811-elf-gnu-toolchain* dostępnego na zasadach licencji GPL (GNU).
Kompilacja do pliku relokowalnego typu *.o* wykonywana jest poleceniem:

```
m6811-elf-gcc -m short -c ra.c
```

gdzie opcja `-m short` zapewnia spakowaną do bajtów postać tablic i sruktur.
Przetworzenie pliku relokowalnego (typu *.o*) na postać binarną (*.s19*) wykonywane jest poleceniem:

```
m6811-elf-objcopy -O srec ra.o ra.s19
```

gdzie opcja `-O srec` zapewnia format pliku wyjściowego (S-rekordy), który zawiera informacje o adresie ładowania danych do pamięci mikrokontrolera oraz sumy kontrolne pozwalające weryfikować poprawność transmisji.

4 Interfejs użytkownika

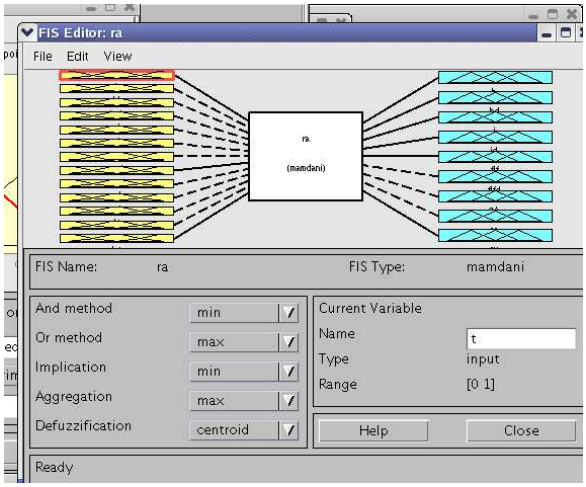
Jedną z fundamentalnych własności robota kulistego jest możliwość dostosowywania jego zachowania stosownie do wymagań pacjenta i potrzeb terapeuty, w wystarczająco szerokim zakresie. Koniecznym elementem gwarantującym tę własność jest interfejs, który za pośrednictwem komputera pozwala na dwustronną komunikację pomiędzy człowiekiem i robotem.
Szczegółowa specyfikacja interfejsu została sformułowana w pracy [3]. Decydujący wpływ na kształt interfejsu miały trzy elementy: postulat o samodzielnej obsłudze robota przez terapeutę bez wsparcia ze strony technika [3], przyjęcie rozwiązania, w którym sterowanie robotem jest oparte na mikrokontrolerze z rodziny HC9S12 z oprogramowaniem typu *fuzzy engine* [23, 14], oraz włączenie do specyfikacji normy IEC 1131 o programowalnych sterownikach fuzzy [12]. Pośredni wpływ miała filozofia *User Centered Design* Katza i Hassa dyskutowana w pracy [1].
Zgodnie z [3] interfejs użytkownika powinien składać się z trzech zasadniczych elementów: modułu profilu dziecka, modułu programowania oraz modułu symulacji/animacji robota. Na pierwszy z modułów składają się te elementy interfejsu, które są mniej lub bardziej potrzebne do przeprowadzenia procesu terapeutycznego z wykorzystaniem robota na konkretnym dziecku. Drugi z modułów umożliwia terapeucie programowanie robota przy użyciu reguł na niskim i wysokim poziomie. Trzeci moduł pozwala na weryfikację nowych programów bez udziału robota metodami symulacyjnymi. Łatwo zauważyć, że pierwsze dwa moduły są niezbędne. Trzeci może być traktowany jako nadmiarowy na etapie prac o charakterze pilotażowo prototypowym.
Przyjęcie normy IEC 1131 pozwoliło na daleko posunięte uniezależnienie rozwiązań stosowanych w interfejsie od rozwiązań przyjętych w trakcie tworzenia *fuzzy engine*. W szczególności, twórcy interfejsu nie mają żadnych ograniczeń na środowisko programistyczne. Prototyp interfejsu i wstępne badania można zrealizować np. na bazie oprogramowania Matlab.[4]
Istnieją przynajmniej trzy ważne przesłanki przemawiające za zastosowaniem Matlaba do budowy prototypu interfejsu. Oprogramowanie to oferuje rozbudowane środowisko GUIDE umożliwiające projektowanie i implementację własnych interfejsów graficznych w oparciu o elementy aktywne grafiki uchwytów (Handle Graphics) i technikę wywołań zwrotnych (callbacks) [21, 20]. W systemie występuje biblioteka *Fuzzy Logic* umożliwiająca efektywne modelowanie technikami logiki rozmytej i analizę takich modeli [11]. Bogaty zestaw funkcji matematycznych oraz grafiki komputerowej i wizualizacji daje możliwości szybkiej realizacji modułu animacji/symulacji robota.
Najszybszym sposobem uzyskania prostego interfejsu robota kulistego jest adaptacja matlabowskiej biblioteki *Fuzzy Logic* poprzez uzupełnienie jej o funkcje dostosowujące do standardów normy IEC 1131. W ten sposób można częściowo zrealizować moduł programowania. Funkcje dostosowujące zostały zebrane w bibliotekach `fuzzyfis`, `fuzzyfcl` i opisane w rozdziale 4.1. Prototyp interfejsu, o nazwie `Szarik`, realizujący w pełni założenia z [3] opisano w rozdziale 4.2.1. Zamieszczono tam najistotniejsze informacje o niezbędnych elementach interfejsu: module profilu dziecka i module programowania. Odnośnie szczegółów implementacyjnych odsyłamy czytelnika do pracy [6]. Podsumowanie i sugestie na temat kierunków rozwojowych interfejsu zamieszczono w rozdziale 4.3.

4.1 Biblioteki: fuzzyfis i fuzzyfcl

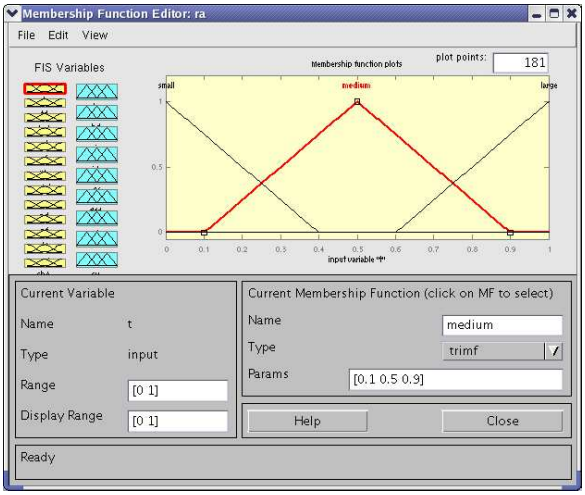
Biblioteka *Fuzzy Control* Toolbox w Matlabie oferuje trzy rodzaje narzędzi: funkcje linii komend, interaktywne narzędzia graficzne i bloki Simulinka [11]. Pierwsze dwie grupy oferują podobne

[4] W pracy wykorzystano środowisko MATLAB udostępnione przez WCSS.

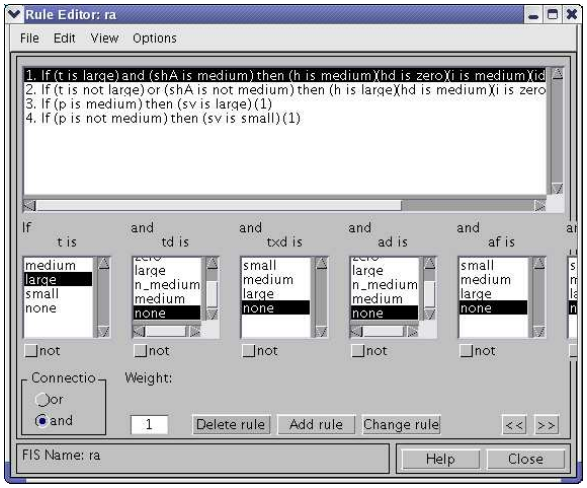
możliwości, przy czym narzędzia graficzne są wygodniejsze a te adresowane linii komend pozwalają na rozwiązywanie bardziej podstawowych problemów oraz pisanie skryptów.
Z punktu widzenia robota kulistego najważniejszymi narzędziami graficznymi są: edytor systemu wnioskującego, edytor funkcji przynależności i edytor reguł. Przykładowe zrzuty z ekranu poszczególnych edytorów przedstawiono na rys. 6. Przy ich pomocy można tworzyć i zmieniać wszystkie elementu modelu rozmytego wnioskowania robota kulistego, w szczególności parametry funkcji przynależności i reguły wnioskowania.



(a) edytor systemu wnioskującego



(b) edytor funkcji przynależności



(c) edytor reguł

Rysunek 6: Narzędzia graficzne biblioteki *Fuzzy Control*.

Model utworzony w edytorze jest zapisywany na matlabowej *strukturze fis* i przechowywany w przstrzeni roboczej pod pewną zmienną. Struktura ta może być zapisana na pliku tekstowym w *formacie fis*. Zarówno struktura jak i format *fis* nie są zgodne ze specyfikacją normy IEC 1131.
Dostosowanie biblioteki *Fuzzy Control* do normy IEC 1131 może być zrealizowane przez

- specyfikację, na gruncie normy IEC 1131, nowej struktury w Matlabie i nowego formatu zapisu na pliku tekstowym, odpowiednio struktury *fcl* i formatu *fcl*;
- utworzenie funkcji konwertujących pomiędzy formatami *fis* i *fcl* oraz zapisujących (wczytujących) strukturę *fis* na (z) pliku w formacie *fis*.

Zgodnie z powyższą wytyczną powstały dwie biblioteki: `fuzzyfcl` i `fuzzyfis`. Pierwszą z nich tworzą dwie funkcje: `fis2fcl` i `writefcl`. Druga składa się również z dwóch funkcji: `fcl2fis` i `readfcl`. Funkcje `writefcl` i `readfcl` są odpowiednikami `writefis` i `readfis` z biblioteki *Fuzzy Control*. Specyfikacja poszczególnych funkcji jest następująca.

mfcl=fis2fcl(mfis, nastepny_konwerter)

Rozmyty model wnioskowania zapisany na strukturze fis i przypisany zmiennej mfis przypisywany jest zmiennej mfcl po uprzednim osadzeniu na strukturze fcl. Konwersja uwzględnia różnice pomiędzy oboma standardami zasygnalizowane przez tabelę 1. Tylko w skrajnych przypadkach wykonanie procedury nie kończy się sukcesem.

Drugi argument funkcji może przyjąć jedną z dwóch wartości: 'unspecified' (domyślnie) lub 'fcl2kb12'. Jeśli jest to 'fcl2kb12' wówczas podczas konwersji do formatu fcl brane są pod uwagę ograniczenia nakładane przez asembler mikrokontrolera MC68HC12.

writefcl(mfcl,nazwa_pliku)

Rozmyty model wnioskowania zapisany na strukturze fcl i przypisany zmiennej mfcl zapisywany jest na pliku `nazwa_pliku` w formacie fcl.

fis=fcl2fis(fcl)

Rozmyty model wnioskowania zapisany na strukturze fcl i przypisany zmiennej mfcl przypisywany jest zmiennej mfis po uprzednim osadzeniu na strukturze fis.

readfcl(fcl,nazwa_pliku)

Rozmyty model wnioskowania zapisany na pliku `nazwa_pliku` w formacie fcl wczytywany jest do przestrzeni roboczej Matlaba na strukturę fcl i przypisywany zmiennej mfcl.

Biblioteki *Fuzzy Logic*, `fuzzyfis`, `fuzzyfcl` mogą być postrzegane jako pilotażowy moduł programowania robota kulistego. Umożliwiają on budowę i modyfikację wszystkich elementów rozmytego modelu wnioskowania robota kulistego, począwszy od definiowania zmiennych lingwistycznych i skończywszy na operacjach na regułach wnioskowania. Taki moduł okazał się cennym narzędziem na etapie tworzenia robota. W kontekście docelowego interfejsu terapeuty taki moduł może być jedynie przyczynkiem do dalszych prac. Podstawowe wady, które należy wyeliminować to:

- pełna dostępność do wszystkich elementów modelu poza regułami wnioskowania;
- możliwość definiowania reguł tylko niskim poziomie, bez możliwości tworzenia i wykorzystywania makr obejmujących dłuższe fragmenty warunku bądź konkluzji reguły zawierające nawiasy i wszystkie rodzaje operatorów logicznych;
- brak integracji z pozostałymi wymaganymi modułami interfejsu terapeuty.

	fis	fcl
funkcje przynależności - wejście	trimf	points
	trapmf	points
	gbellmf, gaussmf, gauss2mf, sigmf, dsigmf, psigmf, pimf, smf, zmf	brak odpowiednika
funkcje przynależności - wyjście	brak odpowiednika	singleton
	trimf, trapmf, gbellmf, gaussmf, gauss2mf, sigmf, dsigmf, psigmf, pimf, smf, zmf	brak odpowiednika
metody rozmywania	centroid	COG, COGS
	bisector	COA
	mom	LM
	lom	RM
	som	LM
metody akumulacji	max	MAX
	sum	BSUM
	brak odpowiednika	NSUM
metody aktywacji	min	MIN
	prod	PROD
operatory	max	MAX
	probor	ASUM
	min	MIN
	prod	PROD
	brak odpowiednika	BSUM, BDIF
negacja w warunku reguły	TAK	TAK
negacja w konkluzji reguły	TAK	NIE
waga reguły	TAK	TAK
mieszane operatory w warunku reguły	NIE	TAK
OR w konkluzji reguły	TAK	NIE

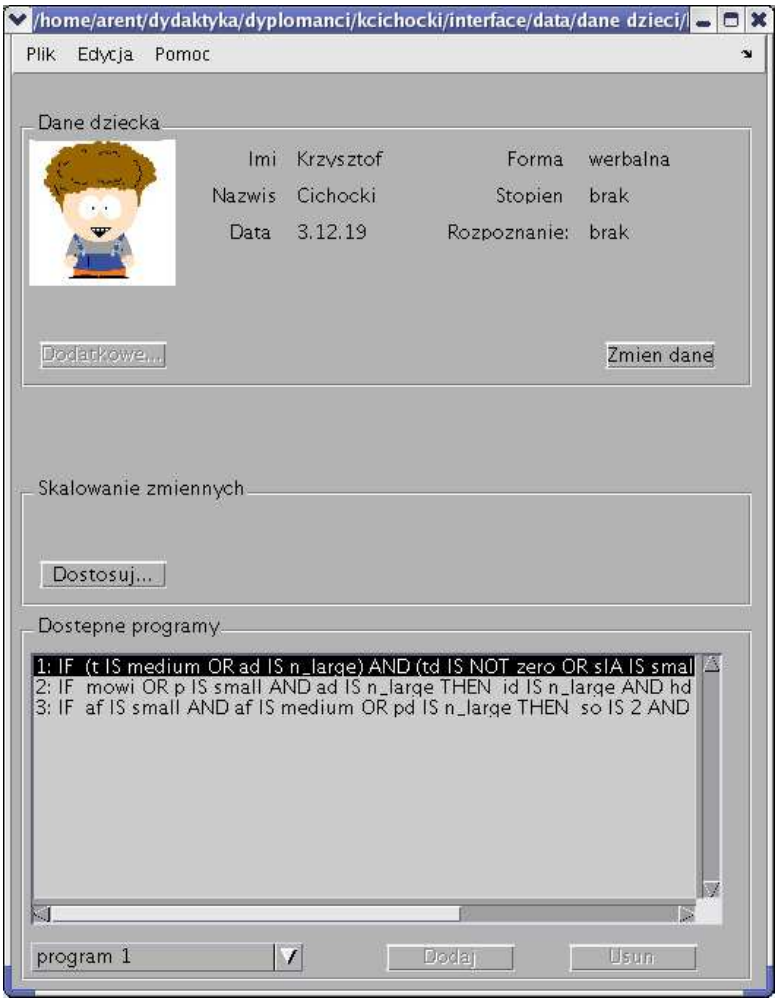
Tablica 1: Ważniejsze podobieństwa i różnice w standardach fis i fcl.

4.2 Szarik

Interfejs Szarik jest pilotażową wersją interfejsu użytkownika robota kulistego. W jego skład wchodzą moduł profilu dziecka i moduł programowania, które w znacznym stopniu spełniają specyfikacje sformułowane w [3]. W niniejszym rozdziale nakreślono koncepcje i możliwości obu modułów. Szczegółowe informacje nt. implementacji można znaleźć w pracy [6].

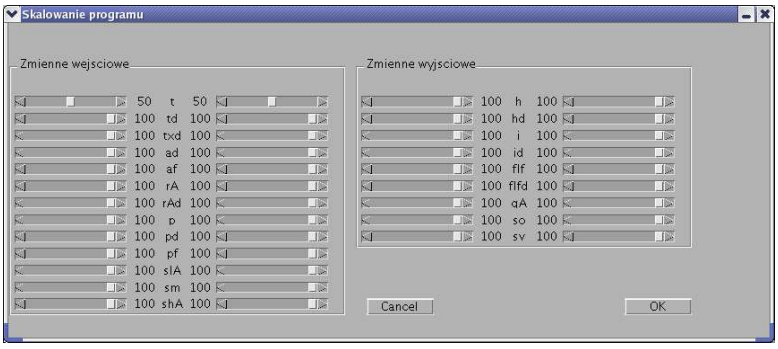
4.2.1 Moduł profilu dziecka

Moduł profilu dziecka jest tym elementem interfejsu, który jest używany przez terapeutę w trakcie pracy z dzieckiem. W szczególności udostępnia on podstawowe dane o dziecku, programy wykorzystywane w pracy z tym dzieckiem oraz indywidualne ustawienia wybranych parametrów robota dla danego dziecka. Okno modułu profilu dziecka przedstawiono na rusunku 7.
Podstawowe dane o dziecku to imię, nazwisko, wiek, forma komunikacji, stopień upośledzenia i rozpoznanie, którym towarzyszy zdjęcie dziecka.



Rysunek 7: Szarik: moduł profilu dziecka.

Ponieważ robot kulisty ze swojej natury wchodzi w interakcje z dzieckiem, jego wrażliwość na bodźce zewnętrzne musi być dostosowana to dziecka. Z tego powodu moduł profilu dziecka daje terapeucie możliwość zmiany granic przedziałów, na których zdefiniowane są funkcje przynależności stowarzyszone z poszczególnymi zmiennymi lingwistycznymi. Do tego celu służy dedykowane okno z suwakami przedstawione na rys. 8. Zmiana szerokości przedziału powoduje automatyczne przeskalowanie współrzędnych parametrów funkcji przynależności.
Najważniejszym elementem modułu profilu dziecka jest lista programów wykorzystywanych przez terapeutę w pracy z dzieckiem. Przez program rozumiemy zbiór reguł specyfikujących zachowanie się robota. Terapeuta ma możliwość przeglądu i podglądu poszczególnych programów oraz ładowania wybranego do pamięci robota kulistego. Z listą programów stowarzyszony jest pełen wachlarz dostępnych operacji: dodawanie nowych, modyfikacje, usuwanie.
Wszystkie dane widoczne w oknie modułu przechowywane są w binarnym pliku z rozszerzeniem `.ch` w kartotece `szarik/dane/children`. Zapisujemy je na pliku wybierając podmenu `Zapisz` w menu `Plik`. Ładowanie programu do pamięci robota inicjalizowane jest przez wybór podmenu `Eksport do pliku` w menu `Plik`. Wyszczególnione przez terapeutę programy można zapisać na pliku w formatach `.fis`, `.fcl`, `.c`, `.s19` wybierając w menu `Plik` podmenu `Eksport do pliku`.

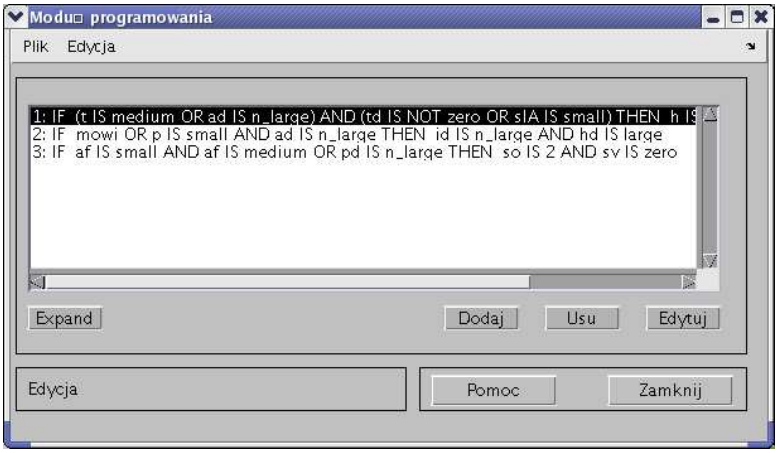


Rysunek 8: Skalowanie przedziału wartości zmiennych lingwistycznych w Szariku.

Wprowadzenie danych nowego dziecka do Szarika rozpoczynamy wybierając w menu `Plik` podmenu `Nowy`. Przejście do pracy z innym dzieckiem odbywa się przez wybór podmenu `Otwórz` w menu `Plik` Edycja wyspecyfikowanego programu bądź tworzenie nowego wymaga wyboru z menu `Edycja` podmenu `Programowanie` .

4.2.2 Moduł programowania

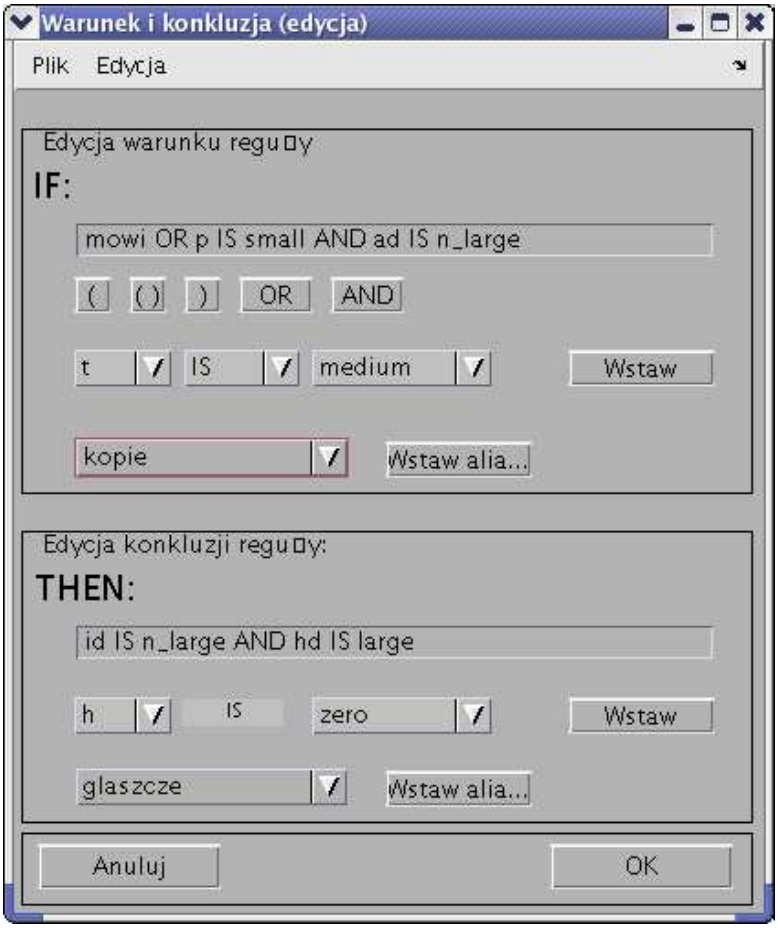
Programowanie robota kulistego sprowadza się do wygenerowania zbioru reguł zgodnych ze standardem IEC 1131. Specyfikacja [3] wprowadza dwa poziomy programowania w interfejsie terapeuty: elementarny i terapeuty. Na pierwszym z nich programista tworzy reguły posługując się wyłącznie zmiennymi lingwistycznymi, termami i operatorami. Poziom terapeuty umożliwia tworzenie aliasów, obejmujących wyrażenia logiczne, i stosowanie tych aliasów na równi z wyrażeniami zgodnymi z normą IEC 1131, z wykorzystaniem nawiasów i operatorów przeczących.
Okno główne modułu programowania przedstawiono na rys. 9. Za jego pośrednictwem można przeglądać listę istniejących programów, otwierać wybrane, inicjalizować proces modyfikacji, tworzyć nowe programy i zapisywać je na pliku. Programy przechowywane są w binarnych plikach z rozszerzeniem `.rul` w kartotece `szarik/dane/rules`. Możne je eksportować do plików w formatach: `.fis`, `.fcl`, `.c`, `.s19`.



Rysunek 9: Szarik: moduł programowania.

Edycja reguł odbywa się przy użyciu okna zamieszczonego na rys. 10. Aliasy dostępne są w roz-

wijanych listach na dole każdego z obszarów IF i THEN, po uprzednim wczytaniu z pliku. Każde wyedytowane wyrażenie jest analizowane i przekształcane do zbioru reguł w postaci dopuszczalnej przez mikrokontroler. Szczegóły związane z implementacją algortymu można znaleźć w pracy [6].



Rysunek 10: Edycja reguły w Szariku.

Okna dedykowane edycji aliasów przedstawione są na rys. 11. Podobnie jak w przypadku reguł można te elementy przeglądać, edytować, definiować, zapisywać do pliku i wczytywać z pliku. Aliasy zapisywane są w binarnych plikach z rozszerzeniem `.als` w kartotece `szarik/dane/aliases`.

4.3 Uwagi

W rozdziale przedstawiono pilotażową wersję interfejsu terapeuty Szarik oraz adaptację biblioteki *Fuzzy Logic* do formy modułu programowania. Na obecnym etapie badań Szarik powinien zostać poddany ocenie terapeutów. Dopiero na podstawie ich doświadczeń i uwag będzie można udoskonalić ten program oraz napisać ponownie, w języku programwania właściwszym dla wersji docelowej.
Zaadaptowana biblioteka *Fuzzy Logic* jest bardzo przydatna na poziomie realizacji i wstępnych testów konstrukcji robota. Umożliwia ona szybki i łatwy dostęp do wszystkich elementów rozmytego modelu wnioskowania robota kulistego, co odgrywa istotną rolę w tej fazie prac.

(a) podgląd aliasu

(b) edycja aliasów

Rysunek 11: Aliasy.

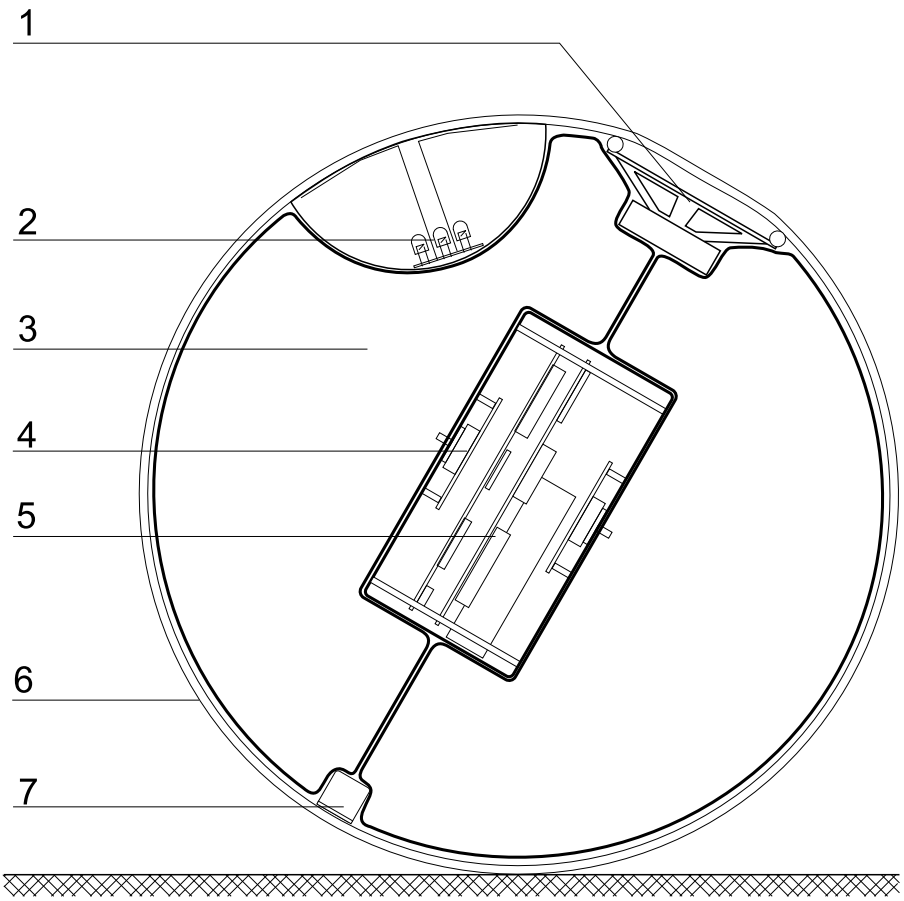
5 Konstrukcja robota

Do konstrukcji obudowy robota została wykorzystana piłka wykonana z półprzezroczystego, elastycznego tworzywa sztucznego (rys. 12).
Wewnątrz obudowy umieszczono wszystkie układy niezbędne do autonomicznej pracy robota:

1. głośnik
2. źródło światła i czujnik zbliżeniowy (1 z 8)
3. dętka (1 z 2)
4. czujnik ciśnienia (1 z 2)
5. sterownik ze źródlem zasilania i pozostałymi czujnikami w obudowie z tworzywa sztucznego
6. obudowa zewnętrzna (piłka półprzezroczysta)
7. mikrofon

5.1 Sensory

Jako sensory zbliżeniowe zastosowano układy wykorzystujące pole magnetyczne. Mają one większy zasięg działania niż sensory optyczne, zatem przy ich wykorzystaniu możliwe było zmniejszenie całkowitej liczby sensorów do 8. Jeśli doświadczenia wykażą potrzebę umieszczenia na powierzchni robota większej liczby sensorów w celu bardziej dokładnego odtworzenia kształtu powierzchni styku, wykorzystane zostaną sensory optyczne działające na zasadzie światła odbitego.
Zastosowanie akcelerometrów i żyroskopów umożliwia rejestrację parametrów położenia i orientacji robota (przyspieszeń i prędkości ruchu obrotowego), czyli takich zdarzeń jak toczenie robota, potrząsanie nim czy rzucanie. Wykorzystane tu zostały dwuosiowe mikromechaniczne akcelerometry ADXL202 [2]. Parametry tych układów zostały zebrane w tabeli 2. Układy te umożliwiają dwuosiowy pomiar przyspieszeń w zakresie $\pm$2g. Zastosowanie dwóch takich układów umożliwia



Rysunek 12: Schemat budowy robota

Parametr	Wartość
zakres pomiarowy	$\pm$2g
czułość	2mg
pasmo przenoszenia	60Hz
wytrzymałość na udary	1000g
wymiary	5mm/5mm/2mm

Tablica 2: Podstawowe parametry akcelerometru ADXL202

badanie przyspieszeń działających na robota w każdym możliwym kierunku (dla uproszczenia zastosowano dla wektora przyspieszenia normę Manhattanu wyznaczaną sprzętowo).
Dodatkowym elementem pozwalającym na wykrycie toczenia robota są trzy jednoosiowe żyroskopy ENC-03J [19]. Parametry żyroskopów tego typu zostały zebrane w tabeli 3. Norma wektora prędkości

Parametr	Wartość
zakres pomiarowy	$\pm$300deg/s
czułość	0.67mV/deg/s
pasmo przenoszenia	50Hz
liniowość	$\pm$5% pełnej skali
wymiary	7mm/12.2mm/2.6mm

Tablica 3: Podstawowe parametry żyroskopu ENC-03J

obrotowej jest znajdowana analogicznie jak w przypadku przyspieszenia.
Czujniki ciśnienia wewnątrz szczelnej i elastycznej obudowy robota umożliwia rejestrację takich zdarzeń jak: odbijanie się robota od podłoża i przeszkód oraz nacisk na obudowę robota, ponieważ odkształcenie mechaniczne obudowy zmieniają ciśnienie panujące w jej wnętrzu (a dokładniej - w dętkach wypełniających wolną przestrzeń wewnątrz kuli). Badanie ciśnienia wewnątrz obudowy umożliwia również odróżnienie od siebie takich zdarzeń jak swobodne odbijanie się robota od podłoża i potrząsania robotem. Zarówno w pierwszym jak i w drugim przypadku przyspieszenia działające na robota mierzone akcelerometrami będą podobne. Natomiast ciśnienie wenątrz obudowy będzie pulsować w pierwszym przypadku, a w drugim przypadku będzie się zmieniać w niewielkim stopniu. Do pomiaru ciśnienia zastosowano czujnik ciśnienia absolutnego MPX4200 [18]. Parametry tego czujnika zostały zebrane w tabeli 4.

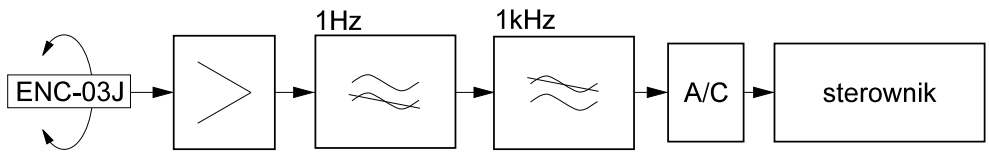
Parametr	Wartość
zakres pomiarowy	20kPa-200kPa
czułość	26mV/kPa
czas odpowiedzi	1ms
liniowość	$\pm$1.5% pełnej skali
wymiary	16mm/16mm/5.6mm

Tablica 4: Podstawowe parametry czujnika ciśnienia MPX4200

Rejestracja dźwięku z otoczenia odbywa się przy użyciu mikrofonów elektretowych umieszczonych przy powierzchni obudowy robota.

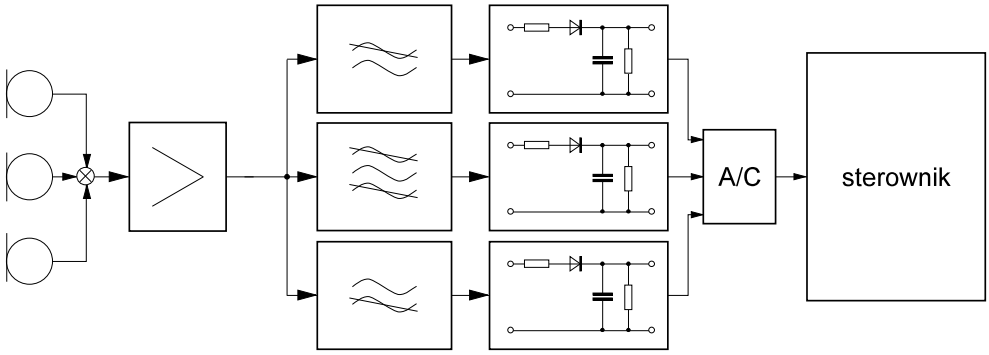
5.2 Elektroniczne układy pomiarowe

Sygnały z czujników zbliżeniowych są binarne, sterownik odczytuje ich stan bezpośrenio na porcie równoległym.
Sygnał z akcelerometru ADXL202 ma postać przebiegu prostokątnego o wypełnieniu zależnym od zmierzonej wartości przyspieszenia. Określenie aktualnego przyspieszenia jest możliwe przez pomiar wypełnienia sygnału za pośrednictwem układu czasowo–licznikowego umieszczonego w sterowniku.
Sygnały z żyroskopu, czujnika ciśnienia i mikrofonu mają postać analogową i wymagają zastosowania dodatkowych układów elektronicznych przed podłączeniem do sterownika.
Żyroskop wymaga zastosowania filtru górnoprzepustowego w celu eliminacji składowej stałej, oraz dodatkowego filtru dolnoprzepustowego eliminującego szumy (rys. 13).



Rysunek 13: Schemat blokowy toru pomiarowego sygnału z żyroskopu

W sygnale dźwiękowym rejestrowanym przez mikrofon interesującymi parametrami są natężenie i częstotliwość dźwięku. W celu uproszczenia ekstrakcji tych parametrów z sygnału zastosowano dodatkowe układy przetwarzające (rys. 14), które umożliwiają pomiar natężenia dźwięku o określonym zakresie częstotliwości jako sygnału napięciowego. Całkowite natężenie dźwięku w takim przypadku



Rysunek 14: Schemat blokowy toru pomiarowego sygnałów dźwiękowych

może być określone jako suma natężeń z wszystkich zakresów częstotliwości.
Czujnik ciśnienia MPX4200 przetwarza ciśnienie na sygnał napięciowy o wartości proporcjonalnej do wartości zmierzonego ciśnienia. Przed podłączeniem do sterownika sygnał ten musi zostać wzmocniony. Filtr dolnoprzepustowy eliminuje szumy pomiarowe.
Przetworzone sygnały z żyroskopu, czujnika ciśnienia i mikrofonu są podłączone do wejść analogowych mikrokontrolera.

5.3 Układy wyjściowe

Diody świecące umieszczone na całej powierzchni obudowy robota (w kolorach: czerwonym, zielonym i niebieskim) są zasilane sygnałem prostokątnym. Jasność świecenia diod LED jest zmieniana przez modulację wypełnienia (PWM), do czego wykorzystywany jest *timer*. Wzajemny stosunek jasności trzech barw składowych daje sterowanie odcieniem barwy. Takim samym sygnałem jest zasilany silnik wibratora. Amplitudę drgań można regulować przez odpowiednie sterowanie prądem silnika. Zarówno diody, jak i silnik są podłączone do sterownika za pomocą typowych układów ULN2801.
Sterowanie rejestratorem dźwięku odbywa się przez symulawanie przycisków sterujących przy pomocy układów kluczujących. Możliwe jest również odtwarzanie prostych dźwieków (tonów) generowanych w postaci sygnałów PWM przez *timer* mikrokontrolera.
Z punktu widzenia sterownika wszystkie sygnały sterujące układami wyjściowymi mają postać dyskretną.

5.4 Sterownik

Wbudowany do robota kulistego sterownik musi być lekki, mieć niewielkie rozmiary, charakteryzować się odpornością na wstrząsy i wibracje. Moc obliczeniowa zastosowanego procesora nie jest sprawą najwyższej wagi, ponieważ zjawiska zachodzące w otoczeniu są wolnozmienne. Duże znaczenie ma natomiast łatwość przyłączania dużej ilości układów wejściowo–wyjściowych (analogowych i cyfrowych) oraz wygoda programowania (w szczególności dotyczy to implementacji *Fuzzy Engine*). W celu zapewnienia komunikacji z terapeutą niezbędny jest port szeregowy. Do zapamiętywania programowanych przez terapeutę reguł zachowania niezbędna jest pamięć nieulotna (najlepiej typu EEPROM).
Wykorzystując doświadczenie z wcześniejszych projektów rozważono kilka wariantów mikrokontrolerów firmy Motorola. Są wśród nich jednostki 16-bitowe (68HC912, MC9S12) i 32 bitowe (68332, 68F396). Podstawowe parametry, istotne dla opisywanego sterownika robota kulistego zebrano w tabeli 5.

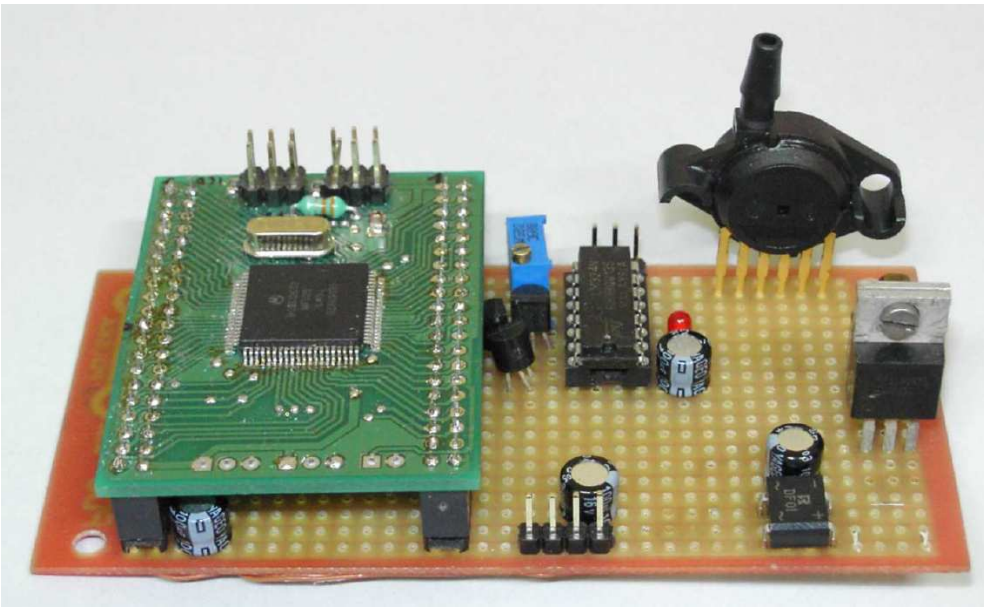
Parametr	68HC12B32	MC9S12A256	68332	68F396
częstotliwość zegara	8MHz	25MHz	25MHz	25MHz
typ jednostki centralnej	CPU12	CPU12	CPU32	CPU32+
rozmiar słowa	16	16	32	32
wbudowana pamięć FLASH	32kB	256kB	0	128kB
wbudowana pamięć EEPROM	0.75kB	4kB	0	0
wbudowana pamięć RAM	1kB	12kB	2kB	8kB
timer-y	12	16	16	24
porty szeregowe (SCI)	1	2	1	1
interfejsy urządzeń	SPI	IIC,2xSPI	QSPI	QSPI
przetworniki A/C	8x10b	16x10b	0	16x10b
wbudowany emulator	BDM	BDM	BDM	BDM
wbudowane instrukcje *FUZZY*	+	+	-	-

Tablica 5: Porównanie własności testowanych mikrokontrolerów.

Analiza właściwości przedstawionych wyżej procesorów wskazuje, że najodpowiedniejszym jest MC9S12A256. Na uwagę zasługuje dobre wyposażenie w układy we/wy, interfejs BDM, wystarczające zasoby pamięciowe, a zwłaszcza zestaw instrukcji w języku wewnętrznym pozwalających zoptymalizować implementację maszyny wnioskującej [7]. W próbnej konstrukcji sterownika użyto mikrokontrolera MC9S12A64 (można zastosować również wersje A32, A128, A256 różniące się tylko ilością pamięci FLASH). Widok układu próbnego z zamontowaną jednostką centralną [22] i czujnikiem ciśnienia przedstawiono na rys. 15.
Układ zmontowano na płytce uniwersalnej, zewnętrzne czujniki i elementy wykonawcze są przyłączane przewodami taśmowymi. Ostateczna wersja sterownika wymaga opracowania specjalizowanego obwodu drukowanego zapewniającego male rozmiary układu i dużą odporność mechaniczną.

6 Podsumowanie

W wyniku podjęcia współpracy z ośrodkami zajmującymi się terapią autyzmu u dzieci młodszych ustalono założenia dla interaktywnego robota przeznaczonego do jej wspomagania. Uzgodniono kulisty kształt robota i wybrano zestaw bodźców, które robot powinien odbierać z otoczenia (siła, dźwięk, dotyk, ruch) i reakcji robota na te bodźce (świecenie, dźwięk, wibracje).



Rysunek 15: Widok płytki sterownika

Zaproponowane układy sensoryczne w połączeniu z odpowiednimi procedurami umieszczonymi w sterowniku robota pozwalają na określenie wielu aspektów aktualnego stanu robota i jego otoczenia. Aby umożliwić łatwe dostosowywanie zachowania robota do sytuacji terapeutycznej (zwłaszcza do różnych dzieci, o różnym stopniu nasilenia zachowań autystycznych) opracowano sterownik, którego ważną zaletą jest łatwość programowania zachowań robota przez osoby nie zajmujące się robotyką. Wykorzystanie sterownika opartego na wnioskowaniu rozmytym w oparciu o bazę wiedzy utworzoną przez eksperta (terapeutę) pozwoliło zbudować odpowiedni interfejs użytkownika.
W trakcie planowanych badań pilotażowych na grupie dzieci z wykorzystaniem modelu robota kulistego i po ich przeprowadzeniu, niezbędne będzie zapewne dokonanie poprawek i modyfikacji koncepcji, rozbudowanie sensorów i efektorów, dodanie bodźców i reakcji.

Literatura

[1] J. A. Adams *Critical Considerations for Human-Robot Interface Development*. 2002 AAAI Fall Symposium: Human Robot Interaction Technical Report FS-02-03, pp.1-8, 2002.

[2] ADXL202E. Low-Cost $\pm 2g$ Dual-Axis Accelerometer with Duty Cycle Output. Analog Devices Inc., 2000.

[3] K. Arent *Interfejs terapeuty kulistego robota społecznego wspomagającego terapię dzieci autystycznych*. W: Postępy robotyki: Sterowanie robotów z percepcją otoczenia, Wydawnictwa Komunikacji i Łączności, Warszawa 2005.

[4] AURORA. `http://www.aurora-project.com`., 1998.

[5] J. J. Buckley, E. Eslami. *An introduction to fuzzy logic and fuzzy sets*. Physica-Verlag, Heidelberg, 2002.

[6] K. Cichocki *Interfejs terapeuty dla robota kulistego społecznego do terapii u dzieci ze spektrum autyzmu*. Praca magisterska, Politechnika Wrocławska, 2005.

[7] CPU 12 Reference Manual. *CPU12RM/AD*, Rev. 1, Motorola Inc., 1996,1997.

[8] K. Dautenhahn. Roles and functions of robots in human society: implications from research in autism therapy. *Robotica*, vol. 21, 2003.

[9] ENC-03J. Piezoelectric Gyroscopes GYROSTAR. `http://www.murata-europe.com`., 2004.

[10] T. Fong, I. Nourbakhsh and K. Dautenhahn. A Survey of Socially Interactive Robots: Concepts, Design, and Applications. *Robotic and Autonomous Systems*, vol. 42, 2003.

[11] Fuzzy Logic Toolbox User's Guide. The MathWorks, 2002.

[12] INTERNATIONAL ELECTROTECHNICAL COMMISION (IEC), IEC 1131 - PROGRAMMABLE CONTROLLERS Part 7 - Fuzzy Control Programming Commitee Draft CD 1.0 (Rel. 19 Jan 97)

[13] M. Kabała. *Układy sensoryczne i wykonawcze w robocie kulistym wspierającym terapię dzieci autystycznych.* W: Postępy robotyki: Sterowanie robotów z percepcją otoczenia, Wydawnictwa Komunikacji i Łączności, Warszawa 2005.

[14] KBG.EXE. A Fuzzy Logic Knowledge Base Generator for the MC68HC11 and MC68HC05 Inference Engines. `http://www.motorola.com`, 1995.

[15] J. Kruk-Lasocka *Autyzm czy nie autyzm. Problemy diagnostyki i terapii pedagogicznej małych dzieci.*. Dolnośląska Wyższa Szkoła Edukacji. Wrocław, 1999.

[16] F. Micheaud, S. Carron. Roball - An autonomous toy-rolling robot. *Proceedings of the Workshop Interactive Robot Entertainment*, 2000.

[17] G. M. More. *Audio Reproduction on HCS12 Microcontrollers*. AN2250/D, Motorola, 2002.

[18] MPX4200A Integrated Silicon Pressure Sensor, `http://e-www.motorola.com`, karta katalogowa.

[19] Piezoelectric Vibrating Gyroscopes (GYROSTAR). `http://www.murata.com`, karta katalogowa

[20] M. Szymkat (red.) *Komputerowe wspomaganie o obliczeniach naukowo-technicznych – przykłady zastosowań pakietów MATLAB i Maple V.*. Katedra Automatyki AGH,Kraków, 1998.

[21] Using MATLAB Graphics. The MathWorks, 2005.

[22] M. Wnuk. *Moduł z mikrokontrolerem MC9S12C32*. Raport SPR 19/2004, Inst. Cyb. Techn. PWr, 2004.

[23] M. Wnuk. *Programowalny sterownik interaktywnego robota kulistego wspomagającego terapię dzieci autystycznych.* W: Postępy robotyki: Sterowanie robotów z percepcją otoczenia, Wydawnictwa Komunikacji i Łączności, Warszawa 2005.

dr inż. Krzysztof Arent
mgr inż. Marek Kabała
dr inż. Marek Wnuk
Instytut Informatyki, Automatyki i Robotyki
Politechniki Wrocławskiej
ul. Janiszewskiego 11/17
50-372 Wrocław

Niniejszy raport otrzymują:

1. OINT ..... 1 egz.

2. WCSS ..... 1 egz.

3. Zleceniodawca ..... 2 egz.

4. Autorzy ..... 3 egz.

Razem : 7 egz.

Raport wpłynął do redakcji I-6
w czerwcu 2005 roku.