1 Zakres ustalonych prac

- Instalacja na dwóch komputerach:
  - serwera Ubuntu v16.04 w maszynie wirtualnej Hyper-V, w środowisku Win10
  - instalacja na Ubuntu środowiska ev3dev (brickstrap — jako guestfs) oraz narzędzi (kompilatory C i C++ itd.)
  - instalacja narzędzi do aktualizacji/modyfikacji i tworzenia obrazu systemu wgrywanego na SDHC do EV3
  - instalacja narzędzi do kompilacji skrośnej z użyciem aplikacji Docker
- Sprawdzić środowisko kompilacji skrośnej na przykładach:
  - `https://github.com/theZiz/ev3c`
  - `https://github.com/ddemidov/ev3dev-lang-cpp`
- Dostarczenie biblioteki do C++ dostarczającej abstrakcji służących do implementacji maszyn stanów dla środowiska ev3dev.
- Złożenie dwóch zestawów LEGO i uruchomienie środowiska ev3dev na sterownikach zestawów.

2 Opis wykonanych prac

2.1 Instalacja systemów Ubuntu

Instalacja systemów Ubuntu w maszynie wirtualnej Hyper-V nie przysporzyła żadnych problemów, należy postępować zgodnie z instrukcjami wyświetlanymi na ekranie.
W laboratorium zainstalowaliśmy dwa systemy Ubuntu w wersji 14.04, a następnie jedną maszynę z systemem Ubuntu 16.04.

2.2 Instalacja na Ubuntu środowiska ev3dev, oraz narzędzi

Gdy zaczynaliśmy prace nad projektem (sierpień 2016) jedyną możliwością natywnej kompilacji skrośnej na sterownik LEGO MINDSTORMS było użycie narzędzia `brickstrap`. Jest to narzędzie służące do instalacji systemu Debian na sterowniku LEGO MINDSTORM, a także udostępniające możliwość kompilacji skrośnej bez konieczności emulacji architektury sterownika. Dzięki narzędziu kompilacja jest znacząco szybsza w porównaniu do metod emulujących architekturę urządzenia i kompilujących projekty w wirutalnym środowisku.
Instalacja środowiska `brickstrap` do celów kompilacji okazała się niełatwym zadaniem. Wynika to z faktu rozbieżności instrukcji instalacji i użycia narzędzia dostarczonej na stronie `http://www.ev3dev.org/docs/tutorials/using-brickstrap-to-cross-compile/` ze stanem faktycznym narzędzia, co w połączeniu z nieudokumentowanymi zmianami zarówno w interfejsie, jak i nazewnictwie pewnych składowych narzędzia sprawiło, że na ten problem poświęciliśmy sporą część czasu.
Na początku września autorzy ev3dev wprowadzili nową metodę natywnej kompilacji skrośnej i zakończyli wsparcie dla kompilacji przy pomocy `brickstrap`. Ponieważ w ramach projektu, przed ogłoszeniem nowej metody, powstała poprawna, sprawdzona i działająca instrukcja 3.1 instalacji i konfiguracji narzędzia `brickstrap` umieszczamy ją w raporcie na wypadek, gdyby czytelnik zechciał użyć tego narzędzia, pomimo braku oficjalnego wsparcia.

2.3 Instalacja środowiska kompilacji skrośnej opartego na aplikacji docker

Po wprowadzeniu przez autorów ev3dev możliwości kompilacji skrośnej w oparciu o narzędzie `docker` postanowiliśmy stworzyć instrukcję instalacji tego narzędzia, a także dostarczyć skrypt, który w prosty sposób pozwalałby użytkownikom na kompilację całego projektu. Ta decyzja została podjęta po wstępnych próbach użycia narzędzia, podczas których okazało się być ono o wiele łatwiejsze w obsłudze, zarówno pod względem instalacji, jak i utrzymania. Nieocenionym wydaje się być prostota zamrożenia wersji ev3dev, która często bywa zmieniana przez autorów bez wsparcia dla starszych wersji. Instrukcja instalacji środowiska i jego użycia została opisana w Sekcji 3.1. Warto podkreślić, że kompilacja skrośna przy użyciu narzędzia `docker` zadziała na każdym systemie wspierającym to narzędzie.

2.4 Sprawdzenie środowiska kompilacji skrośnej na przykładach

Obydwa przykłady zostały sprawdzone zarówno przy użyciu narzędzia `brickstrap`, jak i narzędzia `docker`. Przy użyciu narzędzia `docker` w przykładzie projektu z C należało zdefiniować zmienną `CC` w pliku `Makefile` zgodnie z instrukcją, a przykład C++ skompilować za pomocą skryptu `docker_build.sh`. Przykłady kompilują się przy użyciu obydwu narzędzi.

2.5 Złożenie zestawów LEGO i uruchomienie ev3dev na sterownikach zestawów

W ramach projektu złożyliśmy dwa zestawy LEGO i podłączyliśmy sensory do sterownika głównego. Uruchomienie środowiska ev3dev na sterownikach okazało się proste, aczkolwiek instrukcja umieszczona na stronie ev3dev (`http://www.ev3dev.org/docs/getting-started/`) wskazuje na użycie aplikacji `Etcher` w celu stworzenia karty SD z obrazem środowiska. Pomimo prób użycia tej aplikacji zarówno na systemie Windows, jak i Linux nie udało nam się stworzenie takiej karty przy pomocy aplikacji — dostawaliśmy nieczytelne błędy. Po kilku nieudanych próbach z powodzeniem użyliśmy komendy `dd` dostępnej w systemie Linux w następujący sposób:

```
sudo dd bs=4M if=./ev3-ev3dev.img of=/dev/sdc status=progress && sync
```

gdzie `ev3-ev3dev.img` to nazwa obrazu pobranego ze strony ev3dev, a `/dev/sdc` to ścieżka do karty SD.
Pozostała część instrukcji instalacji środowiska ev3dev na sterowniku LEGO MINDSTORMS okazała się być poprawna.

2.6 Implementacja biblioteki do tworzenia maszyn stanów

W celu ułatwienia użytkownikowi pracy z biblioteką ev3dev zaimplementowaliśmy w języku C++ bibliotekę pozwalającą na proste opisywanie maszyn stanów. Poniżej opisujemy podstawowe własności powstałej biblioteki.

Równoległość

Przedstawione rozwiązanie używa dwóch wątków, aby symultanicznie czytać stan sensorów i motorów podłączonych do urządzenia, oraz realizować maszynę stanów wraz z odpowiednimi komendami do motorów.

Zapewnienie synchronizacji dostępu do sensorów i motorów

Biblioteka ev3dev nie synchronizuje w żaden sposób dostępu do sensorów i motorów urządzenia. Z tego powodu, aby zapewnić bezpieczne czytanie i pisanie danych do urządzeń, dostarczamy specjalnych 'opakowań' na każde z fizycznych urządzeń, które działa identycznie jak obiekt, który opakowuje, jednocześnie gwarantując poprawną synchronizację między wątkami.

Interfejs logowania zdarzeń

Ponieważ znajdowanie błędów w programie opartym o fizyczne urządzenia może być czasochłonne i trudne, biblioteka udostępnia interfejs logowania danych. Użytkownik może w prosty sposób oznaczać poziom ważności komunikatów, dostaje też informację o dokładnym czasie komunikatu, oraz ID wątku który go wygenerował. Przykładowy sposób użycia interfejsu został pokazany poniżej.

```
1 #include "Logger.hpp"
2
3 int main() {
4     INFO << "Info message!";
5     WARNING << "Warning message!";
6 }
```

Wat

3 Instrukcja instalacji

3.1 Instalacja środowiska kompilacji

Instalacja środowiska brickstrap

Wszystkie skrypty użyte w tej instrukcji są również dostępne w katalogu `./scripts` w repozytorium. Metoda opisana poniżej została przetestowana na systemie operacyjnym Ubuntu 16.04.

1. Uruchom skrypt instalacji `brickstrap` (`scripts/install_bricktrap.sh`):

```
sudo apt-key adv --keyserver pgp.mit.edu --recv-keys 2B210565
sudo apt-add-repository "deb http://archive.ev3dev.org/ubuntu trusty main"
sudo apt-get update
sudo apt-get install -y brickstrap

# create a supermin appliance
sudo update-guestfs-appliance
# add yourself to the kvm group
# need to log out and back in for this to take effect
sudo usermod -a -G kvm $USER
newgrp kvm
# fix permissions on /boot/vmlinuz*
sudo chmod +r /boot/vmlinuz*
# And you need to add yourself to /etc/subuid and /etc/subgid to be able
# to use uid/gid mapping.
```

```
sudo usermod --add-subuids 200000-265534 --add-subgids 200000-265534 $USER

# create virtual environment
mkdir work
cd work
 -p argument from ls /usr/share/brickstrap/projects
 -c argument form ls /usr/share/brickstrap/projects/ev3dev-jessie
brickstrap -p ev3dev-jessie -c ev3 -d ev3-rootfs create-rootfs
brickstrap -d ev3-rootfs shell << EOF
apt-get update
apt-get install -y build-essential
EOF
```

2. Zainstaluj środowisko kompilacja skrośnej z `cmake` w wersji 3.x:

   - Zainstaluj `CMake` w wersji 3.5:

     ```
     sudo apt-get install cmake
     ```

   - Zainstaluj zależności w powłoce `brickstrap`:

     ```
     cat > /etc/apt/sources.list <<EOL
     deb http://cdn.debian.net/debian jessie main contrib non-free
     deb-src http://cdn.debian.net/debian jessie main contrib non-free

     deb http://archive.ev3dev.org/debian jessie main
     deb-src http://archive.ev3dev.org/debian jessie main
     EOL
     apt-get update
     apt-get build-dep brickman
     apt-get install symlinks
     symlinks -c /usr/lib/arm-linux-gnueabi
     ```

   - Wyjdź z powłoki `brickstrap` i przygotuj środowisko CMake'a (`scripts/prepare_cmake_env.sh`):

     ```
     sudo apt-get install gcc-arm-linux-gnueabi g++-arm-linux-gnueabi \
         cmake valac pkg-config
     cd ~/work
     cat > arm-linux-gnueabi.cmake <<EOL
     set(CMAKE_SYSROOT $ENV{HOME}/work/ev3-rootfs/rootfs)

     set(CMAKE_SYSTEM_NAME Linux)

     set(CMAKE_C_COMPILER arm-linux-gnueabi-gcc)
     #set(CMAKE_CXX_COMPILER arm-linux-gnueabi-g++)

     set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
     set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
     set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
     ```

```
set(CMAKE_FIND_ROOT_PATH_MODE_PACKAGE ONLY)
EOL
cat > ev3-rootfs-cross.env <<EOL
SYSROOT_PATH=${HOME}/work/ev3-rootfs/rootfs

export PKG_CONFIG_ALLOW_SYSTEM_CFLAGS=1
export PKG_CONFIG_ALLOW_SYSTEM_LIBS=1
export PKG_CONFIG_SYSROOT_DIR=${SYSROOT_PATH}
export PKG_CONFIG_LIBDIR=${SYSROOT_PATH}/usr/lib/arm-linux-gnueabi/pkgconfig
export PKG_CONFIG_LIBDIR=${PKG_CONFIG_LIBDIR}:${SYSROOT_PATH}/usr/lib/pkgconfig
export PKG_CONFIG_LIBDIR=${PKG_CONFIG_LIBDIR}:${SYSROOT_PATH}/usr/share/pkgconfig
export PKG_CONFIG_LIBDIR=${PKG_CONFIG_LIBDIR}:${SYSROOT_PATH}/usr/local/lib/arm-linu
export PKG_CONFIG_LIBDIR=${PKG_CONFIG_LIBDIR}:${SYSROOT_PATH}/usr/local/lib/pkgconfi
export PKG_CONFIG_LIBDIR=${PKG_CONFIG_LIBDIR}:${SYSROOT_PATH}/usr/local/share/pkgcon

export XDG_DATA_DIRS=${SYSROOT_PATH}/usr/local/share:${SYSROOT_PATH}/usr/share
EOL
```

- Użyj polecenia:

```
source ./ev3-rootfs-cross.env
```

3. Skonfiguruj środowisko kompilacji oparte o pliki `Makefile`.

- Stwórz plik `Makefile` (przykładowy plik w katalogu `examples/`):

```
PROGRAM = my-program
CROSS_COMPILE = arm-linux-gnueabi-
SYSROOT = $(HOME)/work/ev3-rootfs/rootfs

CC=$(CROSS_COMPILE)gcc
LD=$(CROSS_COMPILE)ld
CFLAGS= --sysroot=$(SYSROOT) -g -I$(SYSROOT)/usr/include

all: $(PROGRAM)

LIBDIR = -L=/usr/lib/arm-linux-gnueabi
#LIBDIR = -L$(SYSROOT)/usr/lib/arm-linux-gnueabi

LIBS = -lpthread

LDFLAGS= $(LIBDIR) $(LIBS)
SOURCE = my_program.c

OBJS = $(SOURCE:.c=.o)

$(PROGRAM): $(OBJS)
    $(CC) -o $@ $(OBJS) $(LDFLAGS)

clean:
```

```
rm -f $(OBJS) $(PROGRAM)
```

- Użyj pliku `Makefile` do skompilowania projektu

Instalacja środowiska docker

1. Zainstaluj najnowszą wersję aplikacji `docker` dla swojego systemu operacyjnego. Dla Ubuntu 14.04 skrypt do instalacji znajduje się repozytorium, w pliku `scripts/install_docker.sh`, natomiast dla innych systemów instrukcja instalacji znajduje się na stronie . Upewnij się, że użytkownik został poprawnie dodany do grupy `docker` w systemie operacyjnym.

2. Ściągnij obraz zawierający środowisko kompilacji i odpowiednio go oznacz:

```
docker pull fuine/ev3cc:v2
docker tag fuine/ev3cc:v2 ev3cc
```

3. Użyj skryptu `scripts/docker_build.sh` do budowania swoich projektów, opartych o narzędzie `CMake`. Przykładowy plik `CmakeLists/txt` znajduje się w katalogu `pidtest/`, a dodatkową pomoc dot. skryptu można uzyskać uruchamiając go z flagą `--help`.
   W przypadku projektów pisanych w języku C standardowy plik `Makefile` działa, pod warunkiem, że użytkownik użyje kompilatora o nazwie `arm-linux-gnueabi-gcc`.

3.2 Instalacja biblioteki

Dostarczona biblioteka oparta jest o generyczne pliki nagłówkowe, w związku z czym użytkownik powinien zapewnić, że biblioteka jest widoczna w ścieżce 'include' kompilatora, a następnie użyć wyspecjalizowanych typów z biblioteki. Przykładowy projekt pokazujący użycie biblioteki do budowy prostej maszyny stanów jest pokazany w przykładzie `TODO`.

4 Dokumentacja

Dostarczona biblioteka została napisana w oparciu o język angielski, w związku z czym dokumentacja również jest pisana w tym języku, w celu zapewnienia spójności tekstu z implementacją i komentarzami w źródłach biblioteki.

Short overview of the classes in the provided ev3dev lego framework. All classes have been implemented for the basic mechanical-arm lego set. Different implementations of the classes should base on the provided ones. Classes should be reimplemented to match the underlying lego set unless stated otherwise. For the example usage of the framework see *draft.cpp* file.

Safe

`Safe` class is a wrapper around given data and corresponding mutex. It is primarily used to provide thread-safe access to the underlying hardware. It provides overloaded dereference and arrow operators.

Users should not change the implementation of the `Safe` class

CraneControl

Controller for the hardware. It should contain all hardware handles used throughout application and corresponding mutexes. It should also implement methods providing thread-safe access to the underlying hardware by wrapping hardware handle and corresponding mutex in the `Safe` class instance.

See *CraneControl.cpp* for an example implementation.

CraneData

This class represents all data from external sensors, which is neccessary to trigger transitions in state machine. It should implement two methods: * `update` - updates all data fields * `==` operator - used to compare old read with new one and indicate possible change in the reads. User might include eventual tollerance for chosen parameters. Please note that `update` method will be running in a tight loop, therefore it is crucial to read neccessary data only.

`CraneData` interacts with underlying hardware via `CraneControl` instance provided as a reference in the constructor.

See *CraneData.cpp* for an example implementation.

Poller

Class responsible for polling the underlying hardware and providing thread-safe mechanism for acquiring the newest data.

Users should not change the implementation of the `Poller` class

State

Defines state in the state machine. This class stores **actions**, which will be executed once state will be reached, and **transitions**.

- **Actions** are represented as functors (std::function objects) which receive control object reference and data object const reference as arguments.
- **Transitions** to the next state are stored as tuples. Each tuple consists of a functor object (std::function) and a pointer to next state. Functors when called should return a boolean value indicating whether or not the transition should be made. Transition functions get the newest data and elapsed time (duration of time already spent in the current state) as their arguments. Each `State` can store multiple actions and transitions. They will be executed or, in case of transitions - checked, in the same order as they were added to the state.

Users should not change the implementation of the `State` class

EventLoop

Representation of the state machine. Can be run from the given starting state via `run` method. Machine can be stopped via `stop` method.

Users should not change the implementation of the `EventLoop` class

5 Możliwości rozszerzenia biblioteki