Aplikacja do śledzenia celów i fuzji danych

Dokumentacja końcowa

Przemysław Saramonowicz, Marcin Buczko, Jacek Palczewski
Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

26 stycznia 2015 r.

1. Realizacja zagadnień

Podsumowanie zostanie oparte o punkty z dokumentacji wstępnej, które zostaną skomentowane na podstawie bieżących etapów prac.

- **Symulacja ruchu obiektu** - obiekt będzie poruszał się w przestrzeni dwuwymiarowej po zdefiniowanych wcześniej trajektoriach ruchu oraz przekazywał dane o swoim położeniu sensorom.
  - Ten moduł programu udało się zrealizować w zadowalającym stopniu: trajektoria obiektu jest generowana w oparciu o skrypty w języku Python(katalog `maps/`).
- **Sensor ruchu** - odbiera dane o położeniu obiektu i przekazuje je razem z własnym szumem pomiarowym na wejście filtru Kalmana.
  - Szum sensora jest nakładany z rozkładem jednostajnym z parametrem wczytywanym z pliku wejściowego ini.
- **Filtr Kalmana** - Odbiera dane z sensorów i wyznacza stan wewnętrzny modelowanego obiektu.
  - Moduł został zrealizowany jako złożenie dwóch filtrów - każdy działający w jednej płaszczyźnie.
- **Moduł określający błąd pomiaru** - Odbiera dane od obiektu, sensorów oraz filtru Kalmana i określa jak dokładnie filtr Kalmana przewiduje tor w kolejnych iteracjach oraz określa poprawę w stosunku do odczytów sensorów.
  - Moduł został połączony z poniższym, por. punkt niżej i dalsze rozdziały.

- **Interpretacja graficzna działania programu** - Rysuje obiekt w kolejnych iteracjach na jego pozycji oraz jego kopię w miejscu określonym przez układ sensorów z filtrem Kalmana a także trajektorię obu instancji obiektu.
  - Mimo pierwotnych planów o zrealizowaniu tego modułu w języku C++, ze względu na problemy z implementacją wyświetlania wyników symulacji w czasie rzeczywistym Zespół podjął decyzję o zmianie podejścia: program składający się z pozostałych modułów jest zrealizowany jako aplikacja w C++, a bieżący moduł został napisany jako uniwersalny skrypt do środowiska Octave(kompatybilnego z Matlabem). Dokładne działanie całego projektu zostanie opisane później.
- Zależność pomiędzy torem ruchu i prędkością a dokładnością pomiaru; wpływ ilości sensorów na dokładność odczytu.
  - Możliwość uruchomienia z innym torem obiektu(jak również zmiany współczynników - pliki .ini) w opinii Zespołu realizuje olbrzymią część tego założenia.
  - Miejsce na komentarze...
- Przenośność pomiędzy Windowsem a Linuxem.
  - Był to podpunkt intrygujący ze względu na wiele niespodzianek(głównie ze względu na ciekawe różnice pomiędzy kompilatorami(Visual Studio jest o wiele bardziej liberalny i różni się subtelnościami w implementacji niektórych rzeczy nawet w bibliotece standardowej: dla przykładu konstruktor `std::exception` przyjmujący ciągi znaków pod MSVC, który nie jest dozwolony w GCC - tam identyczną funkcję pełni `std::runtime_error`)), aczkolwiek dzięki wykorzystaniu Jenkinsa udało się utrzymać ten podpunkt.

2. Opis stanu prac nad projektem

2.1. Rozwiązania zastosowane w projekcie - synchronizacja, etc.

2.1.1. Klasa Worker

Problem synchronizacji dotyczył praktycznie każdego modułu w aplikacji opartej na wątkach, dlatego został rozwiązany globalnie, poprzez klasę `Worker<T>`, która korzysta z dobrodziejstw programowania generycznego, czyli trejtów, uproszczając całą obsługę synchronizacji do przeciążenia funkcji `ThreadProc` w module dziedziczącym po w.w. klasie, a obsługa zmiennej warunkowej, kolejki i synchronizacji jest w gestii kompilatora dla wielu różnych, różniących się szczegółami klas.

2.2. Jak program działa - opis

1. Aplikacja

   Aplikację można uruchomić z konsoli, bądź poprzez skrypty `runme`, dzięki czemu automatycznie zostanie uruchomiony punkt 6 po zakończonej symulacji.

2. Pliki wejściowe

   Plikami wejściowymi są plik zawierający skrypt w języku Python umieszczony w katalogu `maps/` generujący oraz pliki ini z parametrami dla filtru Kalmana. Plik ini musi mieć taką samą nazwę jak skrypt generatora. w pliku ini zdefiniowane są następujące parametry: `TimeStep` - przedział czasu w jakim wykonujemy kolejne kroki obliczeń, **PosNoise** - szum sensora, **AccNoise** - szum procesu będący parametrem zakłócającym przyśpieszenie, [**InitPosition**] - początkowa pozycję w wektorze stanu filtru Kalmana, [**InitVelocity**] - początkowa prędkość w wektorze stanu filtru Kalmana.

3. Generator

   `Worker<OutputWorker>` uruchamia przeciążoną funkcję ThreadProc, która natomiast wywołuje skrypt który jest załadowany z pliku - za pomocą wyrażeń regularnych jest przeprowadzone wstępne sprawdzenie czy skrypt jest prawidłowy(tj. czy istnieje funkcja która ma taką samą nazwę co plik w celu weryfikacji) jest cyklicznie uruchamiana i wszystkie obliczenia są przeprowadzane w środowisku Python(dzięki czemu między wywołaniami funkcji w Pythonie możemy przechowywać dane w zmiennych globalnych - por. pliki w katalogu `maps/`, a biblioteka `Booost::python` pozwala na załadowanie tych wyników i dalsze przetwarzanie.

4. Sensor

   Sensor na wejściu dostaje współrzędne `(x,y)` punktu z generatora i dodając do nich swój szum przekazuje na wejście filtru Kalmana oraz do zarchiwizowania przez klasę Writer

5. Kalman

   Filtr Kalmana został oparty na bibliotekach "`The KFilter Project`"[1]. Filtr skonfigurowany jest na potrzeby procesu ruchu o zmiennym przyśpieszeniu w przestrzeni dwuwymiarowej. Bierze pod uwagę szumy procesu, oraz szumy pomiarowe. Parametry filtra są definiowane w plikach ini: (początkowe położenie,

[1] http://kalman.sourceforge.net/

początkowa prędkość oraz szum procesu i szum pomiarowy). Filtr operuje na wektorze stanu procesu zawierającym położenie oraz wektor prędkości. Dla każdej współrzędnej został stworzony oddzielny obiekt filtru działający tylko w jednym wymiarze. Jako wejście obiekt filtru dostaje także przyśpieszenie będące sterowaniem obiektu. Przyśpieszenie to jest przekazywane z generatora.

6. Writer

   Zespół podjął decyzję o przechowywaniu wartości w plikach CSV[2] ze względu na prostotę formatu.

7. Skrypty Matlab

   Z powstałego pliku .csv skrypt `script.m` wczytuje współrzędne punktów wygenerowanych w trakcie symulacji przez program, oblicza odchylenie standardowe różnic wartości współrzędnych x, y dla czujnika i filtra kalmana oraz rysuje 3 wykresy :

   (a) wartości zmiennych `X` dla generatora, czujnika i filtru Kalmana;
   (b) wartości zmiennych `Y` dla generatora, czujnika i filtru Kalmana;
   (c) torów ruchu na podstawie odczytu generatora, czujnika i filtru Kalmana.

   W celu uruchomienia programu z tą funkcją należy wywołać skrypt `runme.bat` (lub `runme.sh` dla systemów linuxowych) w argumentach podając mu ścieżkę do skryptu pythona dla interesującej nas trajektorii oraz wywołanie funkcji w języku Matlab np. `runme.bat ../maps/standard acc.py script(1000)`. Argumentem w przypadku skryptu `script.m` jest ilość punktów rysowanych na wykresie.

3. Napotkane problemy, popełnione błędy, wnioski

3.1. Problemy i błędy napotkane podczas pisania

3.1.1. Rozbieżności pomiędzy MSVC i GCC

Pisanie aplikacji dostępnym na wiele platform jest procesem dość wymagającym głównie ze względu na fakt, że MSVC jest, jak już zostało wspomniane

[2] ang. *Comma separated values* - mimo nazwy separatorem zostały średniki, wybrane w celu zachowania zgodności z aplikacją Excel, która była wstępnie wykorzystywania do analizy wyników.

wcześniej, kompilatorem dość liberalnym - pozwala na dużo rozwiązań które nie są możliwe jeżeli przychodzi co do kompilacji pod GCC. Dlatego cennym rozwiązaniem było skorzystanie Github Student Pack[3], dzięki któremu Zespół uzyskał dostęp do darmowej maszyny wirtualnej która pozwoliła na zainstalowanie instancji Jenkinsa działającej 24/7 - CI pozwalało śledzić czy każdy nowy commit(zwykle projektowany pod Windowsem) kompiluje się pod Linuxem - dlatego dość szybko można było w stanie wykryć poniższy błąd.

3.1.2. GCC i argument określany przez typedef

Tutaj Zespół spotkał się z dość dziwnym błędem - GCC nie pozwalało na określanie argumentu funkcji jako typ `void` w przeciwieństwie do MSVC który zezwala na taką operację, dlatego trejt `CommonUtil::Traits::InputWorker` definiuje typ argumentu funkcji `ThreadProc` jako int.

3.1.3. Rozbieżności pomiędzy szkieletem a aplikacją końcową.

Mimo początkowych planów skorzystania z bibliotek SDL i wxWidgets w finalnej wersji nie zostały użyte, głównie ze względu na to, że wyświetlanie komórek w czasie rzeczywistym jest procesem dość złożonym tak samo jak próba wpływu na zmiany na parametry symulacji.

3.1.4. Automatyzacja liczenia pokrycia kodu.

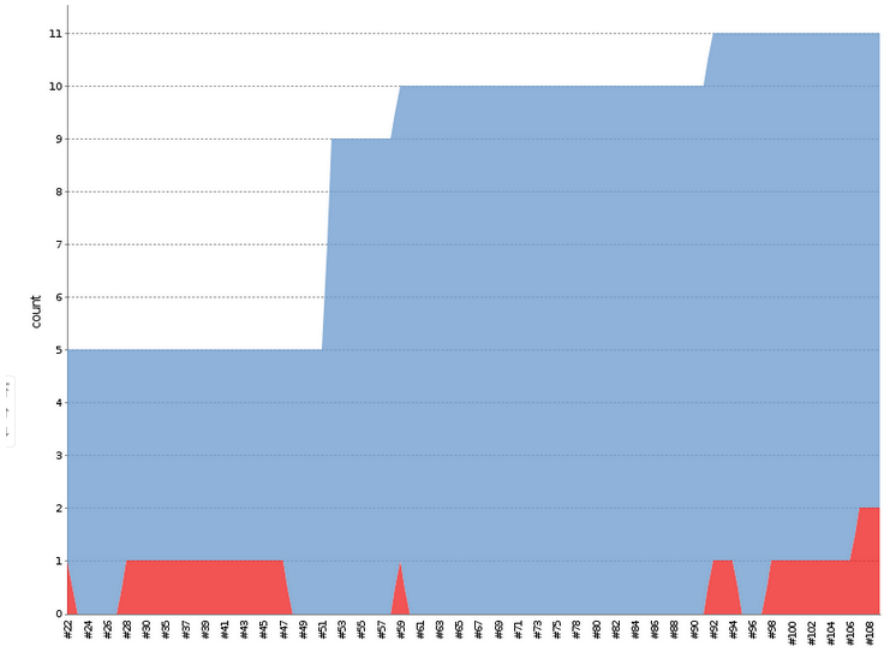
Mimo próby podejścia do automatyzacji(por. przełącznik coverage w pliku `SConscript`) liczenia pokrycia kodu nie udało się osiągnąć zadowalających rezultatów.

3.2. Wnioski, rozwiązania

3.2.1. Automatyzacja testowania

Z drugiej strony automatyzacja testów była dość dobra - każda udana kompilacja kończyła się uruchomieniem skryptu `test_runner.sh` na jenkisie, co kończyło się generowaniem plików .xml które były potem interpretowane przez wtyczkę do postaci wykresu:

[3] `https://education.github.com/pack`



(Gwoli ścisłości: testy zaznaczone na czerwono odnoszą się do skryptów wykonywanych przy wykorzystaniu ścieżek relatywnych, a plugin w jenkinsie odpowiedzialny za ich wykonanie *ucieka* ze względów bezpieczeństwa do folderu `/tmp`, przez co takie testy są skazane na niepowodzenie.)

4. Statystyki

Wynik uruchomienia programu `cloc` w głównym katalogu projektu:

Language	files	blank	comment	code
C/C++ Header	25	756	1723	1706
C++	23	168	6	611
Python	8	46	13	313
MATLAB	2	0	10	72
Bourne Shell	3	2	2	19
DOS Batch	1	5	0	15
XML	2	0	0	6
C	2	6	0	5
SUM:	66	983	1754	2747

Testów natomiast jest ok. 7.