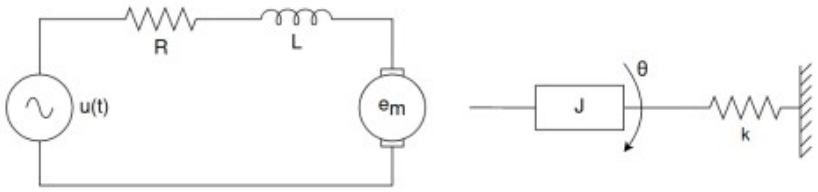
Sprawozdanie z projektu MMM

Hubert Rotkiewicz 193421 ACiR 1

Mateusz Stawski 193201 ACiR 1

Opis układu i treść zadania

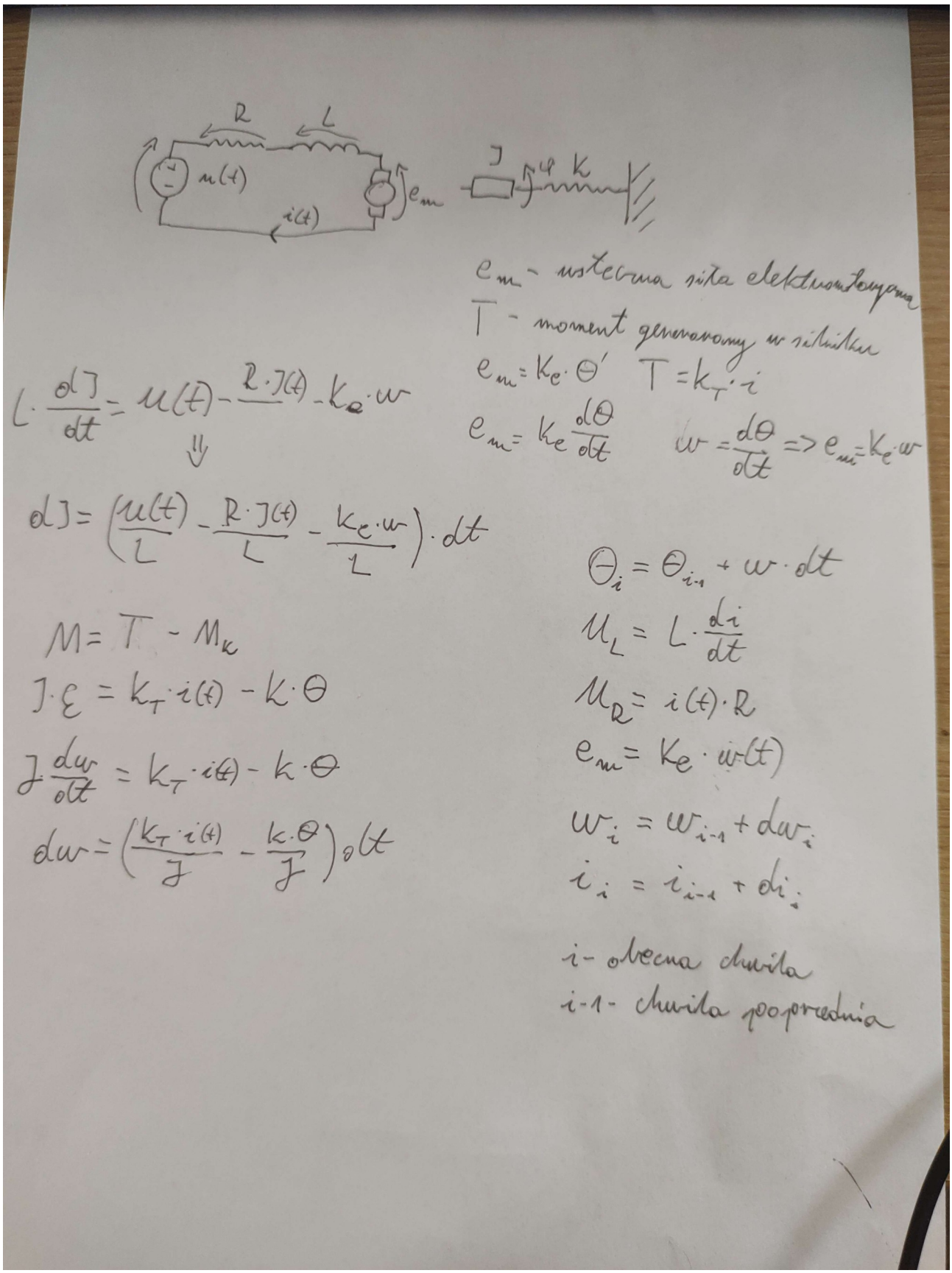
Projekt 3. Dany jest model sinika elektrycznego



gdzie moment obrotowy wału zależny jest od prądu w obwodzie elektrycznym jako $T = K_T i$, natomiast wsteczna siła elektromotoryczna generowana w obwodzie twornika dana jest jako $e_m = K_e \theta'$. Należy opracować model oraz zaimplementować symulację układu pozwalającą na wykreślenie prądu płynącego w obwodzie oraz prędkości kątowej wału silnika. Symulator powinien umożliwiać pobudzenie układu przynajmniej trzema rodzajami synagłów wejściowych (prostokątny o skończonym czasie trwania, trójkątny, harmoniczny). Symulator powinien umożliwiać zmianę wszystkich parametrów układu oraz sygnałów wejściowych.

Działanie programu

Program działa na zasadzie podania przez użytkownika odpowiednich parametrów. Następnie program oblicza kolejne stany na podstawie owych parametrów i stanu poprzedniego. Użytkownik może zadać programowi stała czasową która decyduje o szybkości oraz dokładności wykonywania symulacji. Nie powinno się dawać stałem czasowej większej niż 0.1, gdyż może spowodować to niepoprawny przebieg symulacji. Związane jest to z błędem który przekracza krytyczną wartość, co nie pozwala na poprawne obliczenie stanu następnego. W związku z tym, że program nie posiada żadnych ograniczeń co do wybierania wartości parametrów, użytkownik ma możliwość „zepsucia” programu.



Powyższe równania różniczkowe opisują działanie układu

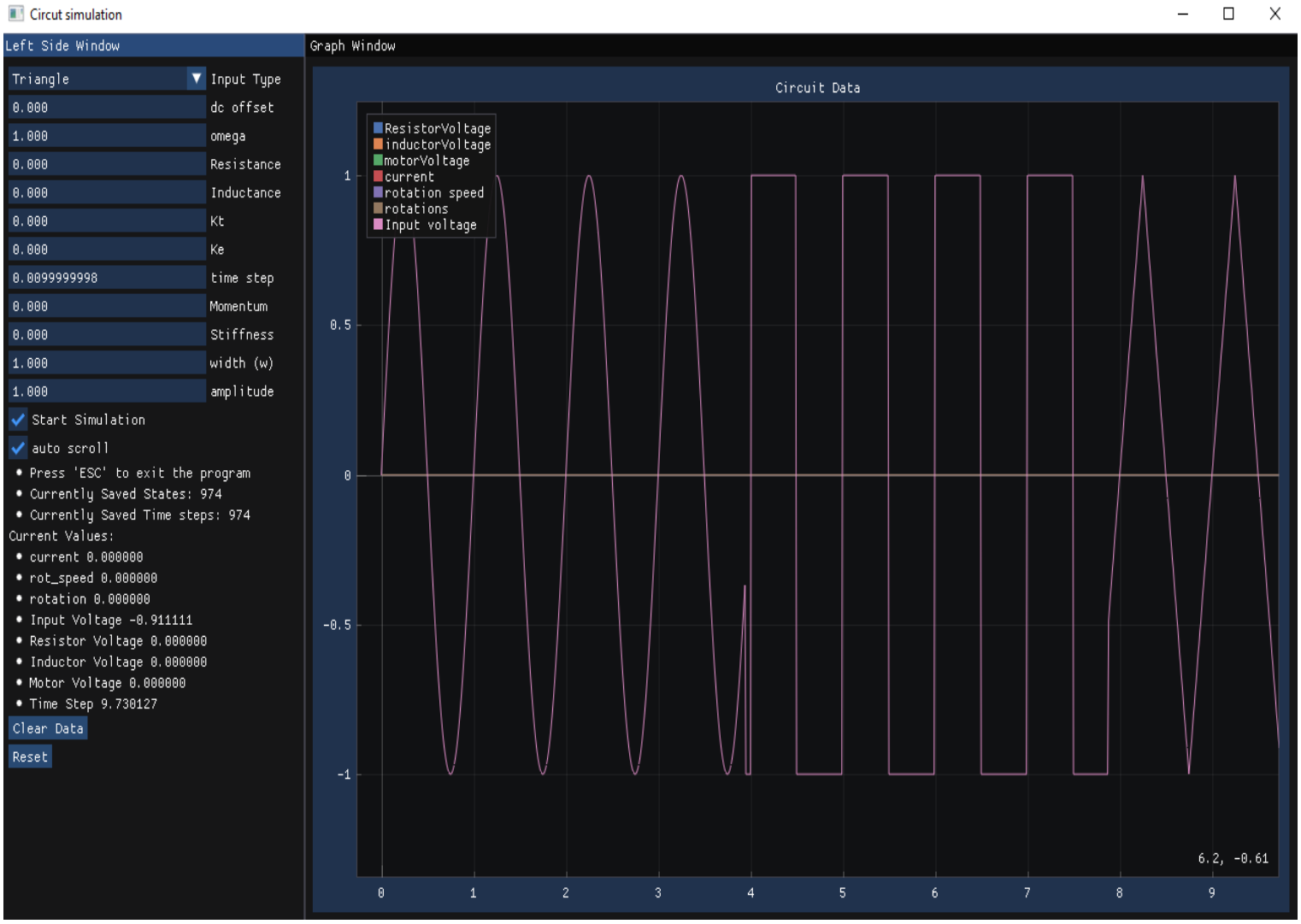
Główne funkcje programu

```
void update(float offset, float amplitude, InputShape shape, float w, float currentTime, float deltaT) {
  float input_Voltage = amplitude * std::visit([&](auto&& func) { return func(w, currentTime); }, input.at(shape)) + offset;
  state.InputVoltage  = input_Voltage;

  float di = (input_Voltage * deltaT) / params.L - (params.R * state.current * deltaT) / params.L - (params.Ke * state.rot_speed * deltaT / params.L);
  float dw = (params.Kt * state.current * deltaT) / params.I - (params.k * state.rotation * deltaT) / params.I;
  if (std::isnan(dw) || std::isnan(di)) return;
  state.rot_speed += dw;
  state.current += di;

  state.rotation        = state.rotation + state.rot_speed * deltaT;
  state.inductorVoltage = params.L * di / deltaT;
  state.motorVoltage    = params.Ke * state.rot_speed;      Hubert Rotkiewicz, 52 minutes ago • Fixed timestep and motor voltage
  state.ResistorVoltage = state.current * params.R;
```

Funkcja „Update” oblicza stan następny układu za pomocą równań różniczkowych



Interfejs programu

Opis interfejsu

Po lewej stronie użytkownik może:

- Zmieniać parametry programu
- Zmieniać przebieg napięcia wejściowego
- Uruchamiać/zatrzymywać przebieg symulacji
- Wybrać, czy kamera ma podążać za przebiegiem, czy też nie
- Zresetować cały przebiegi [Reset]
- Wyczyścić wszystkie dane zapisane do chwili wciśnięcia przycisku [Clear Data]

Program po lewej stronie również pokazuje obecny stan układu w liczbach

Po prawej stronie znajduje się wykres oraz legenda. Użytkownik ma możliwość włączania/wyłączania poszczególnych przebiegów z osi czasu klikając lewym przyciskiem myszy na odpowiednie przebiegi w legendzie. Samo najechanie myszką na przebieg w legendzie spowoduje lekkie podświetlenie przebiegu na wykresie.

Użytkownik może również modyfikować obie osie, klikając na nie prawym przyciskiem myszy.

Program korzysta z biblioteki ImGui i sdl3 jako backend.

```
void WindowManager::renderPlotWindow() {
  ImGui::SetNextWindowPos(ImVec2(300, 0));
  ImGui::SetNextWindowSize(ImVec2(ImGui::GetIO().DisplaySize.x - 300, ImGui::GetIO().DisplaySize.y));
  ImGuiWindowFlags window_flags = ImGuiWindowFlags_NoResize | ImGuiWindowFlags_NoMove | ImGuiWindowFlags_NoCollapse;
  ImGui::Begin("Graph Window", nullptr, window_flags);

  if (ImPlot::BeginPlot("Circuit Data", ImVec2(-1, ImGui::GetIO().DisplaySize.y - 35))) {  // -35 to account for the ImGui stuff
    if (timeSteps.size() > 0 && options.auto_scroll) ImPlot::SetupAxisLimits(ImAxis_X1, timeSteps.back() - 10, timeSteps.back(), ImGuiCond_Always);

    auto plotLine = [&](const char* label, auto valueGetter) {
      std::vector<float> values(states.size());
      std::transform(states.begin(), states.end(), values.begin(), valueGetter);
      ImPlot::PlotLine(label, timeSteps.data(), values.data(), timeSteps.size());
    };

    if (timeSteps.size() > 0) {
      plotLine("ResistorVoltage", [](const CircutState& s) { return s.ResistorVoltage; });
      plotLine("inductorVoltage", [](const CircutState& s) { return s.inductorVoltage; });
      plotLine("motorVoltage", [](const CircutState& s) { return s.motorVoltage; });
      plotLine("current", [](const CircutState& s) { return s.current; });
      plotLine("rotation speed", [](const CircutState& s) { return s.rot_speed; });
      plotLine("rotations", [](const CircutState& s) { return s.rotation; });
      plotLine("Input voltage", [](const CircutState& s) { return s.InputVoltage; });
    }
    ImPlot::EndPlot();
  }
  ImGui::End();
}
```

Funkcja wyświetlająca wykres

```cpp
void WindowManager::renderParametersWindow(std::function<void(void)> const& resetCallback) {
  ImGui::SetNextWindowPos(ImVec2(0, 0));
  ImGui::SetNextWindowSize(ImVec2(300, ImGui::GetIO().DisplaySize.y));

  ImGuiWindowFlags window_flags = ImGuiWindowFlags_NoResize | ImGuiWindowFlags_NoMove | ImGuiWindowFlags_NoCollapse;
  ImGui::Begin("Left Side Window", nullptr, window_flags);

  const char* inputTypeItems[] = {"Harmonic", "Square", "Triangle", "DC"};
  ImGui::Combo("Input type", &options.inputType, inputTypeItems, IM_ARRAYSIZE(inputTypeItems));
  ImGui::InputFloat("DC offset", &options.offset);
  ImGui::InputFloat("Amplitude", &options.amplitude);
  ImGui::InputFloat("Pulsation", &options.pulse);
  ImGui::InputFloat("Resistance", &params.R);
  ImGui::InputFloat("Inductance", &params.L);
  ImGui::InputFloat("Kt", &params.Kt);
  ImGui::InputFloat("Ke", &params.Ke);
  ImGui::InputFloat("time step", &options.time_step, 0.0, 0.0f, "%.10f");
  ImGui::InputFloat("Momentum", &params.I);
  ImGui::InputFloat("Stiffness", &params.k);
  ImGui::Checkbox("Start simulation", &options.start_simulation);
  ImGui::Checkbox("auto scroll", &options.auto_scroll);
  ImGui::BulletText("Press 'ESC' to exit the program");
  ImGui::BulletText("Currently saved states: %zu", states.size());
  ImGui::BulletText("Currently saved time steps: %zu", timeSteps.size());
  ImGui::Text("Current values: ");

  if (states.size() > 0) {
    ImGui::BulletText("current %f", states.back().current);
    ImGui::BulletText("rotational speed %f [rad/s]", states.back().rot_speed);
    ImGui::BulletText("rotation %f", states.back().rotation);
    ImGui::BulletText("Input voltage %f", states.back().InputVoltage);
    ImGui::BulletText("Resistor voltage %f", states.back().ResistorVoltage);
    ImGui::BulletText("Inductor voltage %f", states.back().inductorVoltage);
    ImGui::BulletText("Motor Voltage %f", states.back().motorVoltage);
    ImGui::BulletText("Time step %f", timeSteps.back());
  }

  if (ImGui::Button("Clear Data")) {
    states.clear();
    timeSteps.clear();
  }

  if (ImGui::Button("Reset")) {
    states.clear();
    timeSteps.clear();
    resetCallback();
  }
  ImGui::End();
}
```

funkcja wyświetlająca lewe okno z parametrami.

Wnioski

Stała czasowa powinna być nie większa niż 0,1 sekundy ze względu na bardzo wysoki błąd, który czyni symulację nieprecyzyjną, bądź też w ogóle nie poprawną. Przy stałej czasowej równej 1 sek, program w każdym przypadku daje błędne wyniki. Przy wyższych stałych czasowych program jest szybszy, ale daje mniej precyzyjne wyniki. Natomiast przy bardzo małych stałych czasowych program jest wolniejszy, ale za to daje bardziej precyzyjne wyniki.

Zwiększenie współczynnika Ke powoduje zwiększenie napięcia odkładanego na silniku, a zwiększenie współczynnika Kt powoduje zwiększenie momentu generowanego przez silnik, co skutkuje zwiększeniem maksymalnej pozycji kątowej silnika, oraz zwiększeniem prędkości. Zwiększenie sztywności sprężyny, czyli wsp. k powoduje zmniejszenie maksymalnej pozycji kątowej. Zwiększenie momentu wału powoduje zmniejszenie częstości zmiany pozycji kątowej wału, oraz zmniejsza prędkość kątową. Zwiększenie napięcia wejściowego zgodnie z oczekiwaniami zwiększa prędkość silnika, oraz maksymalną pozycję kątową.