Západočeská Univerzita v Plzni
Fakulta Aplikovaných Věd

Semestrální práce z předmětu KKY/PP

Obslužná aplikace pro model míče na cívce

Filip Jašek

Předmět: KKY/PP (Programové prostředky řízení)
Vyučující: Ing. Pavel Balda Ph.D.
Cvičící: Ing. Ondřej Severa, Ing. Jiří Faist

Datum: 4. února 2022

1 Zadání

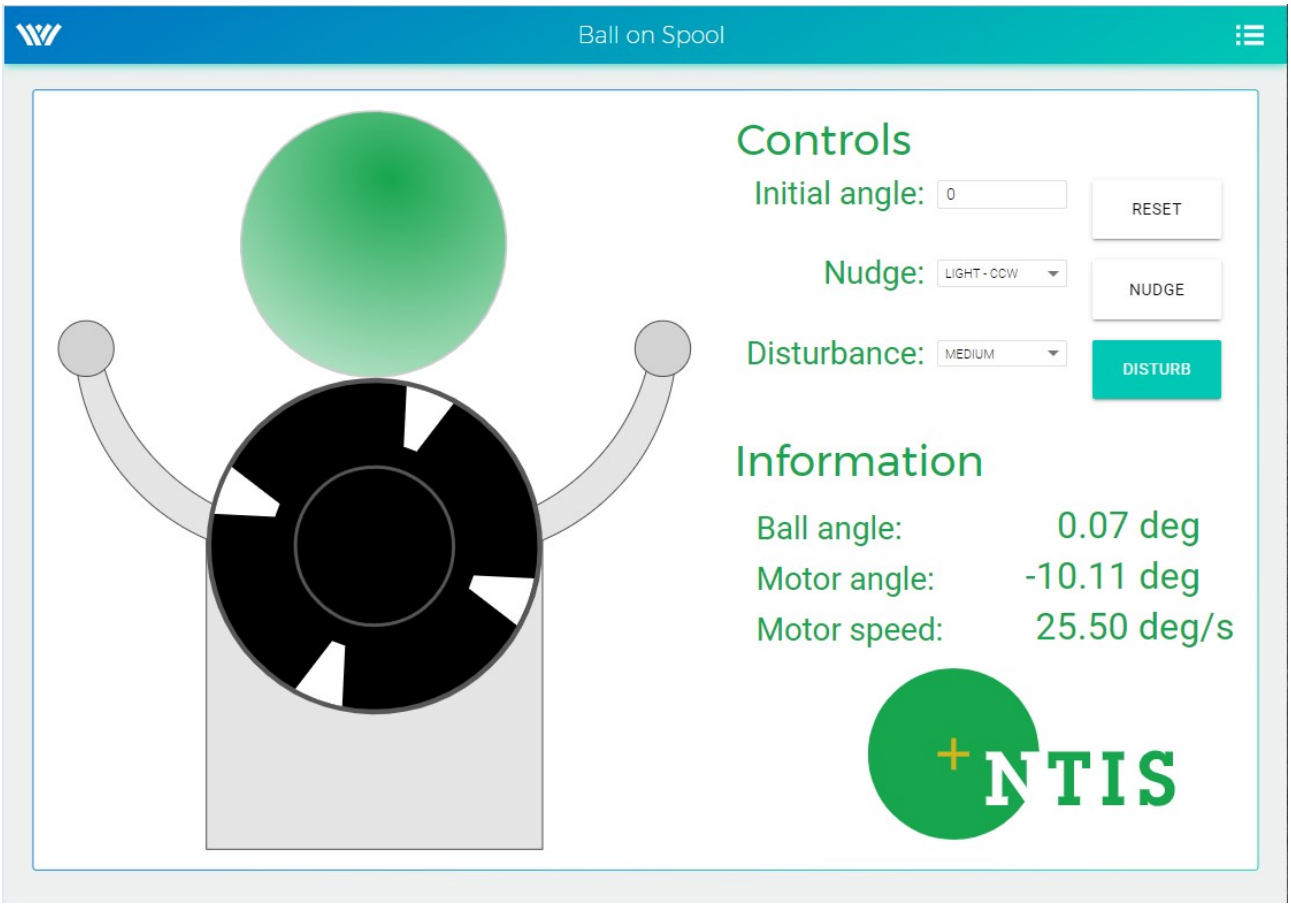
Úkolem bylo vytvořit uživatelské rozhraní HMI na základě poskytnutého matematického modelu. Model popisuje chování míče na pohyblivém válci.

Zadanou technologií pro zpracování bylo **Qt**, konkrétně třída **QPainter** pro grafické rozhraní a **REST** pro komunikaci s modelem spuštěném na serveru pomocí systému REXYGEN.

HMI mělo obsahovat grafiku, která zobrazí aktuální stav míče na válci tak, aby si nezávislý uživatel dokázal představit v jakém stavu se systém nachází viz. dodaný příklad (Obrázek 1).

Popis proměnných, které je potřeba vypisovat:
`Fi1_` — Aktuální natočení špulky [rad]
`dFi1_` — Aktuální rychlost špulky [rad/s]
`Fi2_` — Aktuální poloha středu míče vzhledem ke středu špulky [rad]
`dFi2_` — Aktuální rychlost ”padání”míče (pohyb středu míče vzhledem ke středu špulky) [rad/s]

Názvy parametrů bloků modelu, které je potřeba měnit z uživatelského rozhraní:
`mic.MP_NUDGE:BSTATE` — Skoková porucha v podobě strčení do míče
`mic.CNB_DISTRB:YCN` — Spuštění náhodně generované poruchy
`mic.MP_INTEG_RST:BSTATE` — Reset modelu
`mic.CNR_y0:ycn` — Nastavení počáteční podmínky na polohu míče na válci



Obrázek 1: Dodaný vzor uživatelského rozhraní HMI.

2 Použité technologie

2.1 Qt

Jedná se o multiplatformní framework pro vývoj GUI (Graphical User Interface) v jazyce **C++**. Výhodou je že lze vyvíjet software, který bude přenositelný mezi různými zařízeními a systémy (TV, infotainmenty, mobilní aplikace a další).

2.2 QPainter

Třída frameworku Qt, která se používá pro vykreslení jednoduchých grafik a animací v GUI.

2.3 REST

Zkratka REST znamená REpresentational State Transfer a jedná se o přístup k problematice čtení a zapisování dat z a na server. K datům na serveru přistupuje pomocí URI (Uniform Resource Identifier) což je jednotný identifikátor zdroje skládající se z řetězce definovaného určitou strukturou. Pro přístup k datům používá metody GET, POST, PUT a DELETE. První dvě řečené metody byly použity i v této semestrální práci, kdy díky GET se získala potřebná data ze serveru a pomocí POST byla nová data na server zapsána.

3 Implementace

Pro vytvoření aplikace byla jako základní kámen použita struktura z poskytnutého příkladu na použití QPainteru. V hlavní vykreslovací funkci bylo vytvořen obrázek zjednodušeného fyzického modelu ze zadání. Vykreslování objektů v aplikaci probíhá pomocí funkcí a souřadnic x a y. Počátek základní soustavy je v levém horním rohu vykresleného plátna. Směrem dolů je pak kladná osa y a vpravo kladná osa y. Zajímavé je, že pro vykreslování lze používat i translace a rotace, kdy hýbeme se středem nového počátku pro budoucí vykreslování viz. ukázka kódu níže s animací pohybu míče pomocí translace a dvou rotací pro věrohodnou animaci rotace míče i jeho pohybu vůči středu válce.

```
//vykresleni mice vcetne rotace jeho samotneho
painter.translate(QPoint(zakladnaX + sirkaZakladny * 0.5, zakladnaY));//posun do stredu valce
painter.rotate((180 / M_PI)* (valueFi2->text().toDouble()));//rotace mice okolo valce
painter.translate(QPoint(0, -sirkaZakladny));//posun do stredu valce
painter.rotate((180 / M_PI)* (valueFi1->text().toDouble()));//rotace mice samotneho
QRadialGradient gradient(0,-sirkaZakladny*0.4,sirkaZakladny*0.4,0,-sirkaZakladny*0.2,sirkaZakladny*0.5);
gradient.setColorAt(0, Qt::yellow);
gradient.setColorAt(1, Qt::green);
QBrush brush(gradient);
painter.setBrush(brush);
painter.drawEllipse(QPoint(0, 0), sirkaZakladny / 2, sirkaZakladny / 2);//vykresleni mice
```

Listing 1: Ukázkový kód použití QPainteru pro vykreslování míče a jeho pohybu.

Po vytvoření základní struktury pro zobrazování dat a obrázku modelu následovalo získání dat z modelu pro animaci modelu. Data jsou získávána každý tick pomocí funkce `requestData()` a v případě úspěšného napojení na server jsou data ve funkci `onUserDataFetched()` načtena do odpovídajících proměnných. V následujícím kódu lze vidět, jakým způsobem probíhá načtení pomocí REST a metody get. Je však potřeba dodat, že použitý `networkManager` byl již dříve definovaný jako `networkManager = new QNetworkAccessManager(this);`.

```
void RenderArea::requestData()
{
        QUrl url = QUrl("http://localhost:8008/api/data/mic/TRND?data&mime=application/json");
        QNetworkRequest request = getRequest(url);

        //vyslani requestu na ziskani dat ze serveru
        reply = networkManager->get(request);

        connect(reply, &QNetworkReply::finished, this, &RenderArea::onUserDataFetched);//nacteni dat
        connect(reply,static_cast<void
        (QNetworkReply::*)(QNetworkReply::NetworkError)>(&QNetworkReply::error),
        this,&RenderArea::onNetworkError);
}
```

Listing 2: Ukázkový kód použití REST a metody get.

Druhou zmíněnou metodou byl post, který slouží pro vložení dat na server. Ten je v kódu použit při stisku jednotlivých tlačítek. Jeho aplikace je v následující ukázce kódu.

```
void RenderArea::handleStouchBtn()
{
        //podminka pro zamezeni pokusu o zapis na server pokud neni pripojeni
        // a nebo probiha nejaky predchozi request, ktery jeste nedokoncil svou cinnost zapisu
        if (!connected || inprocess)
        {
                return;
        }
        inprocess = true;//informace, ze se zapisuje na server
        QNetworkRequest request = getRequest(buttonRefs[0]);
        int sp = 1;//hondota pro zapsani
        QString payload = "{\"v\":";
                payload += QString::number(sp);
                payload += "}";

        //vyslani post requestu na vlozeni dat na server
        postreply = networkManager->post(request, payload.toUtf8());

        //connect pro opetovne povoleni zapisu na server a vymazani pameti
        connect(postreply, &QNetworkReply::finished, this, &RenderArea::finishedPost);
}
```

Listing 3: Ukázkový kód použití REST a metody post na obslužné funkcí tlačítka pro skokovou poruchu (”šťouchnutí”).

Jak lze vidět na posledním kódu, bylo potřeba ošetřit zapisování na server, neboť při rychlém stisku tlačítek a horší odezvě mohlo docházet k přepisování requestů a následně pádu aplikace. Byly použity dvě proměnné, které zcela zamezí další tvorbě requestů. `Connected` signalizuje, zdali je HMI připojeno k serveru s modelem (`true`) nebo ne `false`. Podobně funguje `inprocess`, který však značí, zda se právě zpracovává request (`true`) nebo ne (`false`). Stejné ošetření je u všech tlačítek a kritické situace jsou takto eliminovány i při odpojení od serveru za běhu aplikace apod.

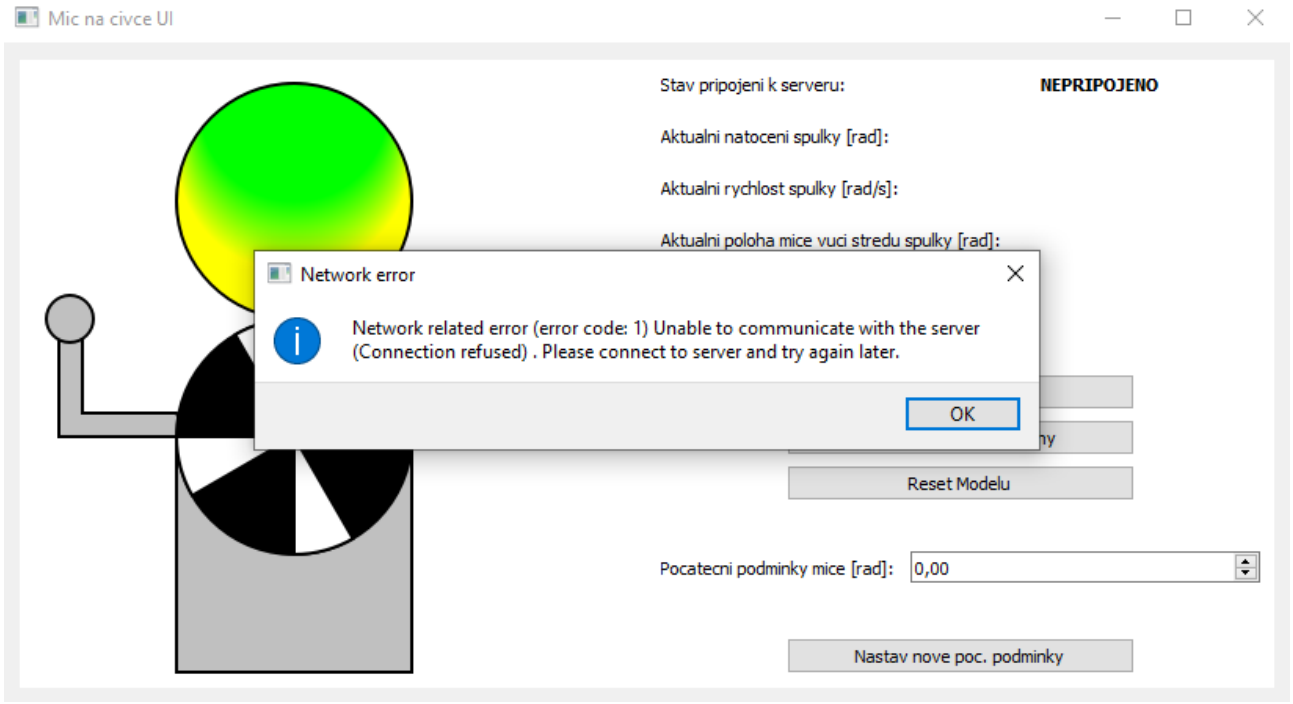
4 Uživatelský manuál

4.1 Upozornění

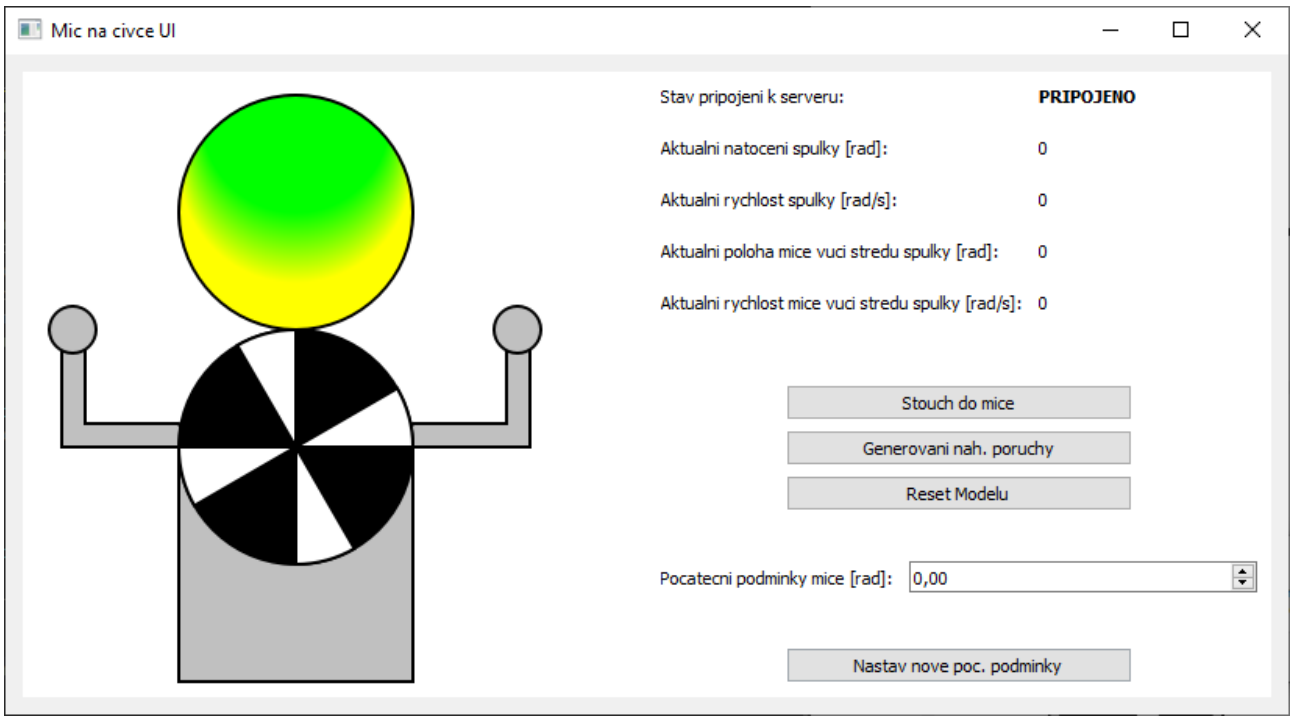
Prvně je potřeba říci, že aplikace byla specificky vytvořena pro zadaný model spuštěný na lokálním serveru a pro vzdálený přístup by bylo potřeba změnit URL adresy dat modelu a tlačítek. Bylo by potřeba zasáhnout do zdrojového kódu a znovu sestavit aplikaci. URL adresy tlačítek jsou však lehce přístupné na jediném místě zaznamenaném v poli `const QUrl buttonRefs` v hlavičce souboru `renderarea.cpp`. Ve stejném místě naleznete i proměnnou `const QUrl dataUrl`.

4.2 Popis prostředí

Po otevření aplikace se zobrazí přímo okno s obrázkem modelu v klidovém stavu na levé straně a na pravé straně připravený prostor pro data z modelu spolu s tlačítky, jejihž funkčnost bude popsána dále. Pokud nebudete zrovna připojeni k serveru, budete upozorněni varovným oknem (Obrázek 2). Po zavření varovného okna lze dále vidět stav připojení v pravé části s daty jako první hodnotu. Jakmile se k serveru připojíte, uvidíte kromě zmíněné signalizace i nové hodnoty u ostatních parametrů jako je uvedeno na Obrázku 3.



Obrázek 2: Chybová hláška při nepřipojení k serveru nebo při odpojení.



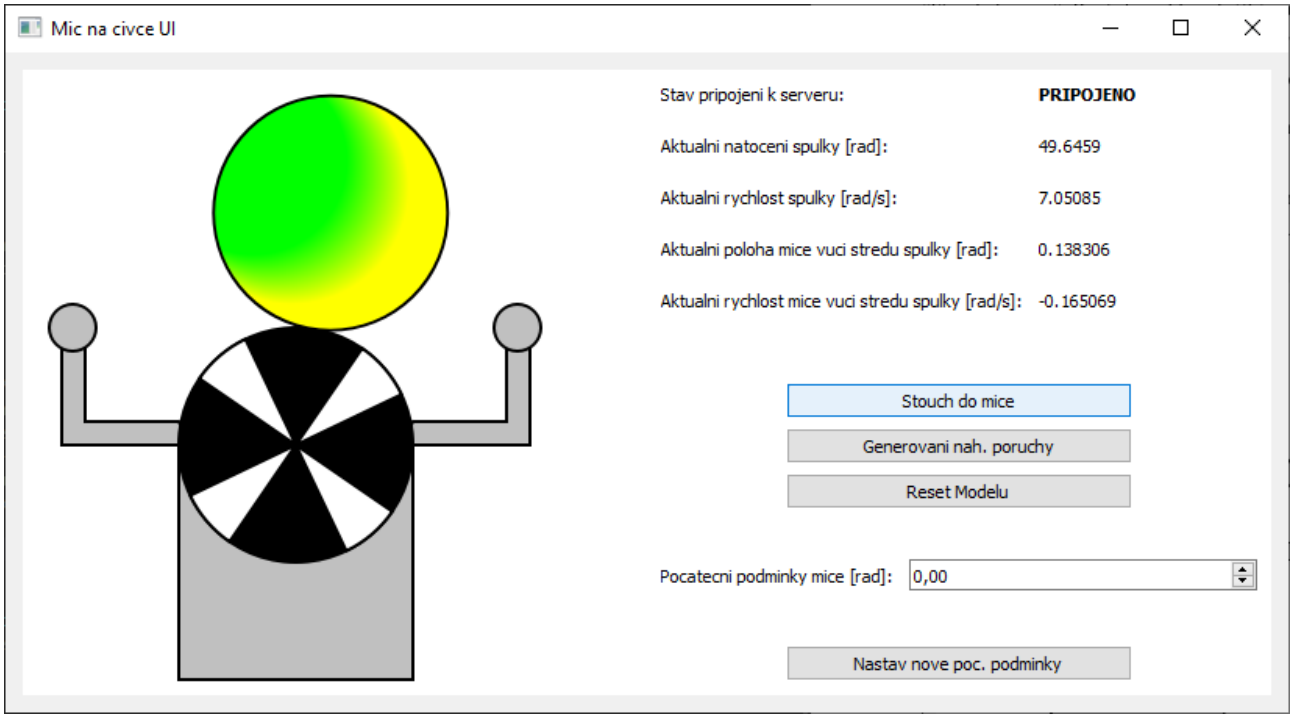
Obrázek 3: Klidový stav po připojení k serveru.

4.3 Interakce s modelem

Interakce s modelem je omezená na tlačítka v pravé části a jedno zadávací pole nad posledním tlačítkem. Hodnoty nad tlačítky jsou získávány jako výstup připojeného modelu a nelze je přepisovat.

Funkce tlačítek:
”Stouch do mice”— Po stisku aplikuje krátký puls na míč.
”Generovani nah. poruchy”— Přepínací tlačítko, které při aktivaci generuje náhodnou poruchu působící na míč. Při opakovaném stisku se tlačítko i porucha deaktivuje
”Reset Modelu”— Po stisku resetuje model do počátečního stavu za daných počátečních podmínkách.
”Nastav nove poc. podminky”— Tlačítko, které pracuje v koordinaci s jediným zapisovatelným polem těsně nad ním. Po jeho stisku odešle jeho obsah v podobě desetinného čísla do modelu jako nové počáteční podmínky modelu (poloha míče [rad]).

Na Obrázku 4 je zachycena aplikace za běhu při simulaci stavu modelu.



Obrázek 4: Aplikace zachycená v běhu po stisknutí tlačítka ”Stouch do mice”.

5 Závěr

Výsledná aplikace se chová tak jak bylo zadáno, vykonává interakce spolehlivě a animace zobrazuje správně na základě modelu. Po 30 minutách běhu zabírá zhruba 14 MB paměti RAM a s přibývajícím časem se tato hodnota příliš nemění.

Vývoj aplikace v Qt je velmi zajímavý pro jeho multiplatformní použití a i neoficiálně podpoře pythonu, který je osobně snažší pro psaní kódu. Díky neznalosti C++ jsem v určitých oblastech tápal, ale více jsem se dostal k řešením problémů s pamětí a zakusil vývoj aplikace s reálným využitím. Pro další vývoj bych však použil více nativní vývojové prostředí Qt, ve kterém lze některé grafické prvky řešit snáze.