TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

DIPLOMOVÁ PRÁCE

Libor Kadlec

Realizace prostředí pro zadávání dat a vizualizaci výsledků výpočtu optimálních trajektorií robotů v prostoru s překážkami

Liberec 2001

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

2612T – Elektrotechnika a informatika

Automatické řízení a inženýrská informatika

Realizace prostředí pro zadávání dat a vizualizaci výsledků výpočtu optimálních trajektorií robotů v prostoru s překážkami

Libor Kadlec

Vedoucí práce : Ing. Jan Cvejn
Technická univerzita Liberec

Rozsah práce a příloh :

Počet stran textu : 53

Počet obrázků : 10

Počet tabulek : 6

Počet stran příloh : 17

Datum: 21. 5. 2001

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Katedra softwarového inženýrství　　　　　　　　　　Školní rok: 2000/2001

ZADÁNÍ DIPLOMOVÉ PRÁCE

pro: Libora Kadlece

studijní program: 2612T – Elektrotechnika a informatika

obor: Automatické řízení a inženýrská informatika

Vedoucí katedry Vám ve smyslu zákona o vysokých školách č.111/1998 Sb. určuje tuto diplomovou práci:

Název tématu:

Realizace prostředí pro zadávání dat a vizualizaci výsledků výpočtu optimálních trajektorií robotů v prostoru s překážkami

Zásady pro vypracování:

1. Vytvoření uživatelského prostředí pro zadávání parametrů optimalizace
2. Vygenerování vstupních souborů pro program DynOpt na základě zadaných dat
3. Zobrazení a tisk výsledných průběhů kinematických veličin a řízení
4. Vytvoření prostředí pro vizuální zadávání tvaru robota a překážek a zobrazování 3d scény

Prohlášení :

„Místopřísežně prohlašuji, že jsem diplomovou práci vypracoval samostatně s použitím uvedené literatury."

V Liberci dne 21. 5. 2001

Libor Kadlec

Libor Kadlec

Poděkování :

Tímto bych chtěl poděkovat vedoucímu své diplomové práce Ing.Janu Cvejnovi a také Ing. Jiřině Královcové za cenné rady a podnětné připomínky, které přispěly k řešení daného problému.

Anotace

Tato práce je zaměřena na programování v Microsoft Visual C++. Výsledným programem je uživatelské prostředí pro zadávání dat a vizualizaci výsledků výpočtu optimálních trajektorií robotů v prostoru s překážkami. Výpočetní modul pro optimalizaci není součástí této práce.

Výpočetní modul DynOpt byl vytvořen tak, aby nebyl součástí uživatelského prostředí, takže je možné kdykoliv výpočetní modul změnit a použít stále stejné uživatelské rozhraní. Tato vlastnost je docílena tím, že optimalizační program DynOpt získává vstupní data pouze pomocí vstupního konfiguračního souboru a výstup výsledků je zapisován do souborů. Tímto způsobem je docíleno oddělení výpočetního modulu a uživatelského rozhraní.

Úkolem uživatelského rozhraní je generovat vstupní konfigurační soubor ze zadaných parametrů pro výpočetní modul a načítat data z výstupních souborů a provádět jejich vizualizaci. Vstupní konfigurační a výstupní simulační soubor mají strukturu, která připomíná zápis programu v jazyce C.

Vizualizace výsledků spočívá ve vykreslení grafů kinematických veličin a v simulaci pohybu robota. Simulace výsledného pohybu a geometrické vlastnosti robota jsou zobrazovány s využitím grafické knihovny OpenGL.

Annotation

This thesis specializes in programming in Microsoft Visual C++. The user interface for data inputting and the result visualization of the calculations of the optimal robot trajectories in the area of blocks is the resultant application. This project does not include optimization module.

The DynOpt computing module was created in the way not to be a component of a user interface, so it's possible to change this module any time and to use the current user interface. It was reached for this property in the way the DynOpt optimization application obtains the enter data only through the enter configuration file and the output data are being recorded to the files. This way achieves a separation of the computing module and the user interface.

Commission of the user interface is to generate the entered configuration file from the entered parameters for the computing module, the data reading from the output files and their visualization practicing. The entered configuration file and the output simulation file have a structure which reminds the notation of the application in C – language.

The result visualization consists in the depiction of the kinematics quantities diagrams and in the simulation of the robot motion. The resultant motion simulation and the robot's metrical properties are being presented using OpenGL (software interface for graphics hardware).

Obsah

1. Úvod

Podíl nasazení průmyslových robotů v technologických pracovištích se neustále rok od roku zvyšuje. V dnešní době se vlivem zkracování morálního zastarávání výrobků stává velmi aktuální problematika efektní automatizace. Řešit tento úkol mohou prostředky pružné automatizace, resp. programovatelná automatická zařízení. Pře seřízení pružné automatické linky na produkci nového výrobku spočívá pouze ve změně programů vložených do řídícího systému. Základ automatických flexibilních linek tvoří průmyslové roboty.

Z hlediska potřeb praxe plyne požadavek vytváření robotizovaných pracovišť s jedním nebo více řídicími systémy schopnými v krátkém čase změny pracovního programu s vlastním plánováním trajektorií pohybu robotů ze zadaných podmínek. Tyto trajektorie by měly být ve zvoleném smyslu optimální.

Cílem této práce je vytvořit uživatelské prostředí pro zadávání dat a vizualizaci výsledků výpočtu optimálních trajektorií robotů v prostoru s překážkami. V dřívější době byl vytvořen Ing. Cvejnem program DynOpt, který tuto optimalizaci vypočítává. Výpočetní modul DynOpt slouží pouze k experimentálnímu využití, umožňuje hledat optimální trajektorie zadaných systémů při zavedení okrajových podmínek a různých typů omezení.

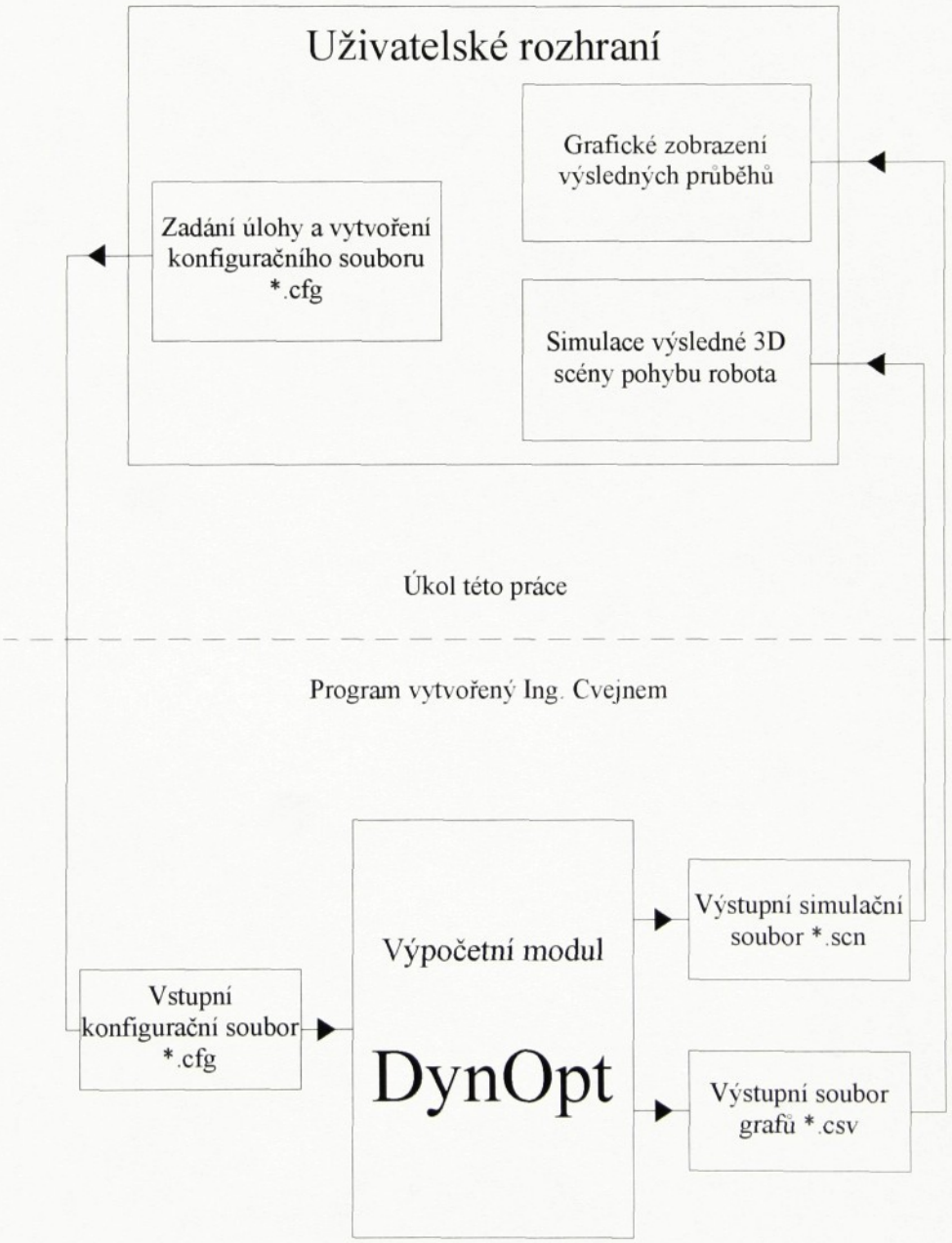
Program DynOpt byl vytvořen tak, aby neobsahoval uživatelské prostředí, takže je možné kdykoliv výpočetní modul změnit a použít stále stejné uživatelské rozhraní. Tato vlastnost je docílena tím, že optimalizační program DynOpt získává vstupní data pomocí textového souboru a výstup výsledků je zapisován do souborů. Tímto způsobem je vytvořena naprostá nezávislost výpočetního modulu na uživatelském rozhraní.

Pro toto prostředí byla vybrána platforma Microsoft z důvodu její masové rozšířenosti.

2. Teoretická část

2.1. Co je program WinOpt a DynOpt

Výpočetní modul DynOpt není součástí této práce, ale je výsledkem práce Ing. Jana Cvejna. Úkolem práce bylo vývojové prostředí WinOpt. Interakce programu WinOpt a DynOpt je znázorněna na *Obrázku č.2.1-1.*



Obr. č. 2.1-1 Vzájemná interakce programů DynOpt a WinOpt

2.2. Program DynOpt

Program DynOpt je výpočetním modulem pro program WinOpt. Pohyb dynamického systému je popsán diferenciální rovnicí ve tvaru :

R.2.2-1. $f(y^{(n)},....,\dot{y},y,t) = u$ kde :

y(t) – N-rozměrný vektor zobecněných souřadnic

u(t) – N-rozměrný vektor řídící veličiny

t – čas

Úkolem výpočetního modulu je nalézt funkci řízení **u(t)** tak, aby byl podél trajektorie minimalizován funkcionál :

R.2.2-2. $$J = \int_0^T f_0(y^{(n)},....,\dot{y},y,u,t)dt \rightarrow \min$$

Funkcionál je zde skalární veličinou, která vyjadřuje kvalitu procesu řízení. Vhodnou volbou kriteriální funkce $f_0(y^{(n)},....,\dot{y},y,u,t)$ lze formulovat libovolné požadavky.

Numerické řešení úloh na minimalizaci funkcionálu obecnými metodami není snadné. Nehledě na řadu jiných komplikací, je značně časově náročné. Optimální trajektorie se takto určí pouze v konečném (nevelkém) počtu boků a mezi nimi se pak provádí interpolace hladkou funkcí.

Z těchto důvodů bylo ve výpočetním modulu DynOpt od tohoto řešení ustoupeno a bylo zvoleno řešení, při kterém se pohyb systému předem aproximuje vhodnou známou funkcí konečného počtu parametrů (v tomto případě spline funkcí), jejichž optimální hodnoty hledáme. Tím se celý problém převede na hledání minima reálné funkce nevelkého počtu proměnných. Takto se získá pouze sub-optimální řešení, je však zřejmé, že vhodným způsobem aproximace trajektorie a volbou dostatečného počtu nezávislých proměnných se lze (teoreticky) k optimální trajektorii libovolně přiblížit.

2.2.1. Spline funkce

Termínem Spline se v Anglii označovala tenká pružná kovová šablona, sloužící pro kreslení křivek v lodním stavitelství. Šablona se prsty, závažími apod. vytvarovala tak, aby procházela žádanými body a poté se podle ní zakreslila křivka. Ukazuje se, že celá křivka, kterou se tímto postupem podařilo získat, byla po částech polynomem třetího stupně.

2.2.2. Aproximace trajektorie spline funkcí

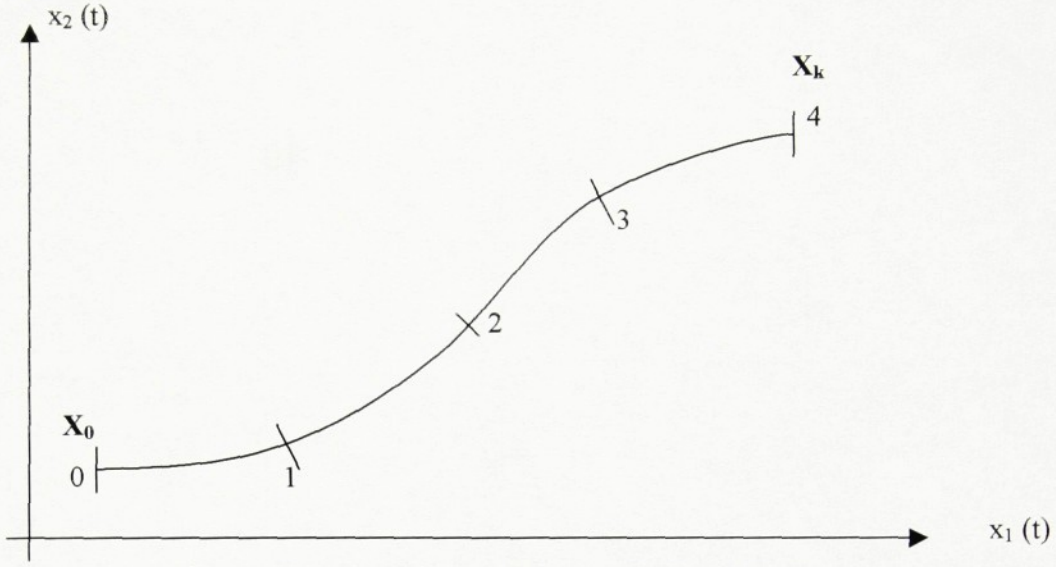
Uvažujeme *N*-rozměrný systém pohybující se z daného počátečního bodu (počátečního stavu) $\mathbf{x}_0$ do koncového bodu $\mathbf{x}_k$. Doba pohybu je rovna T. Rychlost pohybu systému (tj. časová derivace stavu systému) v počátečním bodě nechť je rovna $\mathbf{v}_0$ a v koncovém bodě $\mathbf{v}_k$.

Předpokládejme, že pohyb systému je vyjádřen spojitou funkcí času, která je navíc dostatečně hladká, tj. též její první, druhá, ... až ***k***-tá derivace podle času je spojitou funkcí.

Trajektorii systému rozdělme na ***n*** v čase ekvidistantních úseků. Hraniční body úseků nazvěme uzlovými body a označme je čísly **0, 1, ...*n*** (viz. *Obrázek č. 2.2.2-1*). Časy odpovídající těmto bodům označme $t_0, t_1, \ldots t_n$. Kde $t_0 = 0$ a $t_n = T$.

Pak **x(t)** může být složitou křivkou, zvolíme-li však dostatečně velké n, můžeme funkci **x(t)** v jednotlivých úsecích aproximovat např. polynomy a z nich sestavit funkci, která dostatečně přesně vystihuje průběh **x(t)** mezi body $\mathbf{x}_0$ a $\mathbf{x}_k$. Označme takovou funkci **S(t)** a nazvěme ji spline-funkcí.

V některých případech může být vhodné volit úseky trajektorie různé délky. Pro zjednodušení odvozených vztahů se však touto možností dále nezabývejme. Ve většině případů by ostatně takové zobecnění nemělo praktický význam.



Obr. č. 2.2.2-1 Příklad rozdělení trajektorie systému

Definujeme vektorovou spline-funkci **S(t)** takto:

R. 2.2.2-1 $S(t)=p_i(t)$ pro $t_{i-1} \le t \le t_i$; $1 \le i \le n$

kde $p_i(t)$ jsou vektorové polynomiální funkce

R. 2.2.2-2 $p_i^{(j)}(t_i) = p_{i+1}^{(j)}(t_i)$ pro $1 \le i \le n-1$ a $j = 0, 1, \ldots, k$

R. 2. 2.2-3
$$p_1 = x_0, p_n(T) = x_K$$
$$p_1(0) = v_0, p_n(T) = v_K$$

Rovnice R.2.2.2-2 představují podmínky spojitosti, resp. hladkosti spline v uzlových bodech a rovnice R.2.2.2-3 vyjadřuje okrajové podmínky.

Pozn.: Dále budeme uvažovat podmínky R.2.2.2-2 pouze do druhé derivace včetně (tj. $k=0,1,2$).

Polynomy $\mathbf{p_i(t)}$ jsou určeny konečným počtem koeficientů. Takto je celá spline funkce těchto parametrů. Rovnice R.2.2.2-2 a R.2.2.2-3 pak z prostoru všech možných parametrů vyčleňují podprostor parametrů volitelných, jejichž libovolnou volbou zůstanou tyto vztahy zachovány v platnosti. Označíme-li $\mathbf{N_s}$ celkový počet parametrů spline a $\mathbf{N_x}$ počet volitelných parametrů a **N** rozměr vektoru **x**, platí zřejmě:

R. 2.2.2-4 $N_x=N_s - N.[(k + 1)(n - 2) - 4]$

Je-li $N_x > 0$ je nutno připojit k podmínkám ještě další vztahy pro jednoznačné určení.

2.2.3. Kombinovaná metoda výpočtu optimalizace

Tato kombinovaná metoda optimalizace slučuje přednosti dvou různých algoritmů pro účely optimalizace úloh se složitými omezeními jako je optimální řízení dynamických systémů v prostoru s překážkami. Celá úprava algoritmu spočívá ve střídání určitého počtu kroků Partan a kroků globální optimalizace.Jejich podrobnější popis viz. lit. [1]. Metodu Partan je nutné upravit pro úlohy s omezením tím způsobem, že hledání v daném směru respektuje omezení. Úprava algoritmu jednorozměrného hledání se provede pomocí zlatého řezu. Dále na okraji omezení nemá smysl pokračovat dalšími kroky Partan, ale místo toho provést kroky globální optimalizace, které využívají náhodných směrů.

Vhodný poměr střídání kroků obou metod závisí na počtu volitelných parametrů. Pro větší počet je třeba více kroků globální optimalizace. Pro úlohy optimálního řízení bylo použito např. poměru 4:20 ve prospěch globální optimalizace. Poměr lze nastavit tak, že oba typy kroků dohromady zaberou přibližně stejně času. Přitom kroky Partan je třeba vzhledem k principu metody provést za sebou.

Toto sloučení způsobuje, že v blízkosti volného extrému se uplatňují kroky Partan a na hranici omezení spíše kroky globální optimalizace. Tato metoda má potom globální charakter a je univerzální.

2.2.4. Charakteristika programu DynOpt

Tento program je výsledkem práce Ing. Jana Cvejna. Tato aplikace byla vytvořena tak, aby byla pokud možno nezávislá na uživatelském prostředí pro zadávání parametrů, vizualizaci výsledků a platformě. Vytvoření uživatelského prostředí je úkolem této práce.

Oddělení od uživatelského prostředí je umožněno díku tomu, že veškerá vstupní data jsou načtena ze vstupního textového souboru a výsledkem jsou dva textové soubory. Výpočetní program nemá žádný jiný uživatelský vstup kromě možnosti přerušení výpočtu a neprovádí žádné grafické zpracování výsledků.

Vstupní parametry pro program DynOpt :

- parametry úlohy (konstanty, popř. diferenciální rovnice systému)
- okrajové podmínky problému (úplné nebo i neúplné)

- parametry optimalizačních metod
- omezení
- definice překážek a tvaru robota
- parametry ovlivňující generování výsledků

Program DynOpt pracuje v chráněném módu a nelze jej tedy spustit přímo z MS-DOSu i přesto, že na první pohled jako DOSovské vypadají. Není však problémem tento program spustit pod operačními systémy Windows 95, 98, Milenium, NT, 2000, OS/2 a Linux.

Vstupní soubor pro program DynOpt musí mít příponu *.cfg, ale jako parametr se zadává pouze jméno bez přípony (pouze v případě, že nebude spuštěn z programu WinOpt.

Příklad zápisu do příkazové řádky:

C:\>dynopt cv1

Program DynOpt generuje výstupní soubory se stejným jménem jako vstupní konfigurační soubor, ale pouze přípona těchto souborů je změněna na *.csv a *scn.

Kromě toho se automaticky generují ještě další soubory tgNNNN.csv, 3dNNNN.scn, itNNNN.lst po určitém počtu iteračních kroků, kde za „NNNN" v názvu se dosadí číslo kroku. Periodicita těchto výpisů je nastavena v konfiguračním souboru parametrem TRACEEVERY. Tento parametr byl implicitně zvolen na nulu, takže se generují pouze výsledné soubory. Soubory *.lst obsahují výpis parametrů a momentálních hodnot vnitřních veličin.

Při výpočtu se zobrazují nejdůležitější parametry úlohy a informativní hodnoty o iteračním kroku, aktuální hodnotě funkcionálu, aktuálním úbytku funkcionálu a době výpočtu.

2.2.5. Reprezentace systému a překážek v počítači

Reprezentace tvaru robota a překážek byla navržena Ing. Janem Cvejnem již při práci na výpočetním modulu DynOpt. Navrhl zde nahradit model robota a překážek parametrizovatelnými mnohostěny, které jsou blíže popsány v kap. .Např. válci, bude odpovídat *n*-boký hranol, jehož parametrem je výška, poloměr, počet stran, pozice referenčního bodu a orientace v prostoru. U robotů jsou polohy členů většinou vyjadřovány relativně vzhledem k předcházejícímu členu, proto je vhodné tímto způsobem parametrizovat i model robota.

Z toho tedy vyplývá, je-li více drobných těles seskupeno, je výhodné měnit polohu celku a nikoliv každého detailu. Tyto důvody vedou k reprezentaci robota i překážek stromovou datovou strukturou. Větev je definována tak, že obsahuje souřadnice svého referenčního bodu vůči nadřazené větvi, orientaci a seznam prvků, jimiž jsou buďto tělesa nebo další podřízené větve. Větev, která není obsažena v žádné jiné větvi je kořenem stromu tělesa, nebo další podřízené větve. Větev, která není obsažena v žádné jiné větvi je kořenem stromu. Tato stromová struktura je zobrazena na *obrázku č.2.2.5-1* .

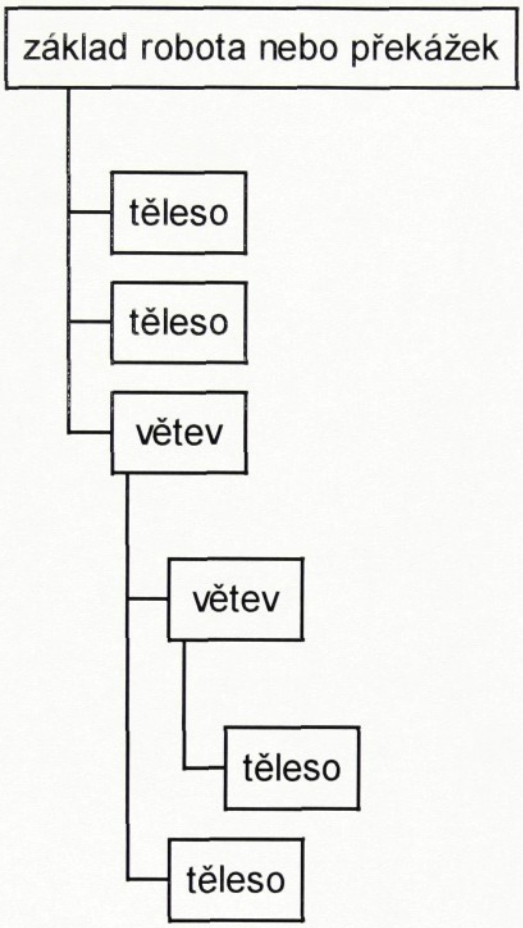
Protože definice kořenového stromu je rekurzivní, má tuto povahu i většina operací, která jsou na něm prováděna. Pro určení absolutních souřadnic a absolutní orientace všech těles obsažených v dané větvi včetně všech podvětví je třeba nejdříve určit absolutní souřadnice a absolutní orientaci dané větve. K těmto hodnotám se pak přičtou relativní souřadnice a orientace dané položky.

$$R_{ix} = R_x + r_{ix}$$
$$R_{iy} = R_y + r_{iy}$$
$$R_{iz} = R_z + r_{iz}$$

R.2.2.5-1

$$A_{ix} = A_x + \alpha_{ix}$$
$$A_{iy} = A_y + \alpha_{iy}$$
$$A_{iz} = A_z + \alpha_{iz}$$

Velká písmena označují absolutní souřadnice a úhly natočení, malá relativní. Relativní souřadnice a orientace všech těles a větví jsou známy pro každý čas t a jsou dopočítány z geometrických vztahů definujících pohyb robota. Absolutní souřadnice kořene stromu jsou rovny jeho relativním souřadnicím. Z těchto údajů lze rekurzivně dopočítat



Obr. č.2.2.5-1 Příklad stromové struktury pro reprezentaci robota a překážek.

absolutní souřadnice a orientaci všech těles.

Stromová struktura je přirozeným vyjádřením vzájemných vztahů těles robota i překážek a umožňuje přehledným způsobem vytvářet model.

2.2.6. Zadání tvaru překážek a systému

Tvar je možné zadávat v okně návrhu do stromové struktury. Je důležitý pro zobrazení systému ve 3D simulaci a hlavně pro optimalizaci s požadavkem vyhýbání se překážkám.

Každé těleso má svoji vlastní množinu parametrů, ale všechna tělesa obsahují parametry z *Tabulky č.2.2.6-1* :

Pozice	relativní pozice referenčního bodu tělesa vůči nadřazenému
Rotace	relativní pootočení referenčního bodu tělesa vůči nadřazenému
Třída	index třídy pro detekci vlastních těles systému

Tab. č.2.2.6-1 Společné parametry těles

JMÉNO POLOŽKY	JMÉNO V KONFIGURAČNÍM SOUBORU	PARAMETRY	POPIS
Blok	BLOCK	a, b, c	kvádr a x b x c, umístěný středem v počátku souřadnic
Center Blok	CENTERBLOCK	a, b, c	kvádr a x b x c, umístěný v počátku souřadnic
Hranol	PRISM	r, n, v	pravidelný hranol, n bočních stěn, r je poloměr kružnice opsané podstavám, v je výška
Kolmý Hranol	RIGHTPRISM	a, b, v	kolmý hranol s podstavou pravoúhlého trojúhelníka o stranách a, b, v výška
Jehlan	SPIRE	r, n, v	pravidelný jehlan, n bočních stěn, r je poloměr kružnice opsané podstavě. v výška
Komolý Jehlan	TRUNCSPIRE	r, n, v0, v	komolý jehlan vytvořený z jehlanu (r, n, v0) seříznutím na výšku v
Kolmý Jehlan	RIGHTSPIRE	a, b, v	kolmý jehlan s podstavou pravoúhlého trojúhelníka o stranách a, b, v výška
větev	BRANCH	seznam těles	větev obsahující další (podřízená) tělesa

Tab. č.2.2.6-2 Elementární tělesa

V tabulce jsou rozepsána elementární tělesa, které je možné použít pro zadání tvaru systému a překážek. Těleso BRANCH není ve skutečnosti těleso, ale větev obsahující další tělesa včetně BRANCH, která jsou považována za parametry BRANCH. Pokud není zadána pozice těles parametry v dialogu vlastností tělesa, bere se implicitní nulová hodnota.

Poloha těles daná parametry pozice a rotace v dialogu vlastností tělesa může být dynamicky měněna rovnicemi v části pohybu na stejném dialogu. Pokud se pozice tělesa mění dynamicky rovnicemi, jsou pevné hodnoty pozice přemazány.

2.2.7. Zadání definičních rovnic

Pomocí definičních rovnic definujeme systém, popř. kriteriální funkci nebo omezení. Tento popis se vyskytuje v parametrech úlohy na záložkách **Systém**, **Kritérium** a **Omezení**.

Každá z rovnic obsahuje na levé straně proměnnou, dále znak „=" a na pravé straně výraz skládající se z proměnných, aritmetických operací, funkcí a závorek. Každá rovnice se chápe jako samostatná položka a je tedy ukončena středníkem. Argumenty funkcí (je-li jich více než jeden) jsou odděleny čárkou.

V případě, že se na levé straně před jménem proměnné nachází klíčové slovo „const", je tato proměnná vyhodnocena pouze jednou, před začátkem výpočtu (v případě kriteriální funkce před každou integrací).

Výpočet rovnic se vždy provádí v pořadí shora dolů. Jedné proměnné je možno přiřadit hodnotu nebo výraz vícekrát.

Proměnné jsou dvojího typu:

- vnitřní proměnné
- pomocné proměnné

Jména vnitřních proměnných mají prefix „$". Pomocné proměnné, které slouží pro uložení mezi výpočtu a zpřehlednění zápisu rovnic, není třeba předem deklarovat. Stačí pouze výraz na pravé straně přiřadit jménu, které nebylo dosud použito. Jména pomocných proměnných nejsou na rozdíl od systémových proměnných sdílena mezi definičními rovnicemi systému, kriteriální funkce nebo pohybu tvaru.

Jsou definovány tyto vnitřní proměnné :

jméno	značení	význam
$N	n	řád dynamické systému (řád nejvyšší derivace)
$M	m	počet stupňů volnosti
$S	-	počet transformačních rovnic
$SDISP	-	počet transformačních rovnic, které se budou zapisovat/zobrazovat
$X(i, j)	$y_j^{(i)}$	hodnota i-té derivace polohy v aktuálním bodě trajektorie j-tého členu
$Y(j)	y_j	poloha v aktuálním bodě trajektorie j-tého členu
$V(j)	$\dot{y}_j$	rychlost v aktuálním bodě trajektorie j-tého členu
$A(j)	$\ddot{y}_j$	zrychlení v aktuálním bodě trajektorie j-tého členu
$U(j)	u_j	řízení j-tého členu
$T	t	aktuální čas
$R(i, X)	ϕ_{ix}	i-tá transformační rovnice, x-souřadnice
$R(i, Y)	ϕ_{iy}	i-tá transformační rovnice, y-souřadnice
$R(i, Z)	ϕ_{iz}	i-tá transformační rovnice, z-souřadnice
$S(X)	s_x	celková dráha systému, x-souřadnice
$S(Y)	s_y	celková dráha systému, y-souřadnice
$S(Z)	s_z	celková dráha systému, z-souřadnice
$F	f_0	integrand funkcionálu
$F0	φ_t	počáteční člen funkcionálu
$FT	φ_0	terminální člen funkcionálu
$CONDITION	-	podmínka přípustnosti
$POS(i, X)	r_{ix}	relativní poloha i-tého tělesa, x-souřadnice
$POS(i, Y)	r_{iy}	relativní poloha i-tého tělesa, y-souřadnice
$POS(i, Z)	r_{iz}	relativní poloha i-tého tělesa, z-souřadnice
$ROT(i, X)	α_{ix}	relativní pootočení i-tého tělesa, x-souřadnice
$ROT(i, Y)	α_{iy}	relativní pootočení i-tého tělesa, y-souřadnice
$ROT(i, Z)	α_{iz}	relativní pootočení i-tého tělesa, z-souřadnice

Tab. č.2.2.7-1 : Vnitřní proměnné programu DynOpt

V *Tabulce č.2.2.7-2* jsou uvedeny použitelné operace mezi proměnnými a číselnými konstantami včetně priorit. Operace s vyšší prioritou jsou vyhodnoceny před operacemi s prioritou nižší.

operace	**priorita**	**význam**
()	6	závorky – změna pořadí vyhodnocování
^	5	umocnění
-	4	negace
funkce	3	funkce s argumentem bez závorek
*, /	2	násobení dělení
+, -	1	sčítání, odčítání

Tab. č.2.2.7-2 : Priorita operací

Dále je uveden seznam použitelných funkcí. Funkce mohou mít jeden nebo dva argumenty, které jsou uvedeny v závorkách za jménem funkce a odděleny čárkou. Funkce s jedním argumentem lze psát bez závorek (např. *sin y(0)*). Operace vyhodnocení argumentu má pak prioritu 3 v *Tab. č.2.2.7-2* . Funkce bez argumentu, tj. matematické konstanty, jsou psány bez závorek.

funkce	**argumentů**	**význam**
pow	2	umocnění
sqrt	1	druhá odmocnina
sqr	1	druhá mocnina
exp	1	e^x
ln	1	přirozený logaritmus
pow10	1	10^x
log10	1	dekadický logaritmus
sin	1	sinus
cos	1	kosinus
tan	1	tangens
asin	1	arcussinus
acos	1	arcuscosinus
atan	1	arcustangens

abs	1	absolutní hodnota
sign	1	znaménko výrazu
int	1	celá část výrazu
eta	1	$\eta(t)$
ge	2	arg 1 $\geq$ arg 2 (vrací 0 nebo 1)
gt	2	arg 1 > arg 2 (vrací 0 nebo 1)
le	2	arg 1 $\leq$ arg 2 (vrací 0 nebo 1)
lt	2	arg 1 < arg 2 (vrací 0 nebo 1)
eq	2	$\lvert$ arg 1 – arg 2 $\rvert < 10^{-10}$ (vrací 0 nebo 1)
min	2	min (arg 1, arg 2) (vrací 0 nebo 1)
max	2	max (arg 1, arg 2) (vrací 0 nebo 1)
and	2	vrací 1 pokud oba operandy nenulové
or	2	vrací 1 pokud alespoň jeden operand je nenulový
not	1	negace
xor	2	exklusive – or
pi	0	konstanta π
RadToDeg	1	převod radiánů na stupně
DegToRad	1	převod stupňů na radiány
Idt	1	integrál výrazu 0 do t_f

Tab. č.2.2.7-3 : Přehled použitelných funkcí

Všechny funkce, včetně logických, pracují s reálnými čísly. Za logickou 0 je považováno číslo 0 a za logickou 1 je považováno libovolné nenulové číslo.

2.2.8. Třídy těles pro zjišťování kolize

Budeme uvažovat dvě množiny těles. Množina **R** bude označovat množinu těles robota a **P** množinu těles překážek. Potom tedy platí, že se robot vyhýbá překážkám za předpokladu, že platí

R. 2.2.8-1 $$P_i \cap R_j = 0$$

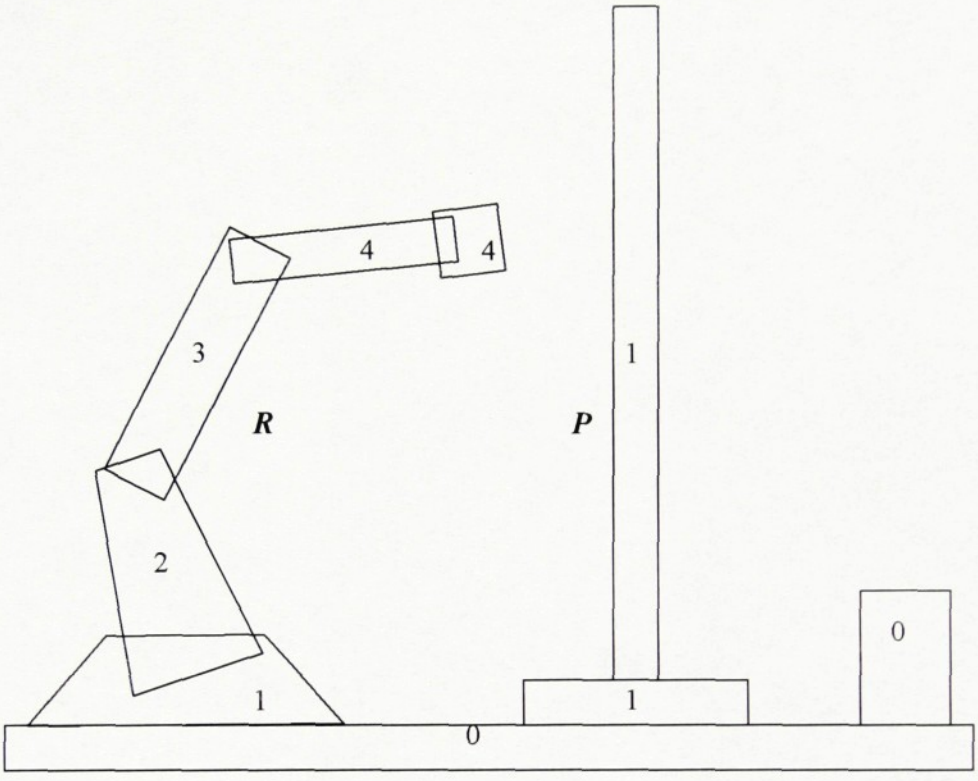
pro $i = 1 \ldots n, j = 1 \ldots m$

Tato podmínka ještě není postačující, protože je zde ještě potřeba zajistit, aby nedocházelo ke kolizi vlastních těles robota (např. uchopovací hlavice o podstavce). V některých případech je tato kolize naopak povolena (např. dva sousední členy spojené osou).

Tento problém je v programu vyřešen tak, že množina těles robota je rozdělena do tříd, kde každá třída je opatřena jedinečným indexem. Třídy s po sobě jdoucími členy mají rozdíl indexů v absolutní hodnotě roven 1. Neaktivním tělesům robota i překážek (např. skrytá tělesa), která nemají být vůbec testována, je přiřazen index roven 0. K podmínce R. 2.2.8-1 pak ještě přibývá podmínka R. 2.2.8-2.

R. 2.2.8-2 $$R_i \cap R_j = 0 \text{ jestliže } |I(R_i) - I(R_j)| > 1$$
a současně $I(R_i) \neq 0, (R_j) \neq 0$
pro $i,j = 1 \ldots n$

Příklad rozdělení do tříd je na *Obrázku č.2.2.8-1* .



Obr. č.2.2.8-1 Příklad rozdělení těles robota a překážek do tříd

2.3. OpenGL

2.3.1. Stručně o historii OpenGL

V roce 1993 vznila první verze této knihovny OpenGL 1.0 v dílnách společnosti Silicon Graphics. Je nástupcem knihovny IRIS Graphics Library. Každá knihovna, která má nést název OpenGL musí být prověřena konsorciem ARB (Architecture Review Board), což je konsorcium zaručující správnost OpenGL. Bylo vytvořeno předními počítačovými firmami (SGI, DEC, IBM, Intel, MS, Intergraph, E&S).

Knihovna je úzce svázána s použitím grafických akcelerátorů.

2.3.2. Možnosti OpenGL

Knihovna OpenGL

- je nezávislá na operačním systému a na HW
- slouží pro psaní grafických aplikací
- nepodporuje okénka
- není objektově orientovaná
- byla uznána jako obecný standard

Množina základních primitiv je velmi redukovaná

- bod
- úsečka
- polygon
- bitmapa

OpenGL nabízí

- řešení viditelnosti objektů ve 3D
- různé druhy transformace grafických objektů
- definice barev popř. stanovení barevných vlastností povrchů
- osvětlení scény (min. 8 světel) včetně možnosti stanovení vlastností jednotlivých světel

- výpočet barvy v interakci se světly
- vytvoření průhledných objektů
- interpolace barvy
- antialiasing
- zapnutí mlhy
- mapování textur
- evaluátory pro výpočet Beziérových křivek a ploch
- možnost sdružení několika příkazů do jediného makropříkazu pro jeho pozdější vyvolání

Součásti knihovny

- OpenGL - malé množství entit (reprezentace nejlépe použitelná pro grafické akcelerátory)
- GLU - OpenGL Utility Library - projekce textur, teselace (rozklad na trojúhelníky), rendering B-spline křivek, základní elementy popsatelné kvadratickými rovnicemi (válce, koule, disky)
- GLUT - OpenGL Utility Toolkit

K vytvoření opravdu realisticky vypadajících scén je však nutno použít jiných metod jako je např. metoda sledování paprsku (ray tracing).

3. Praktická část

3.1. Charakteristika aplikace

Aplikace WinOpt byla vytvořena ve vývojovém prostředí Microsoft Visual C++ za použití aplikačního systému (application framework) knihovny MFC (Microsoft Foundation Class Library verze 4.21). Funkční kostra aplikace byla vytvořena AppWizardem jako SDI (Single Document Interface), tzn. že všechna okna mají pouze jeden společný dokument. Obsahuje celkem tři okna, která sdílejí právě jeden již zmíněný dokument. V následující *tablce č.3.1-1* je rozdělení použitých tříd k jednotlivým oknům.

COptDoc	společná třída dokumentu pro celou aplikaci
Okno pro vykreslování výsledných grafů – slouží pro zobrazování výsledných průběhů kinematických veličin a řízení	
CMainFrame	hlavní rámcová třída celé aplikace
COptView	třída pohledu, která je odvozena od třídy Cview
Návrhové okno – slouží pro zadávání tvaru robota a pro nastavení parametrů optimalizace.	
CCreateFrame	rámcová třída okna
CCreateView	třída pohledu, která je odvozena od třídy CFormView
CRenderView	třída pohledu, která je odvozena od třídy CView
Simulační okno – je určeno pro zobrazení výsledné 3D scény pohybu robota	
CSimFrame	rámcová třída okna
CSimView	třída pohledu, která je odvozena od třídy CView

Tab. č. 3.1-1: Rozdělení tříd k jednotlivým oknům

3.1.1. Popis všech tříd programu

C*Frame :**

jsou rámcové třídy, které se od sebe liší pouze v drobných detailech, jako například u CCreateFrame je vytvářeno dělené okno (Splitter Window). To znamená, že okno je rozděleno na dvě části, z nichž každá má třídu pohledu odvozenou od různých tříd. V tomto případě je to od CView a CFormView. Odvozeny jsou od třídy CframeWnd.

COptView :

třída pohledu, která slouží k vykreslování jednotlivých průběhů kinematických veličin a řízení. Pro vykreslování se využívá volání jednotlivých funkcí GDI. Tato třída je odvozena od třídy Cview.

Třída COptDoc se postará o načtení dat ze souboru grafů do dvourozměrného dynamického pole. Potom se zjistí hlavičky jednotlivých grafů a ty se následně zapíší do ComboBoxu v panelu nástrojů. Zde potom uživateli umožňuje měnit jednotlivé grafy.

Dále potom tato třída zajišťuje nastavení automatických měřítek souřadnic a vypsání legendy ke grafům.

CRenderView a CSimView :

jsou třídy pohledů, které jsou odvozeny od CView a slouží pro vykreslování 3D scén pomocí OpenGL. Grafické výstupy OpenGL směřují do okna. Toto okno musí mít předem vytvořený „render context" (RC) - je odlišný od kontextu zařízení „device context" (DC).
Pro provedení libovolného příkazu OpenGL je nutno nastavit příslušný RC jako aktivní. Po ukončení kreslení jej deaktivujeme.

DC - obsahuje informace potřebné pro GDI
RC - obsahuje informace potřebné pro OpenGL

CRenderView:

v této třídě se zajišťuje okamžité vykreslení robota při návrhu obecného tvaru. Jsou zde metody, které pomocí polygonů vykreslují základní geometrické tvary systému v OpenGL.

CSimView:

je využívána pro simulaci výsledné 3D scény pohybu robota. Výsledek výpočtu je načten z výstupního souboru „*.scn" a je zde snímek po snímku simulován.

CCreateView:

tato třída je odvozena do CFormView a je na tomto okně, které se chová jako dialog umístěn objekt CTreeCtrl. V této stromové struktuře se zadává obecný tvar robota a překážek. Zajišťuje zde veškeré operace s tělesy jako je např. přidání nového, odebrání, editace vlastností tělesa apod.

CCfgSheet, CxxxPage :

tyto třídy spolu vytvářejí dialog typu Property Sheet (vlastnosti umístěné na záložkách). Na těchto záložkách se zadávají parametry pro výpočetní modul DynOpt.

CClientCapture :

slouží pro překreslení scény v OpenGL do DIBu (Device Indepedent Bitmap) a pro následný tisk scény.

CColorComboEx :

od této třídy jsou odvozeny prvky ComboBox, které jsou určeny pro výběr barvy.

CColorDlg :

tento dialog umožňuje uživateli vybrat barvy jednotlivých průběhů kinematických veličin. Tato nastavení se ukládají do registrů Windows pro každého uživatele zvlášť.

CCompColorDlg :

zde si uživatel vybírá barvu vybraného a nevybraného tělesa systému v TreeControlu. Tato nastavení se opět ukládají do registrů Windows.

CComponent :

struktura sloužící k popisu jednotlivých těles systému.

CCompSheet, CxxxPage :

tyto třídy spolu vytvářejí dialog typu Property Sheet (vlastnosti umístěné na záložkách). Na těchto záložkách se zadávají parametry těles robota.

CPoint3D :

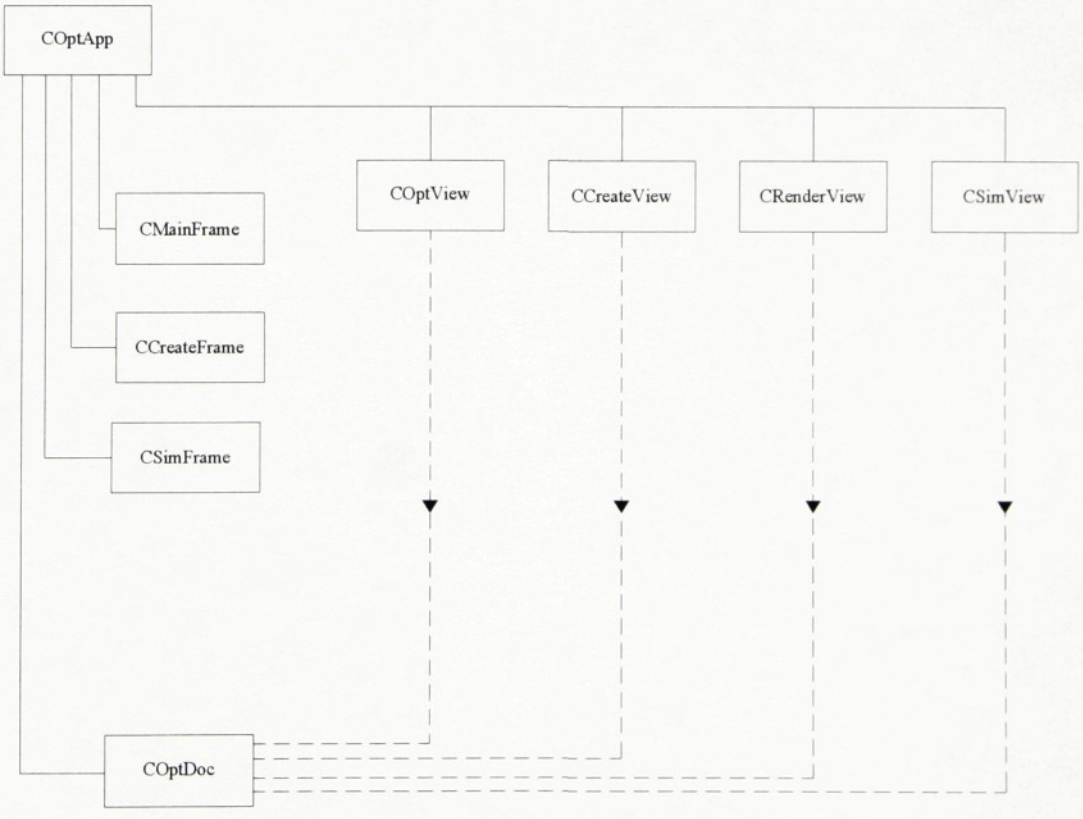
struktura pro popis bodu v prostoru 3D.

CPosition :

struktura popisující pozici a rotaci komponenty robota.

CGraphInfo :

struktura pro záznam vlastností o jednotlivých stopách v grafech.



Obr. č. 3.1-2 Vztahy mezi objekty.

3.2. Stručný popis programu WinOpt

Program WinOpt je výsledkem této práce. Je to uživatelské rozhraní pro zadávání dat a vizualizaci výsledků výpočtu optimálních trajektorií z výpočetního modulu DynOpt.

Program WinOpt umožňuje uživateli:

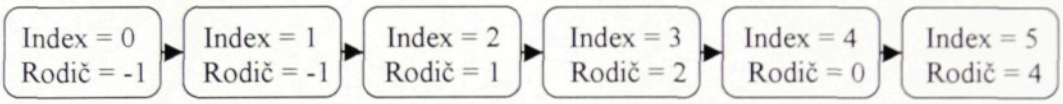
- navrhnout tvar robota
- navrhnout překážky, které ho omezují v pohybu
- zadat parametry úlohy
- okrajové podmínky
- parametry optimalizačních metod
- omezení
- vygenerování konfiguračního souboru z předešlých parametrů pro výpočetní modul DynOpt
- spuštění výpočetní modulu DynOpt s tímto souborem, jako parametrem
- grafické zobrazení výsledných průběhů kinematických veličin a řízení z výstupního souboru
- simulaci výsledné 3D scény pohybu robota

Aplikace WinOpt byla vytvořena v prostředí Microsoft Visual C++. Tento program tedy není na platformě tak nezávislý, jako v případě programu DynOpt, ale byl napsán tak, aby byl spustitelný na platformách Windows 95, 98 a NT , což nemusí být vždy pravda.

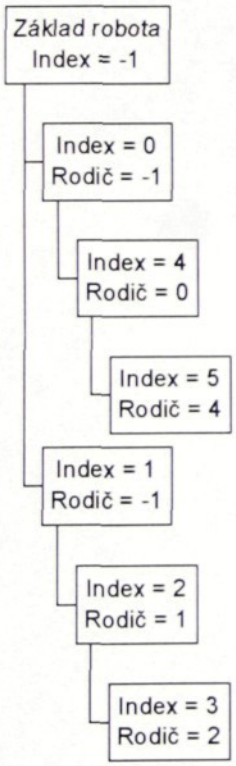
3.3. Reprezentace těles v programu WinOpt

Tělesa, jak již bylo popsáno v kap. 2.2.5, jsou ve stromové struktuře. V programu WinOpt jsou tělesa uložena v jednorozměrném dynamickém poli. Každé těleso je popsáno strukturou, ve které jsou informace týkající se tělesa a ještě také informace pro převod lineární struktury na stromovou. K tomuto převodu postačuje pouze informace o rodiči daného tělesa. Příklad takového převodu je na *Obrázku č.3.3-1* .

Lineární struktura:



Stromová struktura:



Obr. č. 3.3-1 . Převod lineární struktury na stromovou

V programu je jedna lineární struktura, která se převádí ve dvě stromové. Jedna pro tělesa robota (rodičem je index = -1) a druhá pro tělesa překážek (rodičem je index = -2).

3.4. Vstupní, výstupní soubory a jejich zpracování

3.4.1. Vstupní konfigurační soubor „*.cfg“

Popis souboru

Konfiguračním souborem pro výpočetní modul je textový soubor, který obsahuje veškeré vstupní informace. Soubor musí mít příponu „*.cfg“. Tento soubor je generován automaticky při uložení.

Struktura připomíná zápis programu v jazyce C. Soubor obsahuje pojmenované položky, číselné hodnoty a komentáře. Přitom pojmenované položky mohou být strukturované, tj. mohou obsahovat další pojmenované položky nebo hodnoty.

V případě jednoduchých pojmenovaných položek je přiřazení hodnoty zadáno pomocí znaku „=“, přiřazení je ukončeno pomocí středníku „;“.

U strukturovaných pojmenovaných položek je pro přiřazení hodnot použito jako oddělujících znaků složených závorek „{“, „}“ , uvnitř kterých jsou uvedeny jednotlivé pojmenované položky nebo hodnoty.

Komentář je libovolná část textu vymezená znaky „/*“ , „*/“ . Tato část textu se při zpracování konfiguračního souboru ignoruje. Alternativní možností je použít „//“ pro označení jako komentář textu od aktuální pozice do konce řádku. Tyto komentáře nelze přidávat do tohoto souboru pomocí programu WinOpt, ale bylo by potřeba je zde ručně dopsat v textovém editoru.

Výpočetní modul DynOpt nerozlišuje malá a velká písmena ve čtení pojmenovaných položek. Prázdné řádky a nadbytečné mezery se ignorují.

Dále je zde příklad vytvořeného konfiguračního souboru programem WinOpt:

```
ITERATOR
{
        NT=10;
        NS=2;
        Tk=2;
        DIV=8;
        EPS=1e-4;
        ITMAX=5000;
        ITEQUAL=40;
        ITMAXTFIX=20;
        D0TK=1;
```

```
        DKTK=1e-5;
        TRACEEVERY=50;
        REL_GLOBAL=0;
        REL_GRAD=4;
         DIM X0{2;3}
        X0 {{0;0;0};{0;0;0}}
        DIM XK{2;3}
        Xk {{1;1;1};{0;0;0}}

        BASE { MODE=FINITE; }

        OPTMETHOD
        {
                GLOBAL
                {
                        L0=20;
                        Lmin=1e-5;
                        kontr=0.65;
                        nRetries=30;
                }
                GRADIENT
                {
                        d0=0.1;
                        dk=1e-6;
                        ihmax=50;
                }
        }
}

SYSTEM=ANGULAR
{
        m1=34;m2=16;m3=39;m4=35;m5=20;
        l1=0.90;l2=0.90;l3=0.5;l4=0.6;l5=0.6;h=1;
        alfa=3.14159;

        km{0.68;0.68;0.68;}
        im{150;150;150;}
}

CRITERIA=SQRU;
SPACE=NULL;
```

Na pořadí, ve kterém jsou položky uvedeny v souboru nezáleží. V případě chybně zadaných parametrů nebo syntaktické chyby se vypíše hlášení, u syntaktických chyb i příslušný řádek a program dále nepokračuje.

Vytvoření konfiguračního souboru

V předchozí kapitole je příklad konfiguračního souboru pro předdefinovaný typ robota. Tento konfigurační soubor není pro vytvoření náročný. Značně náročnější je vytvoření souboru pro obecný typ robota. V souboru se zapíše „SYSTEM=GENERIC" a následuje výpis stromové struktury těles a jejich pohybu. Dále následuje příklad zápisu cylindrického robota:

```
FIGURE
{
        BLOCK { a=0.7;b=0.7;c=0.3;LEVEL=1; }
        BLOCK { a=0.15;b=0.15;c=2;LEVEL=1; }
        BRANCH
        {
                CENTERBLOCK { a=0.3;b=0.3;c=0.3;LEVEL=1; }
                BRANCH
                {
                        BLOCK { a=0.1;b=0.1;c=2;LEVEL=2; }
                        CENTERBLOCK { a=0.2;b=0.2;c=0.2;LEVEL=3; }
                }
        }
}

MOTION
{

        const l6=0.6;
        const e=0.15;
        const h=1;

        fi0=$Y(1);
        z0=$Y(0);
        x0=$Y(2);

        $POS(3,Z)=z0+h;
        $ROT(3,Z)=fi0*180/3.14159;

        $POS(6,X)=l6+x0;
        $POS(6,Y)=e;
        $ROT(6,Y)=-90;

        $POS(7,X)=l6+x0;
        $POS(7,Y)=e;
}
```

Pro vytvoření tohoto zápisu byl použit rekurzivní algoritmus pro prohledávání stromové struktury do šířky.

Hlavní část rekurzivního algoritmu pro zápis stromové struktury robota do konfiguračního souboru :

```
void COptDoc::WritePos(int iParent, int tabs, strstream& stream, BOOL bBranch, BOOL bFigure)
{
        BOOL bFirst = TRUE;
        CString str;
        //projede všechny tělesa v lineární struktuře
        for (int i = 0; i < m_CountOfComp; i++) {
                if (m_compArray[i].iParent == iParent) {
                        //pokud je těleso ve větvi dané rodičem
                        if (bBranch && bFirst) {
                                WriteTabs(tabs - 1, stream);
                                stream << "BRANCH" << '\n';
                                WriteTabs(tabs - 1, stream);
                                stream << '{' << '\n';
                                bFirst = FALSE;
                                //zapíše pohyb tělesa
                                WriteMotion(m_iPos, i, TRUE);
                                m_iPos++;
                        }
                        //vytvoření tabelátoru na začátku řádku
                        WriteTabs(tabs, stream);
                        //vypsání parametrů tělesa
                        WriteBlock(i, stream, bFigure);
                        if (m_compArray[i].motion.GetLength() > 0) {
                                WriteMotion( m_iPos, i, FALSE);
                        }
                        m_iPos++;
                        //rekurzivní volání
                        WritePos( i, tabs + 1, stream, 1, bFigure);
                }
        }
        if (!bFirst) {
                WriteTabs(tabs - 1, stream);
                stream << '}' << '\n';
        }
}
```

3.4.2. Výstupní soubor pro simulaci „*.scn“

Popis souboru

V tomto souboru jsou obsažena tělesa definující tvar robota a překážek. Tělesa jsou obsažena vícekrát (pro každý uzlový bod) z důvodů animace. Dále je zde ještě trajektorie vybraných bodů. Zpravidla koncový bod robota, ale v konfiguračním souboru výpočetního modulu DynOpt lze zadat i libovolné jiné trajektorie (Transformační funkce). Uzlové body jsou zobrazeny zvýrazněně.

Formát souboru *.scn je stejný jako formát konfiguračního souboru *.cfg až na jména položek. Následuje příklad výstupního souboru pro pohyb jednoho tělesa ve dvou snímcích a části trajektorie se dvěma body:

```
SLIDE {
        FACE {
                COLOR=3;
                FIGURE=BLOCK;
                POINTS {{-0.15;-0.15;0.85}{0.15;-0.15;0.85}{0.15;0.15;0.85}{-0.15;0.15;0.85}{-0.15;-
                        0.15;1.15}{0.15;-0.15;1.15}{0.15;0.15;1.15}{-0.15;0.15;1.15}}
                EDGES {{0;1}{1;2}{2;3}{3;0}{4;5}{5;6}{6;7}{7;4}{0;4}{1;5}{2;6}{3;7}}
                POLYS {{0;1;2;3}{4;5;6;7}{0;1;5;4}{1;2;6;5}{2;3;7;6}{3;0;4;7}}
        }

}

SLIDE {
        FACE {
                COLOR=3;
                FIGURE=BLOCK;
                POINTS {{-0.145;-0.154;0.876}{0.154;-0.145;0.876}{0.145;0.154;0.876}{-
                        0.154;0.145;0.876}{-0.145;-0.154;1.18}{0.154;-
                        0.145;1.18}{0.145;0.154;1.18}{-0.154;0.145;1.18}}
                EDGES {{0;1}{1;2}{2;3}{3;0}{4;5}{5;6}{6;7}{7;4}{0;4}{1;5}{2;6}{3;7}}
                POLYS {{0;1;2;3}{4;5;6;7}{0;1;5;4}{1;2;6;5}{2;3;7;6}{3;0;4;7}}
        }
}

PATH
{
        POINTS{
                {0.6;0.0897;1}
                {0.6;0.09;1}
                {0.601;0.091;1}
                {0.602;0.0927;1}
                {0.603;0.0949;1.01}
                {0.605;0.0978;1.01}
                {0.607;0.101;1.01}
                {0.609;0.105;1.02}
                {0.611;0.11;1.03}
        }
}
```

```
DOTS
{
        COLOR=12;
        POINTS{
                {0.6;0.0897;1}
                {0.611;0.11;1.03}
        }
}
```

SLIDE – význam proměnné části snímku

FACE – znamená těleso definované množinou bodů, hran a polygonů

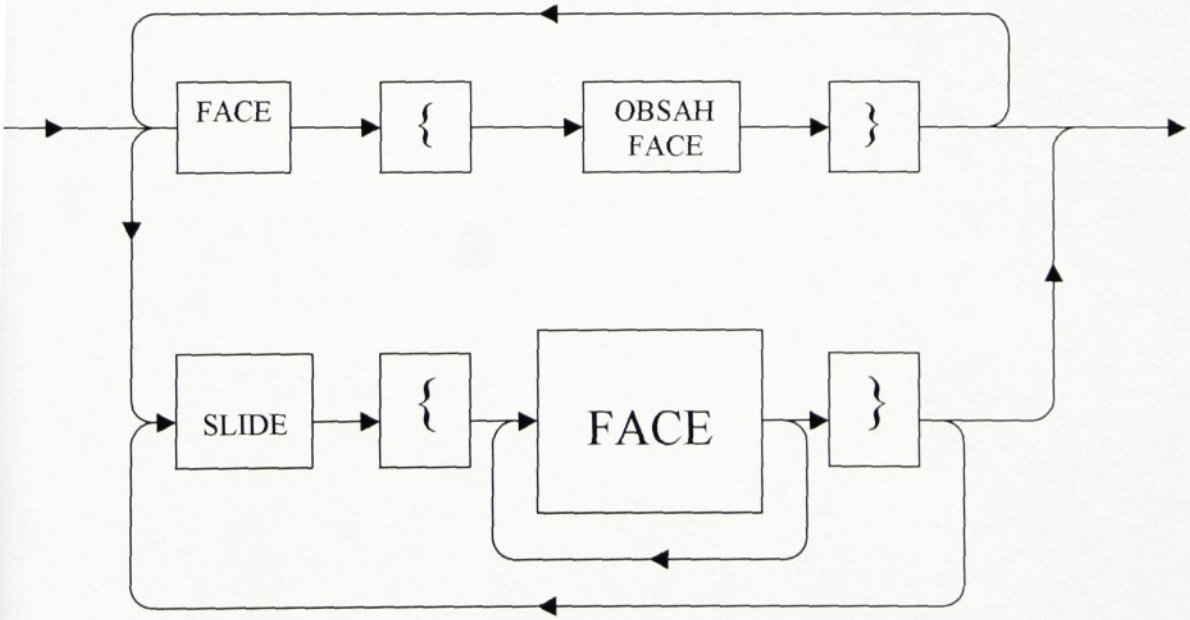
PATH – spojená čára definovaná body

DOTS – jednotlivé body

Načtení souboru

Nejprve je třeba ze souboru odstranit komentáře a prázdná místa. Komentář je libovolná část textu vymezená znaky „/*“, „*/“. Alternativní možností je použít „//“ pro označení jako komentář textu od aktuální pozice do konce řádku.

Potom následuje vlastní načtení, které je prováděno na principu stavového automatu. Část takového automatu, který je určen pro načtení souboru simulace je na *Obrázku č.3.4.2-1* .



Obr. č.3.4.2-1 Diagram stavového automatu pro načtení simulačního souboru

3.4.3. Výstupní soubor pro vykreslení grafů „*.csv“

Popis souboru

Začátek souboru může tvořit libovolný počet řádků začínajících středníkem. Tyto řádky jsou brány jako komentáře a program je ignoruje. Dalším řádkem je hlavička, která obsahuje názvy jednotlivých grafů a průběhů. Položky v hlavičce jsou odděleny čárkou. První položkou je název osy x. Dále následují jména grafů oddělená čárkou. Ke jménu grafu jsou připojena jména průběhů znakem „|“. Od následujícího řádku do konce souboru následují vlastní data. Každý řádek obsahuje jako první hodnotu společnou x – souřadnici a dále hodnoty příslušné jednotlivým průběhům definovaným v hlavičce oddělená čárkou. V případě, že ke jménu grafu nejsou připojena jména průběhů, pak se uvažuje, že má jeden průběh, jehož jméno je shodné se jménem grafu, odpovídá mu jeden sloupec dat. V opačném případě, je-li připojeno ke jménu grafu *n* jmen průběhů, grafu odpovídá *n* sloupců dat.

Příklad

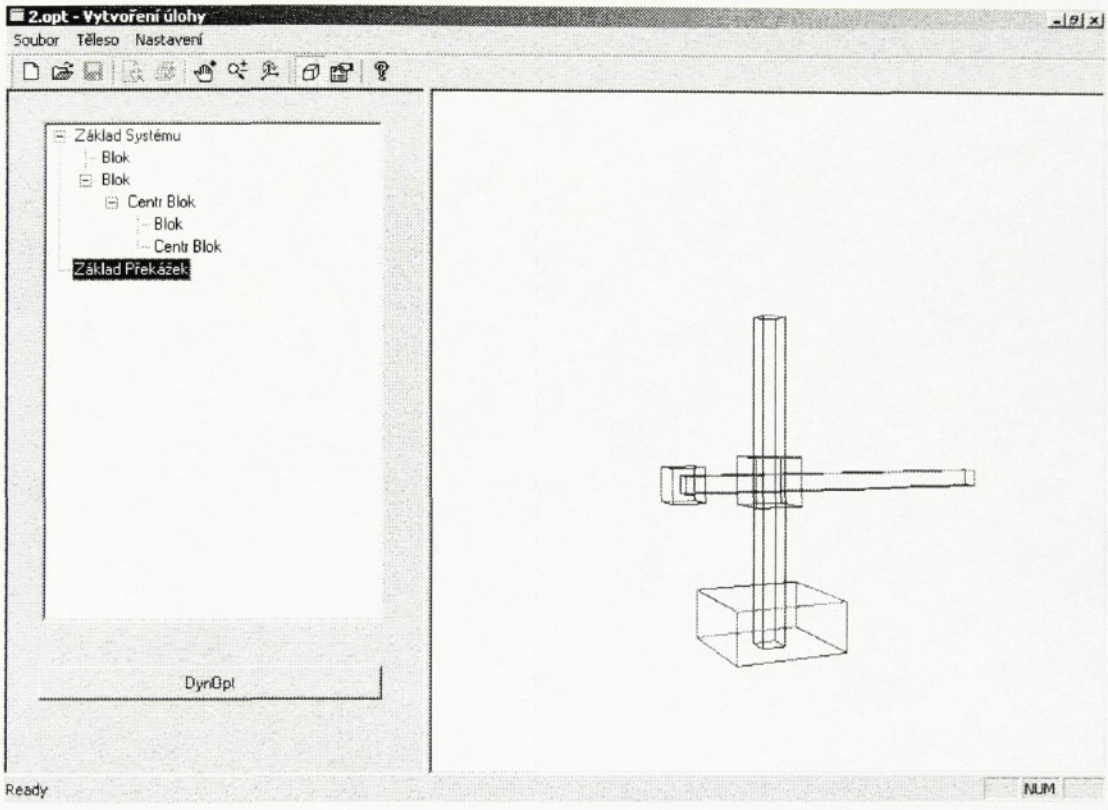
Následuje jednoduchý příklad výstupního souboru grafů :

T[s], Poloha | Y(0) | Y(1) , Rychlost | V(0) | V(1)
0, 0, 0, 0.5, 1
0.2, 0.1, 0.2, 0.5, 1
0.3, 0.15, 0.3, 0.5, 1
0.4, 0.2, 0.4, 0.5, 1
0.5, 0.25, 0.5, 0.5, 1

Z následujícího souboru budou vykresleny dva grafy se společnou x – souřadnicí, která bude **T[s].** Grafy budou mít názvy Poloha a Rychlost a v každém z nich budou vykresleny dva průběhy. V případě grafu polohy to bude **Y(0)**, **Y(1)** a v grafu rychlosti **V(0)** a **V(1)**.

3.5. Popis ovládání programu WinOpt

Konfigurační soubor je vytvořen v okamžiku, když se úloha uloží v okně návrhu. Jedná se o textový soubor, který má příponu „*.cfg“ a jsou v něm uložena veškeré vstupní informace pro výpočetní modul DynOpt. Toto okno je zobrazeno na *Obrázku č.3.5-1* .



Obr.č. 3.5.-1 Ukázka okna pro návrh úlohy

3.5.1. Vkládání těles

V levé části okna je stromová struktura, ve které se zadávají jednotlivá tělesa definující tvar robota a překážek. Toto vkládání je umožněno pouze tehdy, pokud je v parametrech úlohy vybrán typ robota obecný (bude dále podrobněji vysvětleno). Při vytváření nové úlohy je implicitně vybrán typ robota obecný. Tělesa je možno vkládat dvěma způsoby. V každém je třeba označit ve stromové struktuře rodiče daného tělesa. Potom je zde možnost vybrat těleso ze sedmi předdefinovaných elementárních těles. Popis těchto elementárních těles je uveden v kap. 2.2.6. Výběr tělesa je umožněn kliknutím pravým

tlačítkem na rodičovské těleso a následným výběrem z rozbalené nabídky a nebo v nabídce **Těleso** je to umožněno také.

Potom následuje zobrazení dialogu ve kterém je třeba zadat geometrické rozměry, pozici, rotaci a pohyb daného tělesa a pohyb celé větve. Tyto parametry jsou podrobněji vysvětleny v kap. 2.2.6. Těleso se zobrazí v pravé části a permanentně rotuje okolo z-souřadnice.

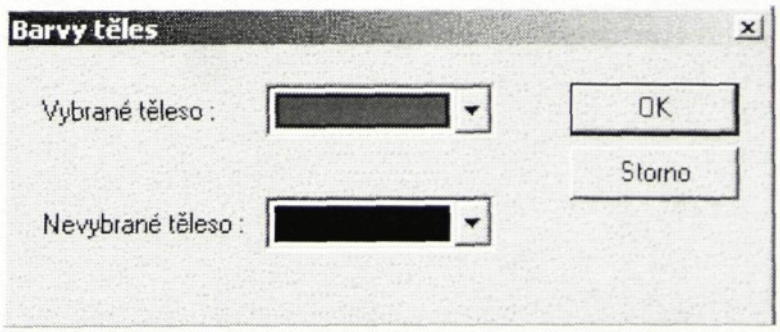
3.5.2. Zobrazení tvaru systému a překážek

K vykreslování úlohy dochází v pravé části okna. Tato část okna má grafické vlastnosti OpenGL. Typ matriálu a světla, které scénu osvětluje, není uživateli umožněno měnit. Je zde pouze možnost změny barvy materiálu, jak je popsáno v následující kapitole a nebo volba typu zobrazované scény.

Se zobrazovanou scénou je možné jakkoliv posunovat, rotovat a nebo měnit její měřítko. Tyto funkce jsou přístupné kliknutím pravého tlačítka myši na zobrazované scéně a následným výběrem příkazu z nabídky a nebo tlačítkem na panelu nástrojů. Typ zobrazované scény může být drátový a nebo vyplněný. Tato změna se provádí tlačítkem na panelu nástrojů.

3.5.3. Změna barvy zobrazovaných těles

Barva těles vybraných a nevybraných ve stromové struktuře je odlišná, ale je také možné, aby byla stejná. Pro lepší orientaci je však vhodné tuto barvu volit rozdílnou. Tyto barvy je možné měnit. V nabídce **Nastavení** klepněte na příkaz **Barvy** a zobrazí se dialog, který je na *Obrázku č.3.5.3-1* .



Obr. č.3.5.3-1 Dialog pro změnu barev těles

Vybrané barvy se ukládají pro každého uživatele zvlášť do registrů Windows.

3.5.4. Nastavení výpočetního modulu DynOpt

Pro nastavení je třeba zadat pouze cestu k výpočetnímu modulu DynOpt. Tato informace je opět ukládána do registrů Windows zvlášť pro každého uživatele. Zadání této cesty je potřeba udělat jako první, protože konfigurační soubor výpočetního modulu DynOpt musí ležet ve stejném adresáři. Program WinOpt toto dělá automaticky, ale je potřeba, aby věděl kde se program DynOpt nachází. Program DynOpt ukládá vygenerované výstupní soubory také do stejného adresáře, takže má-li program WinOpt má automaticky otevírat tyto výstupní soubory, musí vědět kde se nachází program DynOpt. Cestu k výpočetnímu modulu DynOpt je možné zadat v nabídce **Nastavení** příkazem **DynOpt**. Následuje otevírací dialog, ve kterém jste vyzváni k zadání cesty k programu DynOpt.

3.5.5. Ukládání úloh

Uložení se provádí stejně jako v každé aplikaci příkazem **Uložit** z nabídky **Soubor** a nebo na panelu nástrojů. Oproti jiným aplikacím je zde však jeden rozdíl. Při ukládání se uloží dva soubory, jeden bude mít příponu „*.opt“ (pokud není přípona napsána je připsána automaticky) a ještě dojde k vytvoření vstupního konfiguračního souboru pro výpočetní modul DynOpt.

Název konfiguračního souboru může mít maximálně 8 znaků a příponu „*.cfg“. Program WinOpt by omezení velikosti názvu na 8 znaků nemusel mít, ale pak by se musel název konfiguračního souboru zkracovat na 8 znaků a došlo by k rozdílnosti v názvech souborů od stejné úlohy. Jelikož by to bylo velmi nepraktické, bylo vybráno řešení omezením vstupního souboru „*.opt“ pro program WinOpt také na 8 znaků. Pokud tento název bude mít více než 8 znaků, bude uživatel upozorněn a požádán o nový název souboru.

3.5.6. Spuštění optimalizace

Optimalizace se spouští tlačítkem DynOpt na levém panelu. Před spuštěním výpočetního modulu DynOpt je však třeba vygenerovat vstupní konfigurační soubor. Tento soubor se generuje automaticky při uložení. Pokud úloha nebyla ještě uložena, je uživatel požádán dialogem pro uložení k zadání cesty a jména souboru. Konfigurační soubor má potom stejné jméno, pouze s jinou příponou. Pokud byla úloha pouze modifikována, k uložení dojde automaticky bez dotazování. Při ukončení optimalizace se zobrazí výsledné výstupní soubory.

3.6. Zobrazení grafů v programu WinOpt

V tomto okně se vykreslují časové průběhy, které jsou výsledkem výpočetního modulu DynOpt. Tyto data jsou předána uživatelskému prostředí WinOpt prostřednictvím výstupního souboru, který má příponu „*.csv".

Při otevření souboru správného formátu, se zobrazí jména jednotlivých grafů v ComboBoxu, který je umístěn v Panelu nástrojů okna. Zde je možné si vybrat požadovaný graf, který se při jeho výběru zobrazí. Dále je zde možné, aby si uživatel měnil dle své vlastní volby barvy jednotlivých průběhů. Toto je možné provést kliknutím na tlačítko v Panelu nástrojů a nebo v nabídce **Graf** klepněte na příkaz **Barvy**. Nastavení barev se ukládá v registrech Windows pro každého uživatele a je při každé spuštění programu obnoveno.

Při úspěšném ukončení výpočtu optimalizačního modulu je tento výstupní soubor otevřen automaticky (pokud je správně zadaná cesta k výpočetnímu modulu DynOpt) .

3.7. Simulace v programu WinOpt

Toto okno programu WinOpt slouží pro simulaci výsledné 3D scény. Scéna je zobrazována ze souboru, který je výstupem z výpočetního modulu DynOpt. Tento soubor má příponu „*.scn". Jeho podrobnější popis je v kap. 3.4.2.

Možnosti v ovládaní Simulace v programu WinOpt je uvedeno v následující tabulce.

náhled	zobrazí náhled před tiskem
tisk	vytiskne aktuální scénu
posun	posune scénu
rotace	rotuje scénu
měřítko	mění měřítko zobrazované scény
krok a nebo klávesa mezerník	provede následný krok simulace scény
drátový model	provede zobrazení scény drátovým modelem

Tab č.3.7-1 Ovládání Simulace v programu WinOpt

Při otevření souboru se zobrazí první položka animace.

3.8. Průvodce vytvořením úlohy

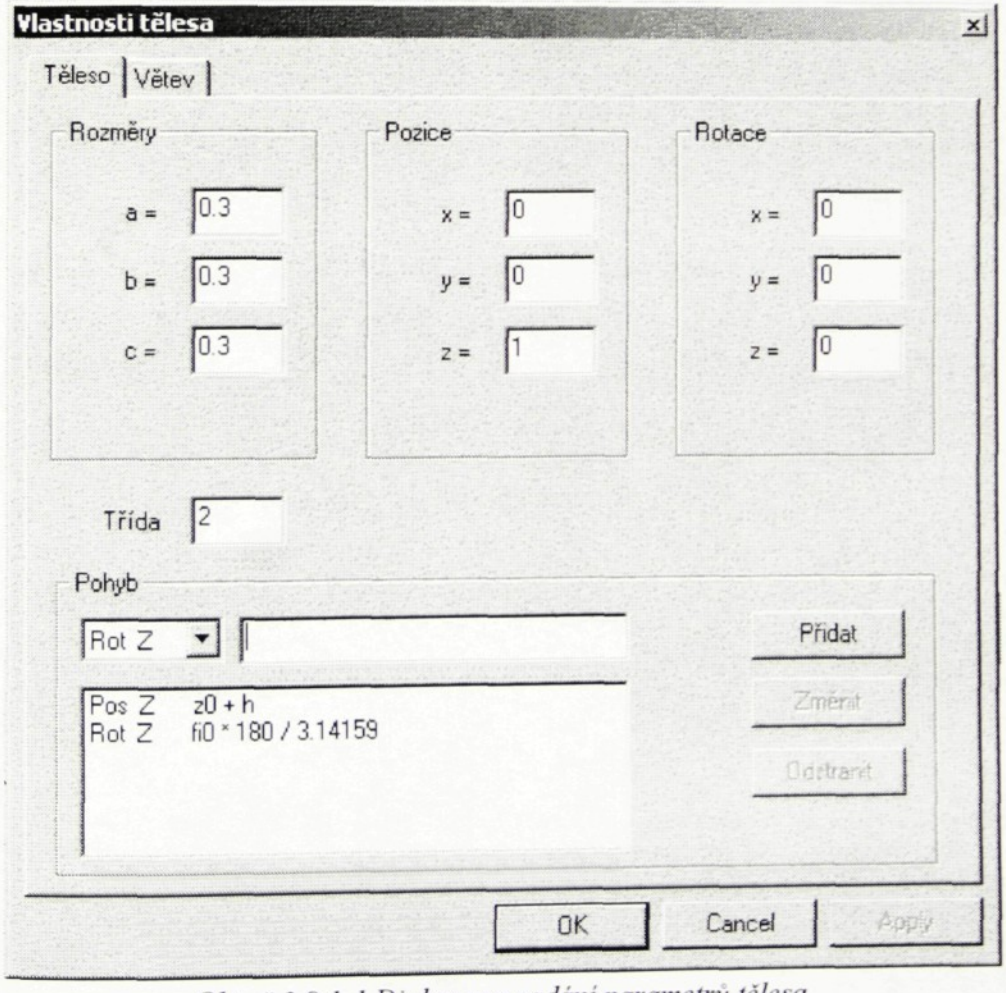
3.8.1. Zadání tvaru a pohybu systému

Tvar systému je možno zadat libovolný a nebo je zde možnost vybrat si ze tří předdefinovaných systémů robotů. Jedná se o základní typy :

- cylindrický
- sférický
- angulární

Podrobný popis těchto systémů robotů je v literatuře [1] .

Způsob zadávaní těchto tvarů je popsáno v kap. 3.5.1, ale zde se budeme zabývat popisem parametrů těchto těles. Při zadání nového tělesa se zobrazí dialog, který je na *Obrázku č.3.8.1-1* . Tento dialog se v některých detailech liší pro různá tělesa.



Obr. č.3.8.1-1 Dialog pro zadání parametrů tělesa

Geometrické parametry se liší pro různá tělesa a slouží pro detekci kolize a pro simulaci výsledné 3D scény.

Pozice a rotace

Zde je možno nastavit počáteční pozici a rotaci pro každé těleso. V případě, že se s tělesem bude následně pohybovat pomocí pohybových rovnic, tak tyto parametry pozice a rotace budou těmito rovnicemi zrušeny.

Třída

Postup při volbě třídy tělesa je podrobně popsán v kap. 2.2.8.

Pohyb

Tato část se vyplňuje pouze tehdy, pokud chceme s tělesem pohybovat. Pohybovat můžeme jak s jednotlivými tělesy, tak i s celými větvemi, které obsahují tělesa. S tělesy je možno posunovat i rotovat. V případě rotace jsou použity jako jednotky stupně. Z vnitřních proměnných mohou pohybové rovnice obsahovat pouze tyto : *SY(j), R(i, X), R(i, X), R(i, X),* a *ST* a nemohou obsahovat funkci *Idt*.

3.8.2. Zadání parametrů pro výpočet

Zobrazení dialogu pro zadání vstupních parametrů se vyvolá příkazem **Parametry** z nabídky **Nastavení** a nebo na panely nástrojů tlačítkem **Parametry**.

Záložka Spline a Opt Alg

V těchto záložkách jsou obsaženy parametry týkající se konstrukce spline, typu optimalizačních algoritmů a okrajové podmínky. Při kliknutí na daný parametr se ve spodní části dialogu zobrazí popis parametru.

Záložka Systém

Je zde možnost vybrat si již předdefinovaný typ robota a pak je možné měnit pouze tyto parametry, popř. definiční rovnice dynamického systému:

Cylindrický: m3, m4, m5, m6, l4, l5, l6, J2, J3, J4, e, h, km (seznam 3 reálných hodnot) , im (seznam 3 reálných hodnot)

Sférický: m1, m2, m3, m4, m5, l1, l2, l3, l4, l5, e, h, km (seznam 3 reálných hodnot), im (seznam 3 reálných hodnot)

Angulární: m1, m2, m3, m4, m5, l1, l2, l3, l4, l5, h, alfa, km (seznam 3 reálných hodnot), im (seznam 3 reálných hodnot)

Všechny parametry jsou reálného typu.

Nebo je možnost vybrat systém obecný. Potom je možné zadat soustavu diferenciálních rovnic definující dynamický systém. Definice se skládá z přiřazení hodnot nebo výrazů proměnným:

$N

- řád dynamického systému, konstanta, povinné

$M

- počet stupňů volnosti, konstanta, povinné

$S

- počet transformačních rovnic, konstanta, nepovinné (standardně 0)

$SDISP

- počet transformačních rovnic, které se budou zapisovat/zobrazovat, konstanta, nepovinné (standardně nastaveno na hodnotu $S)

$U(j)

- řízení, přiřazení výrazů obsahujících libovolné proměnné v *Tab.č.2.2.7-1* kromě *$POS(i,X), $POS(i,Y), $POS(i,Z), $ROT(i,X), $ROT(i,Y), $ROT(i,Z),*
- povinné, $j=0, \ldots(SM-1)$.

$S(X), $S(Y), $S(Z)

- v případě pohybu systému v prostoru s překážkami je nutno definovat funkci S(t), pomocí které jsou určeny body na trajektorii, ve kterých se bude provádět detekce kolize.

Indexy proměnných *$X(i,j), $Y(j), $V(j) a $A(j)* mohou nabývat hodnot $i=0,\ldots\ \$N$, $j=0,\ldots(\$M-1)$.

Záložka Kritérium

V případě kvadratického kritéria *R.3.8.2-1* jsou vybrány standardně koeficienty $k_1 \ldots k_n$ rovny jedné. Jejich změna je umožněna zápisem do textového pole oddělené středníkem „;“.

V přídě obecné kriteriální funkce je třeba do textového pole zadat integrand a terminální člen.

R. 3.8.2-1 $$J = \int_{t=0}^{t_f} (k_1.u_1^2 + k_1.u_1^2 + \ldots\ldots + k_n.u_n^2).dt$$

$F

- integrand funkcionálu
- výraz obsahující libovolné proměnné v *Tabulce č.2.2.7-1* kromě *$POS(i,X), $POS(i,Y), $POS(i,Z), $ROT(i,X), $ROT(i,Y), $ROT(i,Z)*
- povinné

$FT, $F0

- terminální, resp. počáteční člen
- výraz obsahující libovolné proměnné v *Tabulce č.2.2.7-1* kromě *$POS(i,X), $POS(i,Y), $POS(i,Z), $ROT(i,X), $ROT(i,Y), $ROT(i,Z)*
- nepovinné (standardně rovno 0)

Příklad: Obecné kritérium optimality

R.3.8.2-2
$$J = \int_{t=0}^{t_f} (k_1 \cdot |u_1| + k_2 \cdot |u_2| + k_3 \cdot |u_3|) \cdot dt$$

```
$F=10*abs($u(0)) + 20* abs($u(0)) + 20* abs($u(0))
```

Záložka Omezení

Omezení se zadává v podobě nerovností. Pro vyjádření nerovností jsou k dispozici relační funkce (*GT, GE, LE, LT, EQ*), které lze libovolně kombinovat pomocí logických funkcí (*AND, OR, XOR, NOT*) nebo aritmetických operací *, +.

V textovém poli konstanty je možné si předdefinovat konstanty pro zpřehlednění a v poli podmínka pak musí být funkce, která vrací hodnotu větší než nula v případě, že momentální hodnoty vnitřních proměnných leží uvnitř přípustné oblasti řešení a v opačném případě nulu.

Příklad: Omezení rozsahu polohy jednotlivých členů dané rovnicemi *R.3.8.2-3* .

R.3.8.2-3
$$-\frac{2}{10}\pi \le \vartheta \le \frac{4}{10}\pi$$
$$-\frac{2}{10}\pi \le \psi \le \frac{4}{10}\pi$$

Konstanty:

```
th=$Y(1);
psi=$Y(2);

C1=ge(th,-2*PI/10);
C2=le(th,4*PI/10);
C3=ge(psi,-2*PI/10);
C4=le(psi,4*PI/10);
```

Podmínka:

$CONDITION=C1*C2*C3*C4;

Klíčové slovo *$CONDITION* je připsáno programem WinOpt.

Úlohy s omezením lze též řešit přibližně pomocí modifikace funkcionálu přičtením speciálních funkcí,jejichž hodnota je nulová (popř. v absolutní hodnotě nízká) uvnitř dovolené oblasti řešení a velmi narůstá mino tuto oblast. Lze použít tzv. bariérové funkce, jejichž hodnota na hranici omezení narůstá do nekonečna nebo tzv. pokutové funkce, které pouze narůstají s mírou překročení daného omezení. Přísnost požadavku na nepřekročení dovolené oblasti řešení pak lze měnit násobením pokutové funkce vhodným koeficientem. Využití pokutových funkcí může být výhodnější, neboť počáteční odhad řešení v tom případě může ležet i uvnitř zakázané oblasti.

Následující příklad ukazuje pokutovou funkci, která zavádí omezení na řídící veličině. Požadavek je, aby řídící veličina ležela v dané oblasti $U_d \leq u(t) \leq U_h$ .Pak pokutová funkce pro toto omezení vypadá následně :

R.3.8.2-4 $$P_u(t) = K_p \,.\, [(U_h - u(t)).(U_d - u(t)) + |(U_h - u(t)).(U_d - u(t))|]$$

kde $\boldsymbol{K_p}$ je pokutový koeficient

Výsledný minimalizovaný funkcionál pak má tvar :

R.3.8.2-5 $$J_p = \int_0^T [f_0(y^{(n)}, \ldots\ldots, \dot{y}, y, u, t) + P_u(t)]dt$$

3.9. Instalace

Program WinOpt byl vytvořen pro operační systém Windows 95/98/Milenium/NT/2000. Pro spuštění je potřeba zkopírovat následné dynamické knihovny do systému. Aplikace vytvořené v Microsoft Visual C++ vyžadují následující dynamické knihovny:

- mfc42.dll
- msvcrt.dll
- msvcirt.dll

Z důvodů použití OpenGL jsou třeba ještě následující dynamické knihovny:

- opengl32.dll
- glu32.dll
- glut32SGI.dll

Tyto dynamické knihovny je třeba překopírovat do následujících systémových adresářů dle typu operačního systému.

Windows 95/98/Milenium :

\Windows\system\

Windows NT/2000:

\Winnt\system32\

4. Závěr

V praktické části této práce bylo vytvořeno uživatelské prostředí, které umožňuje předávat parametry a zobrazovat výsledky výpočtu z výpočetního modulu DynOpt. S jeho pomocí je možno snadno a rychle vytvářet úlohy a provádět vizualizaci optimálních trajektorií robotů. Tento program názorně ukazuje výhody grafického operačního systému Microsoft Windows.

K značnému usnadnění dochází při vytváření úlohy, kde se zadává obecný tvar robota, protože uživateli je ihned zobrazován geometrický tvar a nemusí čekat, až na výslednou 3D simulaci po výpočtu. Dalším usnadněním je eliminace syntaktických chyb v konfiguračním souboru. Odpadají zde problémy, jako je např. zapomenutí ukončovací složené závorky, což je jedna z nejčastějších chyb apod.

Program neumožňuje otevření více úloh najednou, ale Windows jsou preemptivní multitaskingový operační systém, takže lze spustit několik procesů najednou. Uživatel si tedy může spustit několik programů WinOpt a v každém si otevřít jinou úlohu.

Program DynOpt je určen zatím pouze pro experimentální účely, takže program WinOpt také. Úkolem těchto experimentů je nalézt vhodnou volbu některých parametrů pro optimalizaci. Autor se domnívá, že uživatelské rozhraní WinOpt zjednoduší práci a zkrátí čas při hledání optimálních parametrů výpočetního modulu DynOpt a přispěje tak k jeho využití v praxi.

5. Literatura

[1] CVEJN, J.:Optimální řízení robotů v prostoru s překážkami
Diplomová práce, VŠST Liberec , 1996.

[2] CVEJN, J.:Metoda optimálního řízení dynamických systémů
Disertační práce, TU Liberec , 2000.

[3] LUBOJACKÝ, O. a kol..:Základy robotiky . Učební text, VŠST Liberec , 1990.

[4] PETZOLD, Ch .: Programování ve Windows. Computer Press, 1999.

[5] Elektronická dokumentace k Microsoft Visual C++ 5.0 / 6.0

[6] KRUGLINSKI, D.: Mistrovství ve Visual C++. Computer Press, 1999.

[7] RICHTER, J.: Windows pro pokročilé a experty. Compruter Press, 1997.

[8] HEROUT, P.: ABC programátora v jazyce C. KOPP, České Budějovice 1992

6. Příloha

V příloze je uvedena pouze část zdrojového kódu aplikace z důvodu velké obsáhlosti. Pro uvedení kompletního zdrojového kódu by bylo třeba okolo 200 stran. Vybrána byla třída **COptDoc**, která je společná pro celou aplikaci.

OptDoc.h

```
#if !defined(AFX_OPTDOC_H__167475C0_7950_46BC_9F44_897D06620A38__INCLUDED_)
#define AFX_OPTDOC_H__167475C0_7950_46BC_9F44_897D06620A38__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <fstream.h>
#include <afxtempl.h>
#include <strstrea.h>
//#include "CreateView.h"

#define MAXGRAPHS 100
#define MAXBUFF 1000

#include "afxtempl.h"
#include "CfgSheet.h"

//struktura pro uložení pozice a rotace tělesa
struct CPosition {
        float xPos;
        float yPos;
        float zPos;
        float xRot;
        float yRot;
        float zRot;

void Serialize (CArchive& ar)
{
        if (ar.IsStoring())
                ar << xPos << yPos << zPos << xRot << yRot << zRot;
        else
                ar >> xPos >> yPos >> zPos >> xRot >> yRot >> zRot;
}

};

struct CComponent {
        //struktura pro popis parametrů tělesa
        HTREEITEM hItem;                //handle v objektu CTreeCtrl
        int iParent;                    //pozice v poli rodiče
        UINT figure;                    //typ tělesa
        float s0;                       //geometrické rozměry tělesa
        float s1;
        float s2;
        float s3;
        CPosition pos;                  //poloha tělesa
        CPosition posBranch;            //poloha podřízené větve
        CString motion;                 //pohyb tělesa
        CString motionBranch;           //pohyb podřízené větve



/*
```

```
                    s0      s1      s2      s3
        Block       a       b       c
        CenterBlock a       b       c
        Prism       r       n       v
        Rightprism  a       b       v
        Spire       r       n       v
        TruncSpire  r       n       v0      v
        RightSpire  a       b       v
*/

};

class CCreateView;

class COptDoc : public CDocument
{
protected: // create from serialization only
        COptDoc();
        DECLARE_DYNCREATE(COptDoc)

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(COptDoc)
        public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
        virtual BOOL OnOpenDocument(LPCTSTR lpszPathName);
        virtual BOOL OnCmdMsg(UINT nID, int nCode, void* pExtra, AFX_CMDHANDLERINFO*
        pHandlerInfo);
        protected:
        virtual BOOL SaveModified();
        //}}AFX_VIRTUAL

// Implementation
public:
        void SaveDocument();
        CString m_strMotionBranchRoot;
        void OpenCsvScn();
        CCfgSheet m_cfgSheet;
        HTREEITEM m_hSelItem;
        CString m_strFilePath;
        BOOL m_bLines;
        ifstream m_ifstream;
        float m_fSelCompColor[4], m_fCompColor[4];
        int m_iPoints, m_iTraces;
        strstream m_streamHeader;
        void TreeStore(strstream& stream, BOOL bFigure);
        CString m_strFigure[7], m_strmFigure[7];
        CPosition GetPosition(int pos);
        int GetArrayPos(HTREEITEM hItem);
        HTREEITEM m_hRobot,m_hRestriction;
        int m_CountOfComp;
        CArray <CComponent, CComponent&> m_compArray;
        float (*m_data)[MAXGRAPHS];
        virtual ~COptDoc();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
```

```
protected:

// Generated message map functions
protected:
	//{{AFX_MSG(COptDoc)
	afx_msg void OnParametersCFG();
	afx_msg void OnFileOpenCsv();
	afx_msg void OnFileOpenScn();
	afx_msg void OnUpdateFileSave(CCmdUI* pCmdUI);
	afx_msg void OnSettingsColors();
	afx_msg void OnSettingsDynopt();
	afx_msg void OnSaveProject();
	afx_msg void OnUpdateSaveProject(CCmdUI* pCmdUI);
	afx_msg void OnSaveProjectAs();
	afx_msg void OnAppExit();
	afx_msg void OnUpdateMove(CCmdUI* pCmdUI);
	afx_msg void OnUpdateRotate(CCmdUI* pCmdUI);
	afx_msg void OnUpdateScale(CCmdUI* pCmdUI);
	afx_msg void OnLines();
	afx_msg void OnUpdateLines(CCmdUI* pCmdUI);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
private:
	BOOL m_bSaveAs;
	void GetRGB(int selCompColor, int compColor);
	void DrawGraphs(CString filePath);
	int m_iPos;
	POSITION m_pospos;
	void WriteMotion(int index, int arrayPos, BOOL bBranch);
	strstream m_strcfg, m_strscn;
	void WriteBlockRot(int pos, strstream& stream);
	void WriteBlockPos(int pos, strstream& stream);
	void WriteBlock(int pos, strstream& stream, BOOL bFigure);
	void WriteTabs(int count, strstream& stream);
	void WritePos(int iParent, int tabs, strstream& stream, BOOL bBranch, BOOL bFigure);
	int GetLengthOfBuff();
	void GetLine(int j);
	void GetData();
	void GetSize();
	void SkipCommentsCSV();
	char m_buff[1000];
	void CreateCFG();

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_OPTDOC_H__167475C0_7950_46BC_9F44_897D06620A38__INCLUDED_)
```

OptDoc.cpp

```
#include "stdafx.h"
#include "Opt.h"

#include "OptDoc.h"
#include "OptView.h"
#include "SimView.h"
#include "CompColorDlg.h"
#include "ColorComboEx.h"
#include "CreateView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#include "CfgSheet.h"

extern CComboBox g_comboSelGraph;

void AFXAPI SerializeElements (CArchive& ar, CComponent* pNewComp, int nCount)
{
   //slouží pro ukládání struktury CComponent
   for ( int i = 0; i < nCount; i++, pNewComp++ )
   {
      if (ar.IsStoring()) {
                        ar << pNewComp->figure << pNewComp->iParent << pNewComp->level;
                        ar << pNewComp->motion << pNewComp->s0 << pNewComp->s1;
                        ar << pNewComp->s2 << pNewComp->s3 << pNewComp->motionBranch;
                        ar << pNewComp->motion;
                }
                else {
                        ar >> pNewComp->figure >> pNewComp->iParent >> pNewComp->level;
                        ar >> pNewComp->motion >> pNewComp->s0 >> pNewComp->s1;
                        ar >> pNewComp->s2 >> pNewComp->s3 >> pNewComp->motionBranch;
                        ar >> pNewComp->motion;
                }
                pNewComp->pos.Serialize(ar);
                pNewComp->posBranch.Serialize(ar);
   }
}

/////////////////////////////////////////////////////////////////////////////
// COptDoc

IMPLEMENT_DYNCREATE(COptDoc, CDocument)

BEGIN_MESSAGE_MAP(COptDoc, CDocument)
        //{{AFX_MSG_MAP(COptDoc)
        ON_COMMAND(ID_NASTAVEN_PARAMETRY, OnParametersCFG)
        ON_COMMAND(ID_FILE_OPEN_CSV, OnFileOpenCsv)
        ON_COMMAND(ID_FILE_OPEN_SCN, OnFileOpenScn)
        ON_UPDATE_COMMAND_UI(ID_FILE_SAVE, OnUpdateFileSave)
        ON_COMMAND(ID_SETTINGS_COLORS, OnSettingsColors)
        ON_COMMAND(ID_SETTINGS_DYNOPT, OnSettingsDynopt)
        ON_COMMAND(ID_FILE_SAVE_OPT, OnSaveProject)
        ON_UPDATE_COMMAND_UI(ID_FILE_SAVE_OPT, OnUpdateSaveProject)
        ON_COMMAND(ID_FILE_SAVE_OPT_AS, OnSaveProjectAs)
```

```
        ON_COMMAND(ID_APP_EXIT, OnAppExit)
        ON_UPDATE_COMMAND_UI(ID_B_MOVE, OnUpdateMove)
        ON_UPDATE_COMMAND_UI(ID_B_ROTATE, OnUpdateRotate)
        ON_UPDATE_COMMAND_UI(ID_B_SCALE, OnUpdateScale)
        ON_COMMAND(ID_LINES, OnLines)
        ON_UPDATE_COMMAND_UI(ID_LINES, OnUpdateLines)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// COptDoc construction/destruction
COptDoc* g_pDoc;

COptDoc::COptDoc() : m_cfgSheet("Parametry úlohy")
{
        //inicializace proměnných této třídy
        m_strFigure[0] = "BLOCK";
        m_strFigure[1] = "CENTERBLOCK";
        m_strFigure[2] = "PRISM";
        m_strFigure[3] = "RIGHTPRISM";
        m_strFigure[4] = "SPIRE";
        m_strFigure[5] = "TRUNCSPIRE";
        m_strFigure[6] = "RIGHTSPIRE";

        m_strmFigure[0] = "Blok";
        m_strmFigure[1] = "Centr Blok";
        m_strmFigure[2] = "Hranol";
        m_strmFigure[3] = "Kolmý Hranol";
        m_strmFigure[4] = "Jehlan";
        m_strmFigure[5] = "Komolý Jehlan";
        m_strmFigure[6] = "Kolmý Jehlan";
        m_bLines = FALSE;
        g_pDoc = this;
        m_hSelItem = NULL;

        m_fSelCompColor[4] = 1.0f;
        m_fCompColor[4] = 1.0f;
        m_bSaveAs = FALSE;
}

COptDoc::~COptDoc()
{
        delete[] m_data;
}


/////////////////////////////////////////////////////////////////////////////
// COptDoc serialization

void COptDoc::Serialize(CArchive& ar)
{
        //zde se ukládají data do souboru „*.opt“
        CString str;
        if (ar.IsStoring())
        {
                ar << m_CountOfComp << m_strMotionBranchRoot;
        }
        else
        {
                OnNewDocument();
```

```
            ar >> m_CountOfComp >> m_strMotionBranchRoot;
        }
        m_compArray.Serialize(ar);                          //uložení struktury CComponent
        m_cfgSheet.Serialize(ar);                           //uložení parametrů úlohy
}

/////////////////////////////////////////////////////////////////////////////
// COptDoc diagnostics

#ifdef _DEBUG
void COptDoc::AssertValid() const
{
        CDocument::AssertValid();
}

void COptDoc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// COptDoc commands

BOOL COptDoc::OnNewDocument()
{
        //nastavení všech proměnných do počátečních hodnot před vytvořením nové úlohy
        if (!CDocument::OnNewDocument())
                return FALSE;
        m_strFilePath.Empty();
        m_iPoints = 0;
        m_CountOfComp = 0;
        m_compArray.SetSize(30);
        m_compArray.RemoveAll();
        m_strMotionBranchRoot.Empty();

        m_cfgSheet.SetDefaultData();

        g_comboSelGraph.ResetContent();

        POSITION pos = GetFirstViewPosition();
        GetNextView(pos);
        if (pos != NULL) {
                GetNextView(pos)->OnInitialUpdate();                    //CSimView
                GetNextView(pos)->OnInitialUpdate();                    //CCreateView
        }

        GetRGB(AfxGetApp()->GetProfileInt("Opt","SelBlockColor",3),
                        AfxGetApp()->GetProfileInt("Opt","BlockColor",5));

        return TRUE;
}
```

```
BOOL COptDoc::OnOpenDocument(LPCTSTR lpszPathName)
{
	if (!CDocument::OnOpenDocument(lpszPathName))
		return FALSE;


	//vykresleni kinematickych velicin
	CString name = lpszPathName;

	CString pathName = lpszPathName;
	pathName.Delete(pathName.GetLength() - 4, 4);
	pathName += ".csv";
	m_iPoints = 0;


	DrawGraphs(pathName);                                          //vykreslení grafů

	//simulace pohybu robota
	pathName.Delete(pathName.GetLength() - 4, 4);
	pathName += ".scn";
	m_ifstream.open(pathName,ios::nocreate,filebuf::sh_none);

	UpdateAllViews(NULL);
	m_ifstream.close();
	m_ifstream.clear();

	return TRUE;
}

void COptDoc::OnFileOpenCsv()
{
	//otevření pouze souboru grafů „*.csv“ a jejich vykreslení
	m_iPoints = 0;
	g_comboSelGraph.ResetContent();
	CString path;

	//v dialogu je možné otevřít pouze grafy a nebo celou úlohu „*.opt“
	CFileDialog dlg(1,"*.csv",NULL,OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
				"Graf Soubory (*.csv)|*.csv|Opt Soubory (*.opt)|*.opt||");
	dlg.DoModal();
	path = dlg.GetPathName();
	CString str = dlg.GetFileExt();
	str.MakeLower();
	if (str == "csv")
		DrawGraphs(path);

	if (str == "opt")
		OnOpenDocument(path);

	POSITION pos = GetFirstViewPosition();
	GetNextView(pos)->OnInitialUpdate();
}

void COptDoc::OnFileOpenScn()
{
	//otevření pouze souboru pro simulaci „*.scn“
	CString path;
	//v dialogu je možné otevřít pouze simulaci a nebo celou úlohu „*.opt“
	CFileDialog dlg(1,"*.scn",NULL,OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
				"Sim Soubory (*.scn)|*.scn|Opt Soubory (*.opt)|*.opt||");
```

```
	dlg.DoModal();
	path = dlg.GetPathName();
	CString str = dlg.GetFileExt();
	str.MakeLower();
	if (str == "scn") {
		m_ifstream.open(path,ios::nocreate,filebuf::sh_none);
		if (m_ifstream) {
			POSITION pos = GetFirstViewPosition();
			GetNextView(pos);
			((CSimView*) GetNextView(pos))->ReadFile();
		}
	}
	if (str == "opt")
		OnOpenDocument(path);

	m_ifstream.close();
	m_ifstream.clear();
}

void COptDoc::OpenCsvScn()
{
	//tato metoda je volána po skončení výpočtu optimalizace a zobrazí výsledky výpočtu
	m_iPoints = 0;
	g_comboSelGraph.ResetContent();
	CString path;

	path = GetPathName();
	path.Delete(path.GetLength() - 4, 4);
	path += ".csv";

	DrawGraphs(path);

	POSITION pos = GetFirstViewPosition();
	GetNextView(pos)->OnInitialUpdate();

	path.Delete(path.GetLength() - 4, 4);
	path += ".scn";

	m_ifstream.clear();
	m_ifstream.open(path,ios::nocreate,filebuf::sh_none);
	if (m_ifstream) {
		POSITION pos = GetFirstViewPosition();
		GetNextView(pos);
		((CSimView*) GetNextView(pos))->ReadFile();
	}

	m_ifstream.close();
	m_ifstream.clear();
}

void COptDoc::SkipCommentsCSV()
{
	//přeskakuje komentáře v souboru grafů
	m_buff[0] = ';';
	while ((m_buff[0] == ';' || m_buff[0] == '\0') && !m_ifstream.eof())
		m_ifstream.getline(m_buff,MAXBUFF);
}
```

```
void COptDoc::GetSize()
{
	//získá velikost pole dat ze souboru grafů
	m_iPoints = 0;
	m_iTraces = 0;
	SkipCommentsCSV();
	strstream stream;
	m_ifstream.getline(m_buff,MAXBUFF);
	stream.write(m_buff,GetLengthOfBuff());
	//počet grafu
	while (!stream.eof()) {
		stream.getline(m_buff,MAXBUFF,',');
		m_iTraces++;
	}
	//počet bodu v grafech
	while (!m_ifstream.eof()) {
		m_ifstream.getline(m_buff,MAXBUFF);
		m_iPoints++;
	}
	m_ifstream.close();
	m_ifstream.clear();
}



void COptDoc::GetData()
{
	//načítá data ze souboru grafů do proměnné m_data
	m_data = new float[m_iPoints][100];
	SkipCommentsCSV();
	m_streamHeader.write(m_buff,GetLengthOfBuff());
	for (int j = 0; j < m_iPoints; j++) {
		m_ifstream.getline(m_buff,MAXBUFF);
		GetLine(j);
	}
}

void COptDoc::GetLine(int j)
{
	//načte řádek ze souboru grafů do proměnné m_data
	int i;
	strstream stream;
	stream.write(m_buff,GetLengthOfBuff());
	for (i = 0; i < m_iTraces; i++) {
		stream.getline(m_buff,MAXBUFF,',');
		m_data[i][j] = (float) atof(m_buff);
	}
}

int COptDoc::GetLengthOfBuff()
{
	//vrací velikost dat v bufferu
	for (int i = 0; i < MAXBUFF; i++)
		if (m_buff[i] == '\0')
			break;
	return i;
}
```

```
int COptDoc::GetArrayPos(HTREEITEM hItem)
{
	//získá fyzickou pozici položky ze stromu těles v poli
	if (hItem == m_hRobot)
		return -1;
	if (hItem == m_hRestriction)
		return -2;
	for (int i = 0; i < m_CountOfComp; i++)
		if (hItem == m_compArray[i].hItem)
			return i;
	return -3;
}

CPosition COptDoc::GetPosition(int pos)
{
	//získá absolutní polohu tělesa z relativních poloh rodičovských těles
	CPosition pos3D;
	::ZeroMemory(&pos3D, sizeof(CPosition));
	while (pos > -1) {
		pos3D.xPos += m_compArray[pos].pos.xPos;
		pos3D.yPos += m_compArray[pos].pos.yPos;
		pos3D.zPos += m_compArray[pos].pos.zPos;
		pos3D.xRot += m_compArray[pos].pos.xRot;
		pos3D.yRot += m_compArray[pos].pos.yRot;
		pos3D.zRot += m_compArray[pos].pos.zRot;
		pos = m_compArray[pos].iParent;
	}
	return pos3D;
}

void COptDoc::CreateCFG()
{
	//vytvoří konfigurační soubor „*.cfg“
	CString filePath;
	ofstream cfgstr;
	strstream stream, strobs;

	//získá cestu a jméno pro uložení konfiguračního souboru
	filePath = GetPathName();
	filePath.Delete(filePath.GetLength() - 4, 4);
	filePath += ".cfg";

	cfgstr.open(filePath);

	//v každé Property Page je metoda GetData, která vytvoří část konfiguračního souboru z dat na této stránce
	//ITERATOR
	cfgstr << "ITERATOR" << '\n' << '{' << '\n';
	//spline
	m_cfgSheet.m_splinePage.GetData(cfgstr);
	//optmethod
	m_cfgSheet.m_optPage.GetData(cfgstr);

	//EQUATIONS
	m_cfgSheet.m_constPage.GetData(cfgstr);
```

```
//FIGURE & MOTION
if (!m_cfgSheet.m_constPage.m_iType) {
        //jen jestliže je systém obecný
        TreeStore(stream, TRUE);
        cfgstr.write(stream.str(), stream.pcount());
        cfgstr << '}' << '\n' << '\n';
}

//CRITERIA
m_cfgSheet.m_critPage.GetData(cfgstr);

//SPACE
//vytvoří stromovou strukturu z překážek
TreeStore(strobs, FALSE);
if (strobs.pcount() > 0 || !m_cfgSheet.m_spacePage.m_sCond.IsEmpty()) {
        cfgstr << "SPACE=REAL" << '\n' << '{' << '\n';
        if (strobs.pcount() > 0) {
                //OBSTACLES
                cfgstr << '\t' << "OBSTACLES" << '\n' << '\t' << '{' << '\n';
                cfgstr.write(strobs.str(), strobs.pcount());
                cfgstr << '\t' << '}' << '\n';
        }
        if (!m_cfgSheet.m_spacePage.m_sCond.IsEmpty()) {
                //CONSTRAINTS
                m_cfgSheet.m_spacePage.GetData(cfgstr);
        }
        cfgstr << '}' << '\n';
}
else
        cfgstr << "SPACE=NULL;" << '\n';


oid COptDoc::TreeStore(strstream& stream, BOOL bFigure)

//zapíše do streamu položku FIGURE nebo OBSTACLES dle bFigure
int i = 0;
int tabs = 2;
if (bFigure)
        stream << '\t' << "FIGURE" << '\n' << '\t' << '{' << '\n';

m_iPos = 1;
m_strcfg << '\t' << "MOTION" << '\n' << '\t' << '{' << '\n';

if (!m_strMotionBranchRoot.IsEmpty())
        WriteMotion(0, -1, TRUE);

if (bFigure)
        WritePos(-1, tabs, stream, 0, bFigure);                    //pro robota
else
        WritePos(-2, tabs, stream, 0, bFigure);                    //pro překážky
if (bFigure)
        stream << '\t' << '}' << '\n' << '\n';
m_strcfg << '\t' << '}' << '\n' << '\n';

if (bFigure)
        stream.write(m_strcfg.str(), m_strcfg.pcount());
```

```
void COptDoc::WritePos(int iParent, int tabs, strstream& stream, BOOL bBranch, BOOL bFigure)
{
	//rekurzivní algoritmus, který z lineární struktury těles v počítači vytvoří zápis stromové struktury
	//v konfiguračním souboru
	BOOL bFirst = TRUE;
	CString str;
	//projede všechna tělesa v lineární struktuře
	for (int i = 0; i < m_CountOfComp; i++) {
		if (m_compArray[i].iParent == iParent) {
			//pokud je těleso ve větvi dané rodičem
			if (bBranch && bFirst) {
				WriteTabs(tabs - 1, stream);
				stream << "BRANCH" << '\n';
				WriteTabs(tabs - 1, stream);
				stream << '{' << '\n';
				bFirst = FALSE;
				//zapíše pohyb tělesa
				WriteMotion(m_iPos, i, TRUE);
				m_iPos++;
			}
			//vytvoření tabelátoru na začátku řádku
			WriteTabs(tabs, stream);
			//vypsání parametrů tělesa
			WriteBlock(i, stream, bFigure);
			if (m_compArray[i].motion.GetLength() > 0)
				WriteMotion( m_iPos, i, FALSE);
			m_iPos++;
			//rekurzivní volání
			WritePos( i, tabs + 1, stream, 1, bFigure);
		}
	}
	if (!bFirst) {
		WriteTabs(tabs - 1, stream);
		stream << '}' << '\n';
	}
}

void COptDoc::WriteTabs(int count, strstream& stream)
{
	//vypíše daný počet tabelátorů na začátku řádku
	for (int i = 0; i < count; i ++)
		stream << '\t';
}

void COptDoc::WriteBlock(int pos, strstream& stream, BOOL bFigure)
{
	//vypíše na řádek zápis jednoho tělesa
	UINT figure = m_compArray[pos].figure;
	char ch[3];
	CString s;

	stream << m_strFigure[m_compArray[pos].figure] << " { ";

	if (figure == BLOCK || figure == CENTER_BLOCK) {
		s.Format("a=%g;b=%g;c=%g;",m_compArray[pos].s0, m_compArray[pos].s1,
					m_compArray[pos].s2);
	}
```

```
        if (figure == PRISM || figure == SPIRE) {
                s.Format("r=%g;n=%g;v=%g;",m_compArray[pos].s0, m_compArray[pos].s1,
                m_compArray[pos].s2);
        }

        if (figure == RIGHT_PRISM || figure == RIGHT_SPIRE) {
                s.Format("a=%g;b=%g;v=%g;",m_compArray[pos].s0, m_compArray[pos].s1,
                                        m_compArray[pos].s2);
        }

        if (figure == TRUNC_SPIRE) {
                s.Format("r=%g;n=%g;v0=%g;v=%g;",m_compArray[pos].s0, m_compArray[pos].s1,
                                        m_compArray[pos].s2, m_compArray[pos].s3);
        }
        stream.write(s, s.GetLength());

        if (bFigure)
                stream << "LEVEL=" << _itoa(m_compArray[pos].level, ch, 10) << ';';
        if (m_compArray[pos].pos.xPos != 0 || m_compArray[pos].pos.yPos != 0 ||
                m_compArray[pos].pos.zPos != 0)
                        WriteBlockPos(pos, stream);
        if (m_compArray[pos].pos.xRot != 0 || m_compArray[pos].pos.yRot != 0 ||
                m_compArray[pos].pos.zRot != 0)
                        WriteBlockRot(pos, stream);
        stream << " }\n";
}

void COptDoc::WriteMotion(int index, int arrayPos, BOOL bBranch)
{
        //zapíše do konfiguračního souboru pohyb tělesa
        CString strSource, strPom;
        char ch[10];
        if (bBranch) {
                //ziskani retezce pohybu pro vetev
                if (arrayPos == -1)
                        strSource = m_strMotionBranchRoot;
                else
                        strSource = m_compArray[m_compArray[arrayPos].iParent].motionBranch;
        }
        else
                //získání řetězce pohybu tělesa
                strSource = m_compArray[arrayPos].motion;

        if (!strSource.IsEmpty())
                while (!strSource.IsEmpty()) {
                        m_strcfg << '\t' << '\t' << '$';
                        strPom = strSource.Left(3);
                        strPom.MakeUpper();
                        m_strcfg << strPom << '(' << _itoa(index , ch, 10) << ',';
                        m_strcfg << strSource.GetAt(5) << ")=";
                        strSource.Delete(0,12);
                        m_strcfg << strSource.Left(strSource.Find(",",0));
                        strSource.Delete(0, strSource.Find(",",0) + 1);
                        m_strcfg << ';' << '\n';
                }
}
```

```
void COptDoc::WriteBlockPos(int pos, strstream &stream)
{
	//zapíše pevnou pozici tělesa
	CString s;
	s.Format(" POSITION{%g;%g;%g}", m_compArray[pos].pos.xPos,
				m_compArray[pos].pos.yPos, m_compArray[pos].pos.zPos);
	stream.write(s, s.GetLength());
}

void COptDoc::WriteBlockRot(int pos, strstream &stream)
{
	//zapíše pevnou rotaci tělesa
	CString s;
	s.Format(" ROTATION{%g;%g;%g}", m_compArray[pos].pos.xRot,
				m_compArray[pos].pos.yRot, m_compArray[pos].pos.zRot);
	stream.write(s, s.GetLength());
}

void COptDoc::OnParametersCFG()
{
	//zobrazení dialogu pro zadání parametrů úlohy
	if (m_cfgSheet.DoModal() == IDOK)
		SetModifiedFlag();
	if (m_cfgSheet.m_constPage.m_iType) {
		//v pripade vyuziti preddefinovaneho robota
		m_CountOfComp = 0;
		m_compArray.SetSize(30);
		m_compArray.RemoveAll();
	}
	POSITION pos = GetFirstViewPosition();
	GetNextView(pos);
	GetNextView(pos);
	GetNextView(pos)->OnInitialUpdate();				//CCreateView
}

void COptDoc::DrawGraphs(CString filePath)
{
	//vykreslí grafy ze souboru
	m_ifstream.open(filePath,ios::nocreate,filebuf::sh_none);
	if (m_ifstream) {
		GetSize();
		m_ifstream.open(filePath,ios::nocreate,filebuf::sh_none);
		GetData();
	}
	m_ifstream.close();
	m_ifstream.clear();
}

void COptDoc::OnSettingsColors()
{
	//vyvolá dialog pro nastaveni barev robota v navrhovacím prostředí
	CCompColorDlg dlg;
	dlg.DoModal();
	GetRGB(dlg.m_iSelCompColor, dlg.m_iCompColor);
}
```

```
void COptDoc::OnSettingsDynopt()
{
	//vyvolá dialog, ve kterém se zadává cesta k optimalizačnímu modulu DynOpt a zapíše tuto informaci
	//do registrů Windows
	CFileDialog dlg(TRUE, NULL, "DynOpt", OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
	                                "Programy (*.exe) | *.exe");
	dlg.DoModal();

	CString str;
	str = dlg.GetPathName();


	AfxGetApp()->WriteProfileString("Opt","DynOpt_path", str);
	AfxGetApp()->WriteProfileString("Opt","DynOpt_name", dlg.GetFileName());

	//ziska pracovni adresar programu DynOpt
	str = str.Left(str.GetLength() - dlg.GetFileName().GetLength());
	AfxGetApp()->WriteProfileString("Opt","DynOpt_folder", str);
}

void COptDoc::GetRGB(int selCompColor, int compColor)
{
	//získá jednotlivé složky barev RGB z typu COLORREF
	CColorComboEx combo;
	COLORREF pom = combo.m_TraceColor[selCompColor];

	m_fSelCompColor[0] =  (float)(pom & 0x000000ff) / 255;
	pom >>= 8;
	m_fSelCompColor[1] = (float)(pom & 0x000000ff) / 255;
	pom >>= 8;
	m_fSelCompColor[2] = (float)(pom & 0x000000ff) / 255;

	pom = combo.m_TraceColor[compColor];
	m_fCompColor[0] = (float)(pom & 0x000000ff) / 255;
	pom >>= 8;
	m_fCompColor[1] = (float)(pom & 0x000000ff) / 255;
	pom >>= 8;
	m_fCompColor[2] = (float)(pom & 0x000000ff) / 255;


	m_fCompColor[0] = 0.0f;
	m_fCompColor[1] = 0.0f;
	m_fCompColor[2] = 0.0f;
}
```

```
void COptDoc::OnSaveProject()
{
	//volaná při ukládání úlohy, nebyla zde možnost volat standardní metodu CDocoment::OnSaveDocument z
	//důvodů zkrácení ukládaného jména na 8 znaků, kvůli programu DynOpt
	CString fName;
	int result = -1;
	GetPathName();
	if (IsModified() || m_bSaveAs) {
		if (GetPathName().IsEmpty() || m_bSaveAs) {
			CFileDialog fDlg(FALSE, "*.opt", fName, OFN_HIDEREADONLY |
					OFN_OVERWRITEPROMPT, _T("Soubory Opt (*.opt) | *.opt|"));

			do {
				if (result != -1)
					AfxMessageBox("Nazev souboru může mít max. 8 znaků.",
							MB_ICONINFORMATION);
				result = fDlg.DoModal();
			} while (fDlg.GetFileName().GetLength() > 12 && result == IDOK);

			if (result == IDOK) {
				SetPathName(fDlg.GetPathName());
				OnSaveDocument(fDlg.GetPathName());
				SetTitle(fDlg.GetFileName());
			}
		}
		else {
			OnSaveDocument(GetPathName());
		}
	}

	CreateCFG();
}

void COptDoc::OnUpdateSaveProject(CCmdUI* pCmdUI)
{
	//znepřístupní příkaz uložení, pokud úloha nebyla modifikována
	pCmdUI->Enable(IsModified());
}

void COptDoc::OnSaveProjectAs()
{
	//uložit úlohu jako...
	m_bSaveAs = TRUE;
	OnSaveProject();
	m_bSaveAs = FALSE;
}

BOOL COptDoc::OnCmdMsg(UINT nID, int nCode, void* pExtra, AFX_CMDHANDLERINFO* pHandlerInfo)
{
	//zpřístupnění tlačítek pro rotaci, posun a měřítko
	if (pHandlerInfo == NULL) {
		if (nCode == CN_UPDATE_COMMAND_UI) {
			if (nID >= ID_B_ROTATE && nID <= ID_B_SCALE)
				((CCmdUI*)pExtra)->Enable();
		}
	}

	return CDocument::OnCmdMsg(nID, nCode, pExtra, pHandlerInfo);
}
```