UNIVERZITA PALACKÉHO V OLOMOUCI
PŘÍRODOVĚDECKÁ FAKULTA

DIPLOMOVÁ PRÁCA

Simulácia synchronizácie robotov

Katedra matematické analýzy a aplikací matematiky
Vedúci diplomovej práce: **Mgr. Ondřej Vencálek, Ph.D.**
Vypracoval(a): **Ivana Janíková**
Študijný program: N1103 Aplikovaná matematika
Študijný obor Aplikace matematiky v ekonomii
Forma štúdia: prezenčná
Rok odovzdania: 2017

BIBLIOGRAFICKÁ IDENTIFIKÁCIA

Autor: Ivana Janíková

Názov práce: Simulácia synchronizácie robotov

Typ práce: Diplomová práca

Pracovisko: Katedra matematické analýzy a aplikací matematiky

Vedúci práce: Mgr. Ondřej Vencálek, Ph.D.

Rok obhajoby práce: 2018

Abstrakt: Spontánna synchronizácia sa nachádza všade vôkol nás a táto diplomová práca obsahuje simuláciu procesu synchronizácie robotov. Tento neriadený proces prebieha tak, že pre zvolený počet robotov, ktoré sa majú synchronizovať a každý z nich má na hodinkách náhodne nastavený jeden čas z konečnej postupnosti diskrétnych časov $0, 1, \ldots$ až do zvolenej hodnoty, sú zakaždým náhodne vybrané dva roboty, ktoré si svoje hodinky nastavia na spoločný priemer svojich časov. Dvojice robotov sa synchronizujú až do momentu, kedy majú všetky roboty rovnaký čas. Simulácia, naprogramovaná v prostredí jazyka R, má tri varianty odlišujúce sa spôsobom náhodného výberu dvoch robotov, ktoré budú v aktuálnom kroku procesu synchronizovať svoje časy. Výsledky získané opakovaným spustením každého variantu simulácie s rovnakými nastaveniami sú interpretované skrz grafy typu barplot, boxplot a histogram. Pomocou nich bola sledovaná závislosť doby procesu synchronizácie, meraná v podobe výsledného počtu krokov potrebného k zosynchronizovaniu všetkých robotov a takisto v sekundách, na množstve robotov vstupujúcich do procesu a množstve rôznych časov, ktoré môže mať jeden robot nastavené na hodinkách. Navzájom porovnávané boli i jednotlivé varianty a súčasťou práce je aj vizualizácia procesu synchronizácie robotov.

Kľúčové slová: spontánna synchronizácia, simulácia, roboty, závislosť

Počet stran: 78

Počet príloh: 1

Jazyk: slovenský

BIBLIOGRAPHICAL IDENTIFICATION

Author: Ivana Janíková

Title: Simulation of robots' synchronization

Type of thesis: Master's

Department: Department of Mathematical Analysis and Application of Mathematics

Supervisor: Mgr. Ondřej Vencálek, Ph.D.

The year of presentation: 2018

Abstract: Spontaneous synchronization is all around us and the thesis contains a simulation of synchronization process of robots. The uncontroled process works as follows: given number of robots enters the process with randomly set time on their watches from a sequence of discrete times $0, 1, \ldots$ to a certain value. Each time two robots are randomly selected and they set their watches to an average of their times. Pairs of robots are synchronized until all robots have the same time. The simulation is programmed in R enviroment and has three versions. The versions differ in the way of random selection of these two robots that will synchronize their times in the current process step. The versions of simulation are re-run and their results are interpreted by barplots, boxplots and histograms that help us to analyse dependence of synchronization process time on the number of robots entering the process and the number of different times that robot may have on its watch. The synchronization process time is measured as a number of steps needed to synchronize all robots and also in seconds. The versions of simulation were also compared and the thesis also contains a visualization of synchronization process of robots.

Key words: spontaneous synchronization, simulation, robots, dependence

Number of pages: 78

Number of appendices: 1

Language: Slovak

Prehlásenie

Prehlasujem, že som diplomovú prácu spracovala samostatne pod vedením pána Mgr. Ondřeje Vencálka, Ph.D. a všetky použité zdroje som uviedla v zozname literatúry.

V Olomouci dňa ....

podpis

Obsah

Zoznam obrázkov

Poďakovanie

Rada by som poďakovala vedúcemu mojej diplomovej práce, pánu Mgr. Ondřeji Vencálkovi, Ph.D., za jeho cenné rady a pripomienky, ochotu a všetok čas, ktorý mi na konzultáciách venoval.

Úvod

Hlavným cieľom diplomovej práce je vytvoriť program, ktorý bude simulovať proces synchronizácie robotov. Každý robot v procese synchronizácie má svoje hodinky a na nich náhodne nastavený jeden z možných diskrétnych časov $0, 1, 2, \ldots,$ pc - 1, ktoré sú rovnako pravdepodobné a kde pod pc (skratka od počet časov) rozumieme množstvo časov/časových dielikov, ktoré sú na okrúhlom ciferníku hodiniek. V simulácii je vždy vybraná dvojica robotov, ktoré si zosynchronizujú svoje časy na hodinkách ako ich priemer. Ako si však ukážeme, nie vždy bude pri priemerovaní použitý klasický aritmetický priemer. Proces synchronizácie robotov považujeme za úspešne ukončený v okamžiku, keď všetky roboty majú na svojich hodinkách nastavený rovnaký čas.

Vizuálne je proces synchronizácie prezentovaný v podobe „blikania“ robotov, ktoré evokuje blikanie svätojánskych mušiek, a teda výsledkom úspešného procesu synchronizácie je jednotné blikanie všetkých robotov.

Ďalšou úlohou diplomovej práce je vytvorenú simuláciu použiť k posúdeniu toho, ako je doba vedúca k úspešnému procesu synchronizácie robotov ovplyvnená počtom robotov, ktoré sa tohto procesu účastnia, prípadne počtom časov na ciferníku, ktoré v procese uvažujeme.

V kapitole 1 si v krátkosti objasníme pojem synchronizácia a uvedieme príklady tzv. *spontánnej synchronizácie*, na ktorú je táto diplomová práca cielená. Podrobnému popisu zadanej úlohy, jej riešeniu, tvorbe simulácie v prostredí jazyka R [8] a popisu naprogramovaných funkcií, ktoré simulácia používa, je venovaná kapitola 2. V rámci nej sú rozobrané i tri odlišujúce sa varianty simulácie, ku ktorým sme počas práce na programe dospeli. Tieto tri varianty simulácie

využijeme v kapitole 3 k získaniu dát, ktoré následne analyzujeme a zaujímať nás bude najmä závislosť doby procesu synchronizácie na počte robotov, resp. počte časov. Dáta prezentujeme graficky vo forme grafov typu barplot, boxplot a histogram (takisto v prostredí jazyka R), pomocou ktorých môžeme navzájom porovnávať i jednotlivé varianty simulácie. Posledná časť, kapitola 4, obsahuje detailnejší návod, ako proces synchronizácie previesť do jednoduchej vizuálnej podoby spomínaného blikania svätojánskych mušiek, a to konkrétne do formátu .mp4.

Kapitola 1

Synchronizácia

Pod pojmom synchronizácia môžeme v širšom slova zmysle rozumieť uvedenie udalostí do vzájomného súladu, do spoločného rytmu. Nastane, keď sa dvaja alebo akékoľvek veľké množstvo jednotlivcov skoordinuje a začnú sa správať/fungovať jednotne ako jeden systém.

Vôkol nás nájdeme rozličné typy kolektívneho správania ako príklady synchronizácie: synchronizácia spočívajúca v zosúladení smeru pohybu, synchronizácia dát, synchronizácia obrazu a zvuku, synchronizácia času a iné. Tendencia k synchronizácii sa dotýka mnohých oblastí života a je všade okolo nás. Tiahne sa od subatomárnej úrovne cez zvieraciu ríšu, ľudstvo až k najväčším vesmírnym diaľkam, ako o tom píše uznávaný americký matematik, profesor aplikovanej matematiky na Cornell University a známy popularizátor vedy Steven Strogatz vo svojej knihe *Sync* [12].

Väčšine ľudí možno ako prvé v súvislosti so synchronizáciou napadnú akvabely a ich dokonalo vyzerajúce synchronizované plávanie, keďže sa toto slovo nachádza priamo v názve športu. Iným napríklad vojenský pochod či vystúpenie mažoretiek. Podobných príkladov by sme mohli nájsť veľa, často sa týkajú hudby alebo tanca, čo však majú spoločné? Žiadajú si určitý riadiaci prvok, režiséra, dirigenta, ktorého pokynmi sa všetci riadia, alebo choreografiu, ktorú všetci dodržujú, a to vedie k vzniku synchronizácie. Táto synchronizácia ale nie je tá, ktorou sa chceme v tejto diplomovej práci zaoberať. Nás bude zaujímať práve zlaďovanie rytmických javov, ktoré je automatické, tzv. *spontánna synchronizácia*. Pri spon-

tánnej synchronizácii sa jednotlivci riadia a organizujú sami. Nikto nie je vo vedení. V nasledujúcej kapitole si predstavíme niekoľko príkladov spontánnej synchronizácie z rôznych oblastí ako motiváciu k štúdiu tejto problematiky [3].

1.1. Príklady spontánnej synchronizácie

V predošlých riadkoch sme spomenuli zopár príkladov synchronizovaného správania, ktoré býva riadené. Teraz si uveďme príklady synchronizácie, ktorá nastane spontánne.

Chôdza

Denne môžeme byť sami účastníkmi spontánnej synchronizácie. Stačí ísť bok po boku s iným človekom a o chvíľu si možno všimneme, že kráčame rovnako bez toho, aby sme sa o to cielene pokúšali. Kroky sa nám zladili a majú ten istý rytmus.

Potlesk

I občasní návštevníci divadla, opery či koncertnej haly vedia, že po skončení predstavenia nasleduje potlesk obecenstva. Intenzita a povaha potlesku vyjadruje, ako nadšené bolo obecenstvo výkonom účinkujúcich. Práve tlieskajúcim publikom sa zaoberal vedec Z. Néda a jeho kolegovia, ktorí nahrali a analyzovali potlesk ľudí v divadlách a operách v Rumunsku a v Maďarsku. Spočiatku búrlivý a nejednotný potlesk sa často náhle zmení na synchronizované tlieskanie. Tento fenomén, kedy diváci tlieskajú simultánne a periodicky, môže zaniknúť a znova sa objaviť i niekoľkokrát [6].

Millennium Bridge

Millennium Bridge je oceľový visutý most pre chodcov v Londýne, ktorý bol postavený cez rieku Temža na konci 90. rokov. Architektúra mosta v podobe dosť tenkej a plochej oceľovej lávky s lanami po stranách mostu má podľa vízie jeho autorov pripomínať ostrie svetelného meča - svetelné ostrie. V deň jeho slávnostného otvorenia v roku 2000 sa však stalo niečo neočakávané, ale pozoruhodné. Chvíľu

po tom, čo návštevníci vstupovali na most a začali sa prechádzať, sa most začal hýbať drobnými kmitmi do strán. Aby si chodci udržali rovnováhu, boli, prirodzene, nútení prispôsobiť svoju chôdzu pohybom mosta. Tá pripomínala pohyby začínajúceho korčuliara na ľade, rozkročmo a zo strany na stranu. Ľudí na moste pribúdalo až zrazu všetci postupovali mostom napodobujúc tieto korčuliarske pohyby takmer v perfektnom súlade. Táto neúmyselná synchrónna chôdza zároveň spôsobovala zosilňovanie sprvu malého kolísania mosta, ktoré sa zmenilo na celkom strach naháňajúce kymácanie. Most bol z dôvodu bezpečnostných obáv behom dvoch dní pre verejnosť uzatvorený a začalo sa skúmanie tohto nepredpokladaného javu. Riešením nakoniec bolo umiestnenie niekoľkých desiatok nenápadných hydraulických tlmičov naspodok mosta. Nevšedný príklad synchronizácie ako medzi ľuďmi, tak i medzi nimi a vibráciami mosta [3],[5].

Tlkot srdca

Približne desať tisíc buniek riadi tlkot ľudského srdca. Tieto bunky tvoria tzv. sinoatriálny uzol a spoločne udávajú rytmus, kedy sa srdce sťahuje a uvoľňuje. Samostatne má každá bunka svoj vlastný cyklus, v ktorom sa nabíja a potom vyšle elektrický impulz. Keď sú však viaceré bunky v tesnej blízkosti, chvíľku si držia samostatný rytmus, no v dôsledku vzájomnej komunikácie vo forme vysielaných elektrických nábojov prirodzene prejdú do súladu, vďaka čomu naše srdce rytmicky bije [16].

S búšením srdca súvisí i experiment, ktorý uskutočnili výskumníci z Univerzity v Colorade. Vo svojom pokuse bolestivo zahrievali predlaktie partnerom držiacim sa za ruky a ich tlkot srdca sa postupne zosynchronizoval, čo v závere vedie k zníženiu ich bolesti, ktorú v tej chvíli prežívajú. Výsledky tejto štúdie sú autormi interpretované ako dôkaz, že prítomnosť mužov pri pôrode pôsobí ako analgetikum pre ich rodiace partnerky.

Z príkladov spontánnej synchronizácie prebiehajúcej medzi ľudmi či priamo v ľudskom tele môžeme ešte spomenúť napríklad epileptický záchvat alebo triašku u ľudí trpiacich Parkinsonovou chorobou ako nie práve obdivuhodný následok

zosúladenia neurónov v mozgu. Pomerne známe sú tiež reportovné výsledky niekdajších experimentov, podľa ktorých majú menštruačné cykly spolubývajúcich žien tendenciu k spontánnej synchronizácii. Pravdivosť tohto v súčasnosti už obecne rozšíreného výsledku je však odborníkmi značne kritizovaná a spochybňovaná [3],[11],[16].

Svätojánske mušky

Svätojánske mušky v juhovýchodnej Ázii, v oblastiach Malajzie, Thajska a Novej Guiney, sú jedným z očarujúcich príkladov spontánnej synchronizácie, ktorý nájdeme v prírode. Tisíce svätojánskych mušiek, ktorými sú „posiate“ stromy mangrovníky pozdĺž brehov riek, sa rozsvecuje a zhasína každú noc v perfektnej zhode. Presnejšie sú to samčeky komunikujúce pomocou svetla, ktoré nám predvádzajú túto svetelnú šou. Spočiatku, keď sa stmieva, je ich blikanie nekoordinované. No s prehĺbujúcou sa nocou vznikajú zosúladené skupinky svätojánskych mušiek, synchronizácia sa šíri ďalej až nakoniec celý breh rieky pulzuje v dokonalej harmónii i niekoľko hodín.

Aj v tomto prípade, vo svete zvierat, nájdeme mnoho ďalších príkladov samovoľnej synchronizácie. Medzi nimi napríklad synchrónne cvrkajúce cvrčky či obrovský kŕdeľ škorcov, v ktorom všetky naraz predvádzajú vo vzduchu pozoruhodné manévre meniac smer bez toho, aby došlo k akejkoľvek zrážke. Podobné správanie môžeme vidieť aj pri veľkom húfe rýb. U oboch to často súvisí i s obranou a vyhýbaním sa dravcom [3],[7],[10],[16].

Huygens a kyvadlové hodiny

Mnohé zo spomínaných príkladov boli vedecky skúmané fyzikmi, matematikmi a biológmi 20. a 21. storočia, ktorí upriamili svoju pozornosť na fenomén spontánnej synchronizácie. Avšak prvá písomná zmienka o tomto fenoméne sa viaže k roku 1665. V tomto roku si holandský fyzik, matematik, astronóm a vynálezca kyvadlových hodín Christiaan Huygens, ležiac vo svojej posteli a liečiac sa z ľahšieho ochorenia, všimol tento „zvláštny druh sympatie“ u svojich dvoch nástenných hodín, ktoré viseli na drevenom tráme a kývali sa presne proti sebe v rovna-

kom rytme. Zaujatý týmto zistením začal s hodinami experimentovať. Umiestnil ich na dosku, ktorú položil na dve stoličky stojace operadlom proti sebe, a spozoroval, ako sa hodiny časom dostali prirodzene do súladu. Ak však hodiny umiestnil každú na inú stranu izby, postupne sa zosúladené hodiny rozišli a synchronizácia už nenastala. Huygens sa tak domnieval, že hodiny sa musia nejakým spôsobom navzájom ovplyvňovať, ako napríklad skrz drobné vibrácie dosky či nebadateľné pohyby vzduchu. Huygensove zistenia boli však po zverejnení zväčša ignorované a ďalším nasledovníkom, ktorý sa začal jeho myšlienkou zaoberať, bol až v 60. rokoch 20. storočia teoretický biológ Arthur Winfree [5],[14],[16].

Metronómy

Jednoduchým spôsobom ako demonštrovať spontánnu synchronizáciu je použitie dvoch a viac metronómov, ktoré položíme na nejakú tenkú plochu. Túto plochu spolu s metronómami umiestníme na dve ležiace plechovky. Týmto spôsobom môžu metronómy navzájom komunikovať pomocou mechanických síl, pretože tenká plocha, ktorá sa môže na plechovkách voľne pohybovať, ich spája, a tak umožňuje prenášať jemné pohyby spôsobené kmitaním ručičiek metronómov. Takto môžeme byť svedkami, ako sa metronómy postupne dostanú prirodzene do súladu [3],[13],[15].

Ako je vidieť z predchádzajúcich odstavcov, spontánna synchronizácia nás obklopuje z mnohých strán. Ešte pred všetkými týmito príkladmi sme si mohli myslieť, že samovoľná tendencia k synchronizácii, kedy z chaosu postupne vznikne poriadok, vyžaduje bytosť s mozgovou činnosťou na vyššej úrovni. Potom sme si však ukázali príklad synchronizácie vtákov a rýb, z čoho by sme sa mohli domnievať, že k synchronizácii stačí i úroveň myslenia, ktorá prinajmenšom umožní živému organizmu podvedome sledovať vrodené zvyky o vzdialenosti a rýchlosti. Avšak mozog svätojánskej mušky je možné len ťažko prirovnávať k procesom prebiehajúcim v mozgu vtáka či ryby. V tomto momente sa teda môže objaviť otázka, či účastník synchronizácie musí vôbec mať mozog alebo dokonca byť živým objektom. Práve bunky v srdci a inde v tele žijúceho jedinca, či obyčajné kyvadlá

sú jedným z mnohých dôkazov, že tomu tak nie je a k spontánnej synchronizácii môže dôjsť a aj dochádza v širokej škále ľudských, biologických i mechanických systémov, od atómov až k planétam, ako napríklad v prípade Mesiaca, ktorý je v tzv. synchrónnej rotácii so Zemou, v dôsledku čoho vidíme stále rovnakú stranu Mesiaca [13].

1.2. Kiloboty

Motivácie k štúdiu synchronizácie v podobe uvedených príkladov je vskutku dostatok. Čo nás takisto inšpirovalo k nasledujúcim kapitolám je projekt s názvom Kilobot, za ktorého vznikom stojí Výskumná skupina samoorganizujúcich sa systémov z Harvard University. Na projekte spolupracovali Mike Rubenstein, Radhika Nagpal a ich kolegovia. Projekt dokonca získal miesto v „Top 10“ prelomových vedeckých úspechov roku 2014 časopisu Science a rovnaké umiestnenie i v časopise Nature.

Pod názvom kilobot sa skrýva maličký a veľmi jednoduchý robot, ktorého cena je skutočne nízka. Výskumníci vytvorili pre potreby experimentov v oblasti kolektívneho správania v rámci veľkých autonómnych rojov 1024 takýchto robotov, tzv. roj kilobotov. Každý robot má priemer 33 mm, tri jednoduché nožičky a dva vibračné motory, ktoré v prípade, že vibrujú individuálne, tak sa robot otáča, ak vibrujú súčasne, tak ide rovno. kilobot je naprogramovaný tak, aby dokázal reagovať na svetlo, merať vzdialenosť či „vnímať“ prítomnosť iných kilobotov. So svojimi susedmi dokáže komunikovať pomocou infračerveného svetla odrážajúceho sa od povrchu pod nimi, a to až do vzdialenosti 7 cm.

Všetky kiloboty sú pred experimentom naprogramované naraz ako skupina a každý kilobot dostane rovnaké inštrukcie ako tie ostatné. V experimente dokážu kiloboty spolupracovať bez ďalšieho usmerňovania či zásahu zvonku, prejavovať komplexné správanie a sú príkladom toho, akú komplexnosť môže dosiahnuť jednoduché správanie 1024 malých strojov. Dokážu sa napríklad organizovať do rozličných tvarov, podobne ako bunky v našom tele vytvoria orgán, alebo napodobniť blikanie svätojánskych mušiek, kedy roj kilobotov na začiatku

bliká náhodne až nakoniec dôjde k ich synchronizácii a začnú blikať spoločne, pretože pomocou vhodného algoritmu roboty prispôsobujú svoje vnútorné hodinky. V okamžiku, keď hodinky kilobota dosiahnu najvyššiu hodnotu, tak sa vynulujú a kilobot pošle svojím susedom signál. Vždy keď susedný robot tento signál príjme, nastaví si svoje vlastné hodinky podľa toho, kedy signál prijal. Postupom času týmto prispôsobovaním nastane súlad v blikaní všetkých kilobotov [4], [9].

Kapitola 2

Simulácia procesu spontánnej synchronizácie

V tejto kapitole sa najprv oboznámime so zadaním úlohy a následne vysvetlíme spôsob a štruktúru jej riešenia.

2.1. Zadanie a riešenie úlohy simulácie procesu synchronizácie robotov

Úlohou a cieľom diplomovej práce je vytvoriť simuláciu procesu spontánnej synchronizácie robotov. V zadaní i riešení tejto simulácie je možné nájsť podobnosť s príkladom svätojánskych mušiek a kilobotov, s ktorými sme sa v krátkosti zoznámili v podkapitole 1.1 a 1.2, preto si ich môžeme predstaviť v roli našich robotov, ktoré chceme zosynchronizovať. Zadanie a smer, ktorým sa budeme pri programovaní simulácie uberať, vychádzajú z [2] a [17].

Pre lepšiu predstavu úlohy, s ktorou sa budeme ďalej potýkať, uvažujme malé množstvo robotov, napríklad 10. Toto množstvo robotov označíme pre budúce účely ako `pr` (skratka pre počet robotov a v zdrojovom kóde simulácie predstavuje aktuálne uvažovanú hodnotu z vektora `pocet_robotov`, pozri v 2.2.4). Všetky roboty sú náhodne rozmiestnené na štvorcovej ploche. Ďalej si predstavme, že každý robot má svoje hodinky, ktoré majú len sekundovú ručičku. Z dôvodu jednoduchosti však uvažujme, že hodinky nemajú 60 sekúnd, ale len 5. Táto hodnota predstavuje množstvo časov (časových dielikov), ktoré súčasne

uvažujeme, a označíme ho, opäť pre budúce účely, ako `pc` (skratka pre počet časov a v zdrojovom kóde simulácie predstavuje aktuálne uvažovanú hodnotu z vektora `pocet_casov`, pozri v 2.2.4). Na začiatku procesu synchronizácie má každý robot nastavený na svojich hodinkách náhodný čas, tzn. ručička jeho hodiniek ukazuje na jeden z 5 časových dielikov ciferníka, a ním môže byť čas/časový dielik 0, 1, 2, 3 alebo 4, pričom všetky hodnoty sú rovnako pravdepodobné. Pre každú hodnotu `pc` budeme totiž začínať od 0. Máme teda 10 robotov, a tak aj 10 časov, z ktorých každý prislúcha jednému z robotov. Keďže `pc = 5`, je zrejmé, že niektoré roboty budú mať na začiatku na hodinkách nastavený rovnaký čas, čo ale ničomu neprekáža.

Cieľom simulácie procesu synchronizácie je dosiahnuť stav, keď všetkých 10 robotov bude mať na svojich hodinkách ten istý čas. Proces synchronizácie bude prebiehať tak, že zakaždým náhodne vyberieme dva roboty. Zistíme, aké dva časy, či už rovnaké alebo rôzne, patria týmto dvom robotom. V tejto chvíli sa vynára otázka, ako bude prebiehať komunikácia dvoch robotov, aby jej výsledkom bol rovnaký čas pre oboch. Riešenie, ktoré sme zvolili, je, že tieto dva časy zpriemerujeme a výslednú priemernú hodnotu časov vybraným dvom robotom nastavíme. Následne opäť vyberieme náhodne dva roboty a postup opakujeme. Takto pokračujeme stále dookola, až kým nedôjdeme k momentu, kedy všetkých 10 robotov bude mať rovnaký čas, čo znamená, že došlo k synchronizácii a proces synchronizácie je ukončený.

Priemerovaniu dvojice časov a celej simulácii procesu synchronizácie sa budeme venovať bližšie v ďalších podkapitolách. Po vytvorení simulácie je ďalšou úlohou túto simuláciu využiť napríklad k odhadu strednej doby procesu synchronizácie. Taktiež sa budeme zaoberať skúmaním rôznych závislostí, ako napríklad závislosť strednej doby procesu synchronizácie a množstva robotov `pr`, ktoré vstupuje do procesu, či množstva časových dielikov `pc`, ktoré uvažujeme.

2.2. Varianty simulácie a ich štruktúra v jazyku R

K vytvoreniu zdrojového kódu pre simuláciu procesu synchronizácie využijeme programovací jazyk R [8]. Toto programovacie prostredie obsahuje množstvo funkcií ako pre manipuláciu a štatistické spracovanie dát, tak i pre ich grafické zobrazenie. Mnoho ďalších funkcií je možné využiť prostredníctvom podporných balíčkov, ktoré sa dajú veľmi jednoducho a rýchlo nainštalovať.

Počas práce na simulácii sme dospeli k niekoľkým spôsobom, ako chceme, aby proces synchronizácie robotov prebiehal. Vznikli tak tri varianty zdrojového kódu. V čom sa tieto varianty odlišujú si teraz v krátkosti popíšeme a načrtneme taktiež štruktúru ich zdrojového kódu.

2.2.1. Variant 1

Priebehu procesu synchronizácie, ktorý sme si opísali v podkapitole 2.1, variant 1 v podstate odpovedá. Zo všetkých robotov, ktoré vstúpili do synchronizácie, ich počet je `pr`, náhodne vyberieme naraz dvoch rôznych a zistíme, aké časy majú na svojich hodinkách. Dôležité je, že vo variante 1 je pravdepodobnosť výberu pre všetky dvojice rovnaká. Zistenej dvojici časov vypočítame ich priemer a priradíme ho obom robotom z vybranej dvojice. Takto postupujeme, až kým nemajú všetky roboty zhodný čas.

Čo sa týka priemerovania, to neprebieha pri rôznych dvojiciach časov vždy rovnako a klasicky, ako sme zvyknutí. Znamená to, že niekedy nastane situácia, kedy nám obyčajný aritmetický priemer nebude stačiť. Detailne si postup priemerovania dvoch časov robotov vysvetlíme v podkapitole 2.3 v rámci popisu funkcie `Vypocitaj_priemer_cas()` (pozri 2.3.7), ktorá má priemerovanie na starosti.

V zadaní našej úlohy sme navyše spomenuli, že všetky roboty sú náhodne umiestnené na nejakej štvorcovej ploche. Každý robot má na nej od začiatku svoju polohu, ktorá je daná dvojicou náhodne vygenerovaných hodnôt predstavujúcich `x`-ovú a `y`-ovú súradnicu. Tú istú polohu si robot drží počas celého

procesu synchronizácie, nedochádza teda k žiadnemu pohybu robotov v oblasti štvorcovej plochy.

Variant 1 môžeme považovať za základ pre ďalšie dva varianty v zmysle, že taktiež využívajú pri priemerovaní rovnaký postup a takisto má každý robot pred začiatkom procesu synchronizácie počiatočnú polohu na rovnakej štvorcovej ploche.

Pre lepší prehľad uvádzame zoznam funkcií, ktoré sú volané v priebehu simulácie variantu 1 a sú teda súčasťou jeho zdrojového kódu:

- `SynchPreDvojicuPrAPc()`,
- `SynchronizujVAR1()`,
- `SuVsetkyRovnake()`,
- `UrcPozicie2Robotov()`,
- `Vypocitaj_priemer_cas()`,
- `GenerujNazov()`,
- `UlozGraf()`,
- `UlozGrafy_poSynch()`,
- `UlozSynchGraf()`.

2.2.2. Variant 2

Tento variant sa od variantu 1 odlišuje tým, že dvoch robotov, ktoré majú synchronizovať svoje časy, nevyberáme naraz. V tomto prípade je postup následovný. Predstavme si, že všetky roboty sa nachádzajú na zmieňovanej štvorcovej ploche. Z celkového počtu robotov `pr` najprv vyberieme náhodne jedného robota. Ten má na hodinkách svoj čas a svoju polohu `[x,y]` na ploche. Keďže poznáme aj polohy ostatných robotov, zistíme, ktorý z nich je k vybranému robotu najbližšie a toho vyberieme (pozri funkciu `VyberRobot2_vzd()` v 2.3.10). Tento prístup bol zvolený ako jeden z možných spôsobov (i keď pomerne jednoznačný), ako

„obmedziť“ to, aby vybrané dva roboty mali svoje polohy od seba príliš vzdialené v rámci plochy, teda aby nedochádzalo k priemerovaniu robotov, ktoré sa nemôžu „vidieť“, ak to prevedieme na podobnosť s príkladom svätojánskych mušiek, kŕdľa škorcov či húfa drobných rybičiek z podkapitoly 1.1. Získali sme tak dvojicu robotov a im odpovedajúcu dvojicu ich časov, môžeme teda pristúpiť k ich priemerovaniu. Vypočítaný priemer je obom robotom z dvojice nastavený ako ich nový čas.

U tohto variantu je však ešte ďalší rozdiel a to v tom, že roboty sa v rámci štvorcovej plochy budú pohybovať. Pred tým, než budeme znovu náhodne vyberať jedného robota, sa všetky roboty posunú náhodným smerom zo svojej aktuálnej polohy na ploche. Tým sa samozrejme zmenia aj ich aktuálne súradnice `[x,y]` na nové. Po posune robotov sa celý postup z predchádzajúceho odstavca opakuje, až kým nemajú všetky roboty rovnaký čas. Viac o tomto pohybe robotov sa dočítame v podkapitole 2.3 u funkcie s názvom `PosunVektor()` (pozri 2.3.12).

Opäť prikladáme zoznam funkcií, ktoré sú volané v priebehu simulácie variantu 2 a sú teda súčasťou jeho zdrojového kódu:

- `SynchPreDvojicuPrAPc()`,
- `SynchronizujVAR2()`,
- `SuVsetkyRovnake()`,
- `UrcPozicie2Robotov_vzd()`,
- `VyberRobota1()`,
- `Vypocitaj_vzdialenosti()`,
- `VyberRobota2_vzd()`,
- `Vypocitaj_priemer_cas()`,
- `PosunVektor()`,
- `GenerujNazov()`,

- `UlozGraf()`,
- `UlozGrafy_poSynch()`,
- `UlozSynchGraf()`.

2.2.3. Variant 3

Posledný variant simulácie je podobný variantu 2. Roboty sa tiež pohybujú, a tak sa po každom výpočte priemeru časov dvoch robotov a jeho priradení týmto robotom všetky posunú rovnakým spôsobom ako v predchádzajúcom variante. Odlišnosť od variantu 2 nastáva v bode, keď potrebujeme zistiť druhého robota. Máme teda už náhodne vybraného jedného robota a máme aj vypočítané vzdialenosti ostatných robotov od neho. Oboje rovnakým spôsobom ako vo variante 2. V tejto chvíli však nevyberáme druhého robota ako toho s najmenšou vzdialenosťou, ale najprv transformujeme všetky vypočítané vzdialenosti na pravdepodobnosti, kde potom každá z nich predstavuje mieru, s akou môžeme očakávať, že ako druhý robot do dvojice bude vybraný práve ten robot, ktorému táto pravdepodobnosť prislúcha. Druhý robot, ktorého potrebujeme do dvojice, je nakoniec vybraný náhodne podľa postupu, ktorý využíva predchádzajúce pravdepodobnosti. Viac si popíšeme v podkapitole 2.3 u funkcie s názvom `VyberRobota2_pst()` (pozri 2.3.11).

Nasleduje zoznam funkcií, ktoré sú volané v priebehu simulácie variantu 3 a sú teda súčasťou jeho zdrojového kódu:

- `SynchPreDvojicuPrAPc()`,
- `SynchronizujVAR3()`,
- `SuVsetkyRovnake()`,
- `UrcPozicie2Robotov_pst()`,
- `VyberRobota1()`,
- `Vypocitaj_vzdialenosti()`,

- `VyberRobota2_pst()`,
- `Vypocitaj_priemer_cas()`,
- `PosunVektor()`,
- `GenerujNazov()`,
- `UlozGraf()`,
- `UlozGrafy_poSynch()`,
- `UlozSynchGraf()`.

2.2.4. Štruktúra variantov

Všetky 3 varianty zdrojového kódu simulácie začínajú definovaním premenných – vektorov a matíc – a ich východiskových hodnôt. Zväčša ide o nulové vektory a matice, do ktorých sú v priebehu procesu synchronizácie zapisované výsledky simulácie. Na samom začiatku zdrojového kódu sa však nachádza zopár premenných, ktoré je potrebné pred spustením simulácie nastaviť na hodnoty podľa vlastných preferencií. Simulácia je naprogramovaná tak, že pred jediným spustením celého zdrojového kódu, resp. hlavného kódu (pozri obrázok 2.2) je možnosť zvoliť naraz niekoľko rôznych hodnôt `pr` (počet robotov), ktoré potom tvoria vektor `pocet_robotov`, a naraz niekoľko rôznych hodnôt `pc` (počet časov), ktoré potom tvoria vektor `pocet_casov`. Tieto a ďalšie vstupné premenné sú nasledujúce:

- `pocet_robotov` - premenná, ktorá určuje, aké množstvá robotov vstupujú do procesu synchronizácie. Ide o vektor ľubovoľnej dĺžky, kde každý prvok vektoru bude v procese synchronizácie predstavovať spomínanú hodnotu `pr`,
- `pocet_casov` - premenná určuje, aké množstvá časov/časových dielikov uvažujeme v procese synchronizácie. Opäť ide o vektor ľubovoľnej dĺžky a v tomto prípade bude každý prvok vektoru predstavovať v procese synchronizácie spomínanú hodnotu `pc`,

- `pocet_behov` - táto premenné určuje, koľkokrát chceme spustiť proces synchronizácie pri konkrétnom `pc` (počte časov) a `pr` (počte robotov), pričom každé takéto opakovanie (beh) pre danú dvojicu `pc` a `pr` vychádza z rovnakého vektoru `roboty_casy` (náhodne vygenerovaný vektor celočíselných hodnôt 0 až `pc` a dĺžky `pr`),

- `pocet_ulozeni` - premenná sa používa v prípade, ak chceme vytvárať grafy, keď sú časy všetkých robotov už zosynchronizované. Vyjadruje, koľkokrát ešte „obehne ručička na hodinkách celý ciferník“, bližšie pozri funkciu 2.4.3, v ktorej sa táto premenná vyskytuje.

Tieto štyri premenné sa nachádzajú v každom variante zdrojového kódu. Variant 2 a variant 3 obsahuje ešte jednu premennú a tou je smerodajná odchýlka `smer_odch`, ktorej hodnota sa nastavuje v súvislosti s pohybom robotov na štvorcovej ploche. Obrázok 2.1 ukazuje príklad nastavenia hodnôt týchto vstupných premenných.

```
pocet_behov <- 50
pocet_casov <- c(12,25)
pocet_robotov <- c(100,200,300)
smer_odch <- 0.05
```

Obr. 2.1: Vstupné premenné

Po definíciách potrebných premenných nasledujú definície funkcií, ktoré simulácia vo svojom chode využíva. Tie si rozoberieme v ďalšej podkapitole 2.3. Na konci zdrojového kódu sa nachádza hlavný kód, ktorý v prípade, že máme v prostredí uložené definície všetkých potrebných funkcií a premenné s ich vstupnými hodnotami, spustí celú simuláciu procesu synchronizácie robotov. Pozri obrázok 2.2.

Spustením hlavného kódu prebehne celý proces synchronizácie pre každú jednu neopakujúcu sa dvojicu `(pc,pr)`. Najprv zoberie prvú hodnotu vektoru `pocet_casov` a pre toto množstvo časov vezme postupne každú hodnotu vektoru `pocet_robotov`. Po skončení zoberie druhú hodnotu vektoru `pocet_casov`

```
##### HLAVNY KOD #####

m <- 1
for (pc in pocet_casov)
{
  polovica_casov <- pc/2
  casy <- 0:(pc-1)
  for (pr in pocet_robotov)
    {
      x <- runif(pr)
      y <- runif(pr)
      roboty_pozicie <- 1:pr
      roboty_casy <- sample(casy, pr, replace = TRUE)

      SynchPreDvojicuPrAPc()

      tab_vektory_dob_synch[m,] <- vektor_doby_synch
      tab_vektory_krokov[m,] <- vektor_kroky
      tab_vektory_hodnot_synch[m,] <- vektor_synch_hodnoty

      priem_dob_synch[m] <- mean(vektor_doby_synch)
      priem_krokov[m] <- mean(vektor_kroky)
      priem_synch_hodnot[m] <- mean(vektor_synch_hodnoty)
      m <- m+1
    }
}
tab_priem_dob_synch <- matrix(priem_dob_synch, nrow = length(pocet_robotov),
                              dimnames = list(pocet_robotov, pocet_casov))
tab_priem_krokov <- matrix(priem_krokov, nrow = length(pocet_robotov),
                           dimnames = list(pocet_robotov, pocet_casov))
tab_priem_synch_hod <- matrix(priem_synch_hodnot, nrow = length(pocet_robotov),
                              dimnames = list(pocet_robotov, pocet_casov))
```

Obr. 2.2: Hlavný kód

(v prípade, že má tento vektor dĺžku viac ako 1), predstavujúcu aktuálne uvažované množstvo časov, a opäť vytvorí dvojicu postupne s každou hodnotou vektoru `pocet_robotov`, predstavujúcou aktuálne uvažované množstvo robotov. Získame tak toľko dvojíc `(pc,pr)`, pre ktoré sa proces synchronizácie uskutočnil, koľko je násobok dĺžok vektorov `pocet_casov` a `pocet_casov`.

Pre každú dvojicu `(pc,pr)` sú ako prvé náhodne vygenerované východiskové `x`-ové a `y`-ové súradnice pre polohy robotov v rámci štvorcovej plochy a taktiež je náhodne vygenerovaných `pr` časov (vektor `roboty_casy`), jeden pre každého robota, ktoré sa však môžu opakovať. Nasleduje volanie funkcie `SynchPreDvojicuPrAPc()`, pozri 2.3.1. Jej výsledky pre každú dvojicu `(pc,pr)` sú zapísané do matíc, ktoré sú na konci simulácie vypísané v konzole ako výstup simulácie.

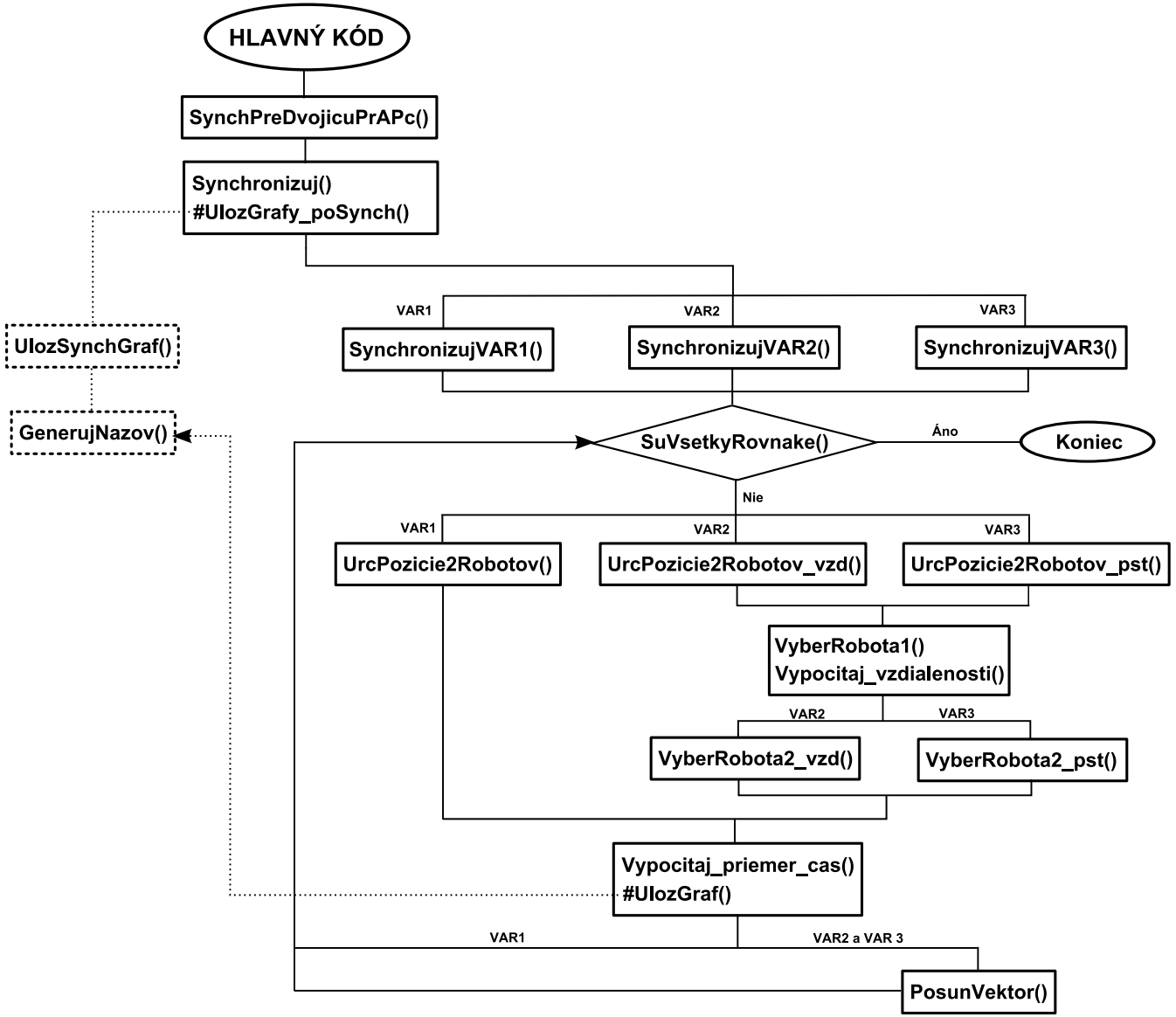
2.3. Popis funkcií v simulácii

V podkapitole sa zoznámime s funkciami definovanými v zdrojovom kóde simulácie z dôvodu jednoduchšieho a rýchlejšieho porozumenia funkčnosti naprogramovanej simulácie procesu synchronizácie robotov. K lepšiemu zorientovaniu sa môže pomôcť i obrázok 2.3 zobrazujúci prepojenie funkcií vyskytujúcich sa vo variantoch.

2.3.1. SynchPreDvojicuPrAPc()

Kód funkcie `SynchPreDvojicuPrAPc()` je takmer rovnaký pre všetky tri varianty simulácie. Funkcia je volaná priamo z hlavného kódu vždy pre aktuálnu dvojicu `(pc,pr)`. Jej úlohou je pre túto dvojicu `(pc,pr)` spustiť proces synchronizácie, či už raz, alebo niekoľkokrát v závislosti od hodnoty nastavenej v premennej `pocet_behov`, pozri v 2.2.4.

Súčasťou funkcie je aj meranie trvania doby procesu synchronizácie, ktoré sa spustí pred zavolaním funkcie `SynchronizujVAR1()`, prípadne funkcie `SynchronizujVAR2()` alebo funkcie `SynchronizujVAR3()` (obecne o ktorejkoľvek z týchto funkcií pozri 2.3.2 s názvom `Synchronizuj()`), podľa toho, v kto-



Obr. 2.3: Grafické prepojenie funkcií

rom variante zdrojového kódu sa práve nachádzame, a stopne sa po vrátení jej výsledku. Ním je vektor rovnakých časov robotov, ktorý vznikol procesom synchronizácie z pôvodného vektora `roboty_casy`. Nakoniec sú pre každý beh uložené do odpovedajúceho vektora tieto hodnoty: trvanie doby procesu synchronizácie (`doba_synchr[3]`), čas, na ktorý sa všetky roboty zosynchronizovali (`synch_hodnota`), a počet krokov (`k`), koľko bolo potrebných, než došlo k synchronizácii časov všetkých robotov. Tieto tri hodnoty budeme detailne sledovať v kapitole 3.

V priebehu funkcie `SynchPreDvojicuPrAPc()` je taktiež možnosť ukladať grafy volaním funkcie `UlozGrafy_poSynch()`. Vo východiskovom nastavení zdrojového kódu každého variantu je funkcia vložená ako komentár z dôvodu potreby určitých dodatočných nastavení. Podrobnejšie o tejto funkcii v 2.4.3. Definíciu kódu funkcie `SynchPreDvojicuPrAPc()` vo variante 3 pozri na obrázku 2.4.

```
SynchPreDvojicuPrAPc <- function()
{
  for (j in 1:pocet_behov)
  {
    start_synchr <- proc.time()

    pom <-  SynchronizujVAR3()
    zosynchr_casy <- pom$vektor_casov
    sur_x <- pom$sur_x
    sur_y <- pom$sur_y
    synch_hodnota <- zosynchr_casy[1]

    konec_synchr <- proc.time()

    doba_synchr <- (konec_synchr - start_synchr)

    #UlozGrafy_poSynch(k, zosynchr_casy, sur_x, sur_y)

    vektor_doby_synch[j] <<- doba_synchr[3]
    vektor_kroky[j] <<- k
    vektor_synch_hodnoty[j] <<- synch_hodnota
  }
}
```

Obr. 2.4: Funkcia `SynchPreDvojicuPrAPc()` vo variante 3

2.3.2. Synchronizuj()

Hlavnou úlohou funkcie `Synchronizuj()` je pri každom náhodnom výbere dvojice robotov, ktoré majú na svojich hodinkách nastavené nejaké časy, priradiť obom robotom z dvojice rovnaký čas, ktorý je priemerom ich aktuálnych časov. Volaním funkcie `UrcPozicie2Robotov()` (vzhľadom k variantu pozri 2.3.4, 2.3.5 alebo 2.3.6) najprv získame náhodnú dvojicu pozícií robotov (`dvojica_pozicie`). Pozíciou rozumieme niektorú hodnotu z vektora `roboty_pozicie`, ktorý je postupnosťou čísel 1 až `pr`. Tento vektor je vytvorený v hlavnom kóde (pozri obrázok 2.2). Na základe vybranej dvojice pozícií zistíme jej odpovedajúcu dvojicu časov (`dvojica_casy`) a zavolaním funkcie `Vypocitaj_priemer_cas()` (pozri 2.3.7) vypočítame priemer týchto časov, ktorý následne nastavíme obom robotom na pozíciách `dvojica_pozicie` ako ich nový čas. Tento postup sa vykonáva až do chvíle, keď budú mať všetky roboty nastavený rovnaký čas, čo sa overuje po každej synchronizácii časov dvoch robotov pomocou funkcie `SuVsetkyRovnake()` (pozri 2.3.3).

Ďalšou úlohou funkcie je počítať tzv. kroky (`k`), pričom jeden krok zahŕňa postup popísaný vyššie. Posledný krok v tejto funkcii teda znamená, že v tomto kroku došlo k synchronizácii časov všetkých robotov, čoho výsledkom je vektor dĺžky `pr`, ktorého všetky prvky majú rovnakú hodnotu `synch_hodnota`.

Počas vyhodnocovania funkcie `Synchronizuj()` je takisto možné ukladať grafy, a to v každom kroku `k` pomocou funkcie `UlozGraf()` (pozri 2.4.1). Aby vytvorené grafy odpovedali našim predstavám, obsahuje funkcia `Synchronizuj()` pred volaním funkcie `UlozGraf()` jeden `for` cyklus a za ním vytvorenie premennej `bliknutie`. Cyklus má na starosti posun ručičky na hodinkách o jeden časový dielik dopredu pre všetky roboty a premenná `bliknutie` je vektorom núl a jedničiek. Tá je potom súčasťou dát, ktoré sú vstupom k požadovanému vykresleniu grafu. Cyklus a premenná nám tak pomôžu k celkovej vizualizácii procesu synchronizácie robotov, pretože keď ručička dôjde na hodinkách na čas nula, všetky roboty, ktoré v danom kroku majú aktuálne čas nula, bliknú. Funkcia `UlozGraf()` je vo východiskovom nastavení zdrojového kódu zakomentovaná kvôli potrebe určitých

nastavení pred jej spustením. O vizualizácii a jej potrebných nastaveniach pojednáva kapitola 4.

Variant 2 a variant 3 má vo funkcii `Synchronizuj()` ešte niečo ďalšie a to je volanie funkcie `PosunVektor()`. Vysvetlenie tejto funkcie pozri v 2.3.12. Všetky tri varianty sa líšia tiež tým, že každý volá viac či menej odlišnú obmenu funkcie `UrcPozicie2Robotov()`, a to variant 1 funkciu `UrcPozicie2Robotov()` (pozri 2.3.4), variant 2 funkciu `UrcPozicie2Robotov_vzd()` (pozri 2.3.5) a variant 3 funkciu `UrcPozicie2Robotov_pst()` (pozri 2.3.6). Kvôli týmto odlišnostiam je i názov funkcie `Synchronizuj()` v zdrojovom kóde rozlíšený podľa daného variantu, a to na `SynchronizujVAR1()`, `SynchronizujVAR2()` a `SynchronizujVAR3()`. Definíciu funkcie pre variant 3 je možné vidieť na obrázku 2.5.

2.3.3. SuVsetkyRovnake()

Táto jednoduchá funkcia je volaná v rámci funkcie `Synchronizuj()` (pozri obrázok 2.5) a jej úlohou je pred každým ďalším krokom `k` skontrolovať aktuálny vektor s časmi robotov. V prípade, že časy v tomto vektore sú všetky rovnaké, vyhodnocovanie funkcie `Synchronizuj()` je ukončené a jej výstupom je vektor časov v danom kroku `k`. Došlo k synchronizácii časov všetkých robotov. Inak funkcia pokračuje stále ďalej.

Definícia funkcia je totožná pre všetky varianty zdrojového kódu (pozri obrázok 2.6).

2.3.4. UrcPozicie2Robotov()

Funkcia patrí k variante 1 a jej úlohou je náhodne vybrať naraz dve rozdielne pozície robotov z postupnosti čísel 1 až `pr`. Táto postupnosť je vytvorená v hlavnom kóde (pozri obrázok 2.2). Dvojica pozícií ako výstup funkcie `UrcPozicie2Robotov()` je ďalej použitá vo funkcii `Synchronizuj()` (pozri 2.3.2). Pozri obrázok 2.7 s definíciou funkcie `UrcPozicie2Robotov()`.

```
SynchronizujVAR3 <- function()
{
  k <<- 0
  sur_x <- x
  sur_y <- y

  vektor_casov <- roboty_casy

  while (!SuVsetkyRovnake(vektor_casov))
  {
    k <<- k+1

    dvojica_pozicie <- UrcPozicie2Robotov_pst(sur_x, sur_y)
    dvojica_casy <- vektor_casov[dvojica_pozicie]

    priemer <- Vypocitaj_priemer_cas(dvojica_casy)

    vektor_casov[dvojica_pozicie[1]] <- priemer
    vektor_casov[dvojica_pozicie[2]] <- priemer

    #print(k)
    #print(vektor_casov)

    for (s in 1:length(vektor_casov))
    {
      if (vektor_casov[s] == (pc - 1))
        vektor_casov[s] <- 0
      else
        vektor_casov[s] <- vektor_casov[s] + 1
    }

    #bliknutie <- 1*(vektor_casov == 0)
    #dataset <- data.frame(sur_x, sur_y, bliknutie)
    #UlozGraf(dataset, sur_x, sur_y, bliknutie, k)

    sur_x <- PosunVektor(sur_x)
    sur_y <- PosunVektor(sur_y)
  }
  #print(vektor_casov)
  return(list(vektor_casov=vektor_casov, sur_x=sur_x, sur_y=sur_y))
}
```

Obr. 2.5: Funkcia `SynchronizujVAR3()` vo variante 3

```
SuVsetkyRovnake <- function(roboty_casy)
{
  for (i in 1:(length(roboty_casy)-1))
  {
    if(roboty_casy[i] != roboty_casy[i+1])
      return(FALSE)
  }
  return(TRUE)
}
```

Obr. 2.6: Funkcia `SuVsetkyRovnake()`

```
UrcPozicie2Robotov <- function(pozicie_robotov)
{
  vybrane_pozicie <- sample(pozicie_robotov, 2, replace=FALSE)

  return(vybrane_pozicie)
}
```

Obr. 2.7: Funkcia `UrcPozicie2Robotov()` vo variante 1

2.3.5. UrcPozicie2Robotov_vzd()

Výstupom funkcie, ktorá sa nachádza vo variante 2, je, tak ako i u predchádzajúcej funkcie, dvojica rozličných pozícií robotov. Tá bude však k dispozícii až po získaní výstupu z volania funkcie `VyberRobota1()` (pozri 2.3.8), následne funkcie `Vypocitaj_vzdialenosti()` (pozri 2.3.9) a nakoniec funkcie `VyberRobota2_vzd()` (pozri 2.3.10). Dvojica pozícií je ďalej použitá vo funkcii `Synchronizuj()` (pozri 2.3.2). Definíciu funkcie `UrcPozicie2Robotov_vzd()` pozri na obrázku 2.8.

```
UrcPozicie2Robotov_vzd <- function(x, y)
{
  R1_pozicia <- VyberRobota1(roboty_pozicie)

  vzdialenosti_robotov <- Vypocitaj_vzdialenosti(x, y, R1_pozicia)

  R2_pozicia <- VyberRobota2_vzd(vzdialenosti_robotov, R1_pozicia)

  vybrane_pozicie <- c(R1_pozicia, R2_pozicia)

  return(vybrane_pozicie)
}
```

Obr. 2.8: Funkcia `UrcPozicie2Robotov_vzd()` vo variante 2

2.3.6. UrcPozicie2Robotov_pst()

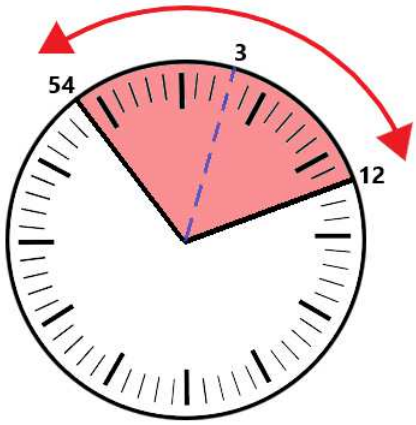
Funkcia má rovnaký výstup ako funkcia `UrcPozicie2Robotov_vzd()` (pozri 2.3.5) a je s ňou takmer identická. Rozdiel je len v tom, že nevolá funkciu `VyberRobota2_vzd()` (pozri 2.3.10), ale funkciu `VyberRobota2_pst()` (pozri 2.3.11). Nachádza sa vo variante 3 zdrojového kódu a jej definíciu možno vidieť na obrázku 2.9.

```
UrcPozicie2Robotov_pst <- function(x, y)
{
  R1_pozicia <- VyberRobota1(roboty_pozicie)

  vzdialenosti_robotov <- Vypocitaj_vzdialenosti(x, y, R1_pozicia)

  R2_pozicia <- VyberRobota2_pst(vzdialenosti_robotov, R1_pozicia)

  vybrane_pozicie <- c(R1_pozicia, R2_pozicia)

  return(vybrane_pozicie)
}
```

Obr. 2.9: Funkcia `UrcPozicie2Robotov_pst()` vo variante 3

2.3.7. Vypocitaj_priemer_cas()

Vo všetkých troch variantoch zdrojového kódu sa používa rovnaká funkcia `Vypocitaj_priemer_cas()`, ktorá nám zo vstupu v podobe dvojice časov, ktoré majú dva vybrané roboty, spočíta ich priemer. Avšak nie vždy ho počítame ako klasický aritmetický priemer dvoch hodnôt. Uveďme si názorný príklad. Na obrázku 2.10 sú vyznačené dva časy robotov: 54 a 12 (uvažujeme `pc = 60`, t. j. časový dieliky 0 až 59). Ich aritmetický priemer je 33. K tomuto času by teda oba časy robotov mali v rámci bieleho výseku ciferníka rovnakú vzdialenosť. Vzdialenosťou máme na mysli počet časových dielikov, tzn. od času 54 i od času 12 je to k času 33 rovnako presne 21 časových dielikov. No my chceme takúto zhodnú vzdialenosť zisťovať vždy v rámci toho výseku, ktorý je menší, v tomto prípade je ním červený výsek. Potrebujeme teda, aby nám funkcia `Vypocitaj_priemer_cas()` vrátila výsledný „priemer“ časov 54 a 12 rovný času 3, ku ktorému majú oba časy rovnakú vzdialenosť 9 časových dielikov.

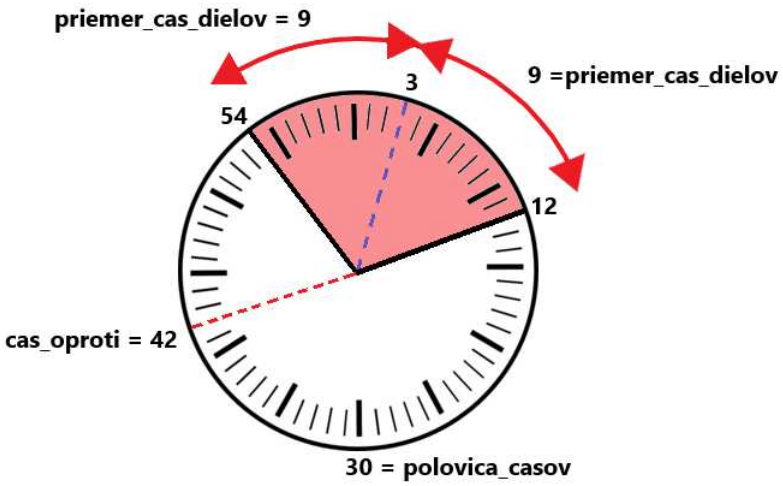


Obr. 2.10: Spôsob priemerovania dvoch časov

Túto funkcionalitu sa nám podarilo naprogramovať tak, aby fungovala v prípade akéhokoľvek `pc`. Využíva k tomu funkcie `min` a `max` a premenné (pozri obrázok 2.11):

- `pc` - aktuálne uvažované celkové množstvo časových dielikov,
- `polovica_casov` - polovičné množstvo z aktuálne uvažovaného `pc`. V prípade nepárneho `pc` pracuje funkcia `Vypocitaj_priemer_cas()` s desatinným číslom,
- `cas_oproti` - časový dielik/čas, ktorý sa nachádza presne naproti minima z dvojice časov robotov. Môže vyjsť i ako desatinné číslo,
- `priemer_cas_dielov` - polovica vzdialenosti (počet časových dielikov) od jedného času k druhému z aktuálnej dvojice časov robotov. Je to vzdialenosť k výslednému „priemeru“, ktorá je od oboch časov rovnaká.

K situácii, kedy k výpočtu výsledného priemeru dvoch rôznych časov nepoužijeme aritmetický priemer, dochádza v prípade, keď menší výsek ciferníka medzi týmito časmi obsahuje čas 0, prípadne jeden z časov je rovný 0. Nie je to však vždy, napr. pri `pc = 60` a pri dvojici časov `(0,14)` je samozrejme použitý



Obr. 2.11: Priemerovanie - premenné

aritmetický priemer. Ak by sme teda mali zhrnúť, aké podmienky musí dvojica časov splňovať, tak sú to:

- minimum dvoch časov je menšie ako `polovica_casov` a súčasne
- maximum je väčšie ako `cas_oproti`.

Vtedy buď od minima časov odčítame `priemer_cas_dielov`, alebo ho k maximu časov pričítame. Celú definíciu funkcie pozri na obrázku 2.12. Ešte poznamenajme, že vypočítaný priemer môže vyjsť ako desatinné číslo, preto je nakoniec vždy zaokrúhlený smerom nahor.

2.3.8. VyberRobota1()

Funkcia `VyberRobota1()` je súčasťou variantov 2 a 3. Jej návratová hodnota predstavuje náhodne vybranú pozíciu robota z postupnosti čísel 1 až `pr`. Robot odpovedajúci tejto pozícii bude jedným z dvoch robotov, ktoré budú v danom kroku funkcie `Synchronizuj()` (pozri 2.3.2) priemerovať svoje časy na hodinkách. Definíciu funkcie pozri na obrázku 2.13.

```
Vypocitaj_priemer_cas <- function(dvojica_casy)
{
  cas_oproti <- (min(dvojica_casy) + polovica_casov) %% pc
  priemer_cas_dielov <- (pc - max(dvojica_casy) + min(dvojica_casy))/2

  if (min(dvojica_casy) == max(dvojica_casy))
  {
    priem <- ceiling(dvojica_casy[1])
  }
  else if (min(dvojica_casy) >= polovica_casov)
  {
    priem <- ceiling(mean(dvojica_casy))
  }
  else
  {
    if (max(dvojica_casy) <= cas_oproti)
    {
      priem <- ceiling(mean(dvojica_casy))
    }
    else
    {
      if (priemer_cas_dielov <= min(dvojica_casy))
      {
        priem <- ceiling(min(dvojica_casy) - priemer_cas_dielov)
      }
      else
      {
        priem <- ceiling(max(dvojica_casy) + priemer_cas_dielov)
        if (priem == pc)
        {
          priem <- 0
        }
      }
    }
  }

  return(priem)
}
```

Obr. 2.12: Funkcia `Vypocitaj_priemer_cas()`

```
VyberRobota1 <- function(pozicie_robotov)
{
  vybrana_pozicia <- sample(pozicie_robotov, 1)

  return(vybrana_pozicia)
}
```

Obr. 2.13: Funkcia `VyberRobota1()` vo variante 2 a 3

2.3.9. Vypocitaj_vzdialenosti()

S touto funkciou sa stretneme vo variante 2 a 3, v ktorých sa vyberá náhodne najprv jeden robot a až potom ten druhý. A práve k výberu druhého robota potrebujeme predtým spočítať, akú vzdialenosť má každý jeden robot od robota získaného z funkcie `VyberRobota1()` (pozri 2.3.8). Roboty si predstavíme umiestnené na našej štvorcovej ploche ako body v rovine. Každý robot má svoje súradnice `[x,y]`, a tak vzdialenosť dvoch bodov v rovine vypočítame jednoducho ako dĺžku prepony pravouhlého trojuholníka. Výstupom funkcie je potom vektor vzdialeností dĺžky `pr - 1`. Definíciu funkcie pozri na obrázku 2.14.

```
Vypocitaj_vzdialenosti <- function(sur_x, sur_y, pozicia)
{
  vzd_x <- sur_x[pozicia] - sur_x[-pozicia]
  vzd_y <- sur_y[pozicia] - sur_y[-pozicia]

  vektor_vzdialenosti <- sqrt(vzd_x ^ (2) + vzd_y ^ (2))

  return(vektor_vzdialenosti)
}
```

Obr. 2.14: Funkcia `Vypocitaj_vzdialenosti()` vo variante 2 a 3

2.3.10. VyberRobota2_vzd()

Funkcia je obsiahnutá len vo variante 2. Vstupuje do nej vektor vzdialeností, ako výstup funkcie `Vypocitaj_vzdialenosti()` (pozri 2.3.9), ostatných robotov od robota vybraného pomocou funkcie `VyberRobota1()` (pozri 2.3.8). Úlohou funkcie je vybrať toho, ktorý má túto vzdialenosť najmenšiu. Funkcia vracia pozíciu tohto robota, pričom je zohľadnené to, akú pozíciu mal robot z funkcie `VyberRobota1()`, pretože je nežiaduce, aby získaná dvojica pozícií dvoch robotov, ako výstup funkcie `UrcPozicie2Robotov_vzd()` (pozri 2.3.5), obsahovala dvakrát tú istú pozíciu. Definíciu funkcie pozri na obrázku 2.15.

```
VyberRobota2_vzd <- function(vektor_vzdialenosti, vylucena_pozicia)
{
  vybrana_pozicia <- which.min(vektor_vzdialenosti)

  if (vybrana_pozicia >= vylucena_pozicia)
    vybrana_pozicia <- vybrana_pozicia + 1

  return(vybrana_pozicia)
}
```

Obr. 2.15: Funkcia `VyberRobota2_vzd()` vo variante 2

2.3.11. VyberRobota2_pst()

Ako bolo spomínané vyššie, do funkcií pre výber druhého robota do dvojice vstupuje vektor vzdialeností, ktorý je výsledkom funkcie `Vypocitaj_vzdialenosti()` (pozri 2.3.9). Všetky vzdialenosti tohto vektoru však chceme previesť na pravdepodobnosti stretnutia druhého robota s prvým, pričom by malo platiť, že robot s najmenšou vzdialenosťou k predtým vybranému robotu má túto pravdepodobnosť najväčšiu. Rozhodli sme sa transformovať vzdialenosti na tvar prevrátených hodnôt. Pravdepodobnosť stretnutia je teda nepriamoúmerná vzdialenosti. Ďalšími výpočtami dospejeme k vektoru pravdepodobností a následne k vektoru ich kumulatívnych súčtov. Vygenerovaním náhodného čísla z rovnomerného rozdelenia v intervale `(0,1)` a jeho postupným porovnávaním s prvkami vektora kumulatívnych súčtov nakoniec dosiahneme výslednú pozíciu druhého robota do hľadanej dvojice. Opäť je zohľadnená pozícia prvého robota z funkcie `VyberRobota1()` (pozri 2.3.8), aby získaná dvojica pozícií dvoch robotov, ako výstup funkcie `UrcPozicie2Robotov_pst()` (pozri 2.3.6), neobsahovala dvakrát tú istú pozíciu. Funkcia je súčasťou varianty 3 a jej definíciu možno vidieť na obrázku 2.16.

2.3.12. PosunVektor()

Funkcia vznikla v dôsledku potreby meniť polohy robotov na štvorcovej ploche, inak by v prípade variantu 2 a variantu 3 mohlo dôjsť k situácii, že by bol k prvému robotu z dvojice vyberaný opakovane rovnaký druhý robot. Napríklad vo variante 2 by pri nemenných súradniciach `[x,y]` mal k robotu, povedzme

```
VyberRobota2_pst <- function(vektor_vzdialenosti, vylucena_pozicia)
{
  prevratene <- 1/vektor_vzdialenosti
  vektor_pravdep <- (1/sum(prevratene))*prevratene
  kumulat_sucty <- rep(0,length(vektor_pravdep))

  nahodne_cislo <- runif(1)

  for (i in 1:length(vektor_pravdep))
  {
    if (i == 1)
      kumulat_sucty[i] <- vektor_pravdep[i]
    else
      kumulat_sucty[i] <- kumulat_sucty[i-1] + vektor_pravdep[i]
  }

  for (i in 1:length(kumulat_sucty))
  {
    if (nahodne_cislo <= kumulat_sucty[i])
    {
      vybrana_pozicia <- i
      break
    }
  }

  if (vybrana_pozicia >= vylucena_pozicia)
    vybrana_pozicia <- vybrana_pozicia + 1

  return(vybrana_pozicia)
}
```

Obr. 2.16: Funkcia `VyberRobota2_pst()` vo variante 3

s pozíciou 4, ktorý by bol vybraný náhodne ako prvý z dvojice, najmenšiu vzdialenosť robot, povedzme s pozíciou 9. A preto, ak by bol v akomkoľvek ďalšom kroku procesu synchronizácie vybraný robot s pozíciou 4, tak podľa najmenšej vzdialenosti k nemu by bol za robota do dvojice zakaždým zvolený robot s pozíciou 9. A takto by si mohli priemerovať svoje časy dookola tie isté dvojice, prípadne roboty v rámci nejakého zhluku, ale už nie roboty z dvoch rôznych zhlukov. Preto by nemuselo vôbec dôjsť k synchronizácii všetkých robotov na rovnaký čas, ale len k synchronizácii zhlukov alebo dvojíc.

Posun robotov prebieha tak, že k prvkom vstupného vektora `x`-ových, resp. `y`-ových súradníc je pripočítaná náhodne vygenerovaná hodnota z normálneho rozdelenia so strednou hodnotou 0 a smerodajnou odchýlkou nastavenou v premennej `smer_odch`. Zároveň je zabránené tomu, aby nové súradnice robotov prekročili hranice našej štvorcovej plochy (grafu). Definíciu funkcie pozri na obrázku 2.17.

```
PosunVektor <- function(v)
{
  dv <- rnorm(n=pr,mean=0,sd=smer_odch)

  pom_v <- ifelse((v+dv >= 1) | (v+dv <= 0), v-dv, v+dv)

  return(pom_v)
}
```

Obr. 2.17: Funkcia `PosunVektor()` vo variante 2 a 3

2.4. Doplnkové funkcie pre ukladanie grafov

Funkcie, ktoré sme popísali v podkapitole 2.3, sú nevyhnutné pre žiadaný priebeh simulácie. V tejto podkapitole však vysvetlíme ešte zopár ďalších, ktoré ale k samotnému procesu synchronizácie nie sú potrebné. Ich úlohou je vykresliť grafy a následne ich uložiť. Pred spustením simulácie spoločne s priebežným ukladaním grafov je nutné uskutočniť niekoľko krokov, o ktorých pojednáva kapitola 4.

2.4.1. UlozGraf()

Táto funkcia je volaná vnútri funkcie `Synchronizuj()` (pozri 2.3.2) a v každom jej kroku slúži k uloženiu grafu. Graf znázorňuje rozmiestnenie a svietenie/blikanie robotov na štvorcovej ploche práve v danom kroku procesu synchronizácie a má pripomínať nočné blikanie svätojánskych mušiek, ktoré bolo jedným z príkladov uvedených v kapitole 1.1. Uložené grafy je potom možné využiť k vytvoreniu vizualizácie procesu synchronizácie robotov (pozri kapitolu 4).

Na začiatku funkcie `UlozGraf()` (pozri obrázok 2.18) je nastavená cesta k zložke (v našom kóde je to `C:\SYNC`), kam chceme grafy ukladať. Toto nastavenie je samozrejme možné zmeniť. Dôležité však je, aby daná zložka pred spustením simulácie už v danom umiestnení existovala. Funkcia totiž nevykonáva jej samotné vytvorenie, vytvára iba súbory `.png`. Názvy všetkých súborov sú tvorené pomocou funkcie `GenerujNazov()` (pozri 2.4.4).

K vytvoreniu grafu je použitá funkcia `ggplot`, ktorá v našom prípade vykreslí bodový graf (z datasetu, ktorý vznikne v aktuálnom kroku procesu synchronizácie), kde na osi `x` budú `x`-ové a na osi `y` `y`-ové súradnice robotov. Keďže chceme, aby roboty (body grafu) v konečnom dôsledku predvádzali blikanie, je ako faktor použitý vektor núl a jedničiek, ktorý sa tiež zakaždým vytvára v aktuálnom kroku funkcie `Synchronizuj()`. Ak robotu so súradnicami `[x,y]` odpovedá nula v tomto vektore, robot nesvieti a na grafe tak splýva s pozadím. Ak mu odpovedá jednička, robot svieti a na grafe je zobrazený žltou farbou. Ďalej sú v rámci funkcie `ggplot` použité nastavenia pre obmedzenie rozsahu osí, farbu pozadia grafu, názov grafu, odstránenie názvov osí a legendy.

Funkcia `ggplot` sa nachádza v podpornom balíčku `ggplot2` [18], ktorý je potrebné mať nainštalovaný. Na začiatku zdrojového kódu každej varianty je príkaz `library(ggplot2)`, ktorý pred spustením simulácie balíček otvorí a načíta. Tento príkaz, takisto ako aj volanie funkcie `UlozGraf()` vo funkcii `Synchronizuj()`, treba v kóde najprv odkomentovať.

Ukladanie grafov má zmysel povoliť v prípade jedného behu procesu synchronizácie pre jednu ľubovoľnú dvojicu `(pc,pr)`. *Je teda vhodné nastaviť premennú*

`pocet_behov = 1` *a premenné* `pocet_casov` *a* `pocet_robotov` *ako vektory dĺžky 1. Pokiaľ tak nebude, grafy ukladané ako súbory vo formáte* `.png` *budú v mieste ukladania navzájom prepisované, keďže proces synchronizácie bude prebiehať viac ako raz, no názvy súborov budú rovnaké. Kapitola 4 obsahuje návod so všetkými potrebnými nastaveniami pre správne fungovanie ukladania grafov.*

```
UlozGraf <- function(data_grafu, x, y, faktor, krok)
{
  cesta <- file.path("C:","SYNC", GenerujNazov(krok))
  png(file = cesta)
  nazov_grafu <- paste("Krok ", krok)
  print(ggplot(data = data_grafu, aes(x = x, y = y)) +
    geom_point(aes(colour = factor(faktor))) +
    xlim(0, 1) + ylim(0, 1) +
    scale_colour_manual(values = c("gray14", "yellow2")) +
    ggtitle(nazov_grafu) +
    theme(panel.background = element_rect(fill = 'gray14', colour = 'gray14'),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          plot.background = element_rect(fill = "gray80"),
          axis.title = element_blank(),
          axis.ticks = element_blank(),
          axis.text = element_blank(),
          legend.position="none"))

  dev.off()
}
```

Obr. 2.18: Funkcia `UlozGraf()`

2.4.2. UlozSynchGraf()

Nasledujúca funkcia vytvára a ukladá graf rovnako ako funkcia `UlozGraf()` (pozri 2.4.1), je však použitá až v prípade, kedy sú už všetky časy robotov zosynchronizované na ten istý čas, a teda na grafe buď všetky roboty svietia žltou farbou, alebo nesvietia žiadne, a teda splývajú s pozadím grafu.

Volanie funkcie `UlozSynchGraf()` prebieha vo funkcii `UlozGrafy_poSynch()` (pozri 2.4.3).

2.4.3. UlozGrafy_poSynch()

Funkcia je obsiahnutá vo funkcii `SynchPreDvojicuPrAPc()` (pozri 2.3.1) v každej variante a pracuje už so synchronizovanými časmi všetkých robotov. Má za úlohu v každom kroku cyklu posunúť všetkým robotom ručičku na hodinkách o jeden časový dielik dopredu a uložiť graf pomocou funkcie `UlozSynchGraf()` (pozri 2.4.2). Koľko krokov bude mať tento cyklus záleží na nastavených premenných `pc` a `pocet_ulozeni` (pozri v 2.2.4). Dôvodom k tomu je, aby sme získali grafy, ktoré znázorňujú blikanie robotov po úspešnom zosynchonizovaní všetkých ich časov na jeden spoločný čas. Prípadná vizualizácia procesu synchronizácie (pozri kapitolu 4) tak neskončí v okamihu, kedy prvýkrát bliknú všetky roboty naraz, ale bliknutie sa zopakuje ešte niekoľkokrát na základe premennej `pocet_ulozeni`. Definíciu funkcie pozri na obrázku 2.19).

```
UlozGrafy_poSynch <- function(posl_krok, synch_casy, sur_x, sur_y)
{
  for (i in 1:(pocet_ulozeni*pc))
  {
    #print(i+posl_krok)
    #print(synch_casy)

    if (synch_casy[1] == (pc - 1))
      synch_casy <- rep(0,length(synch_casy))
    else
      synch_casy <- synch_casy + 1

    bliknutie_synch <- rep(0, length(synch_casy))

    if (synch_casy[1] == 0)
      bliknutie_synch <- rep(1, length(synch_casy))

    dataset_synch <- data.frame(sur_x, sur_y, bliknutie_synch)

    UlozSynchGraf(dataset_synch, sur_x, sur_y, bliknutie_synch, i+posl_krok)

    sur_x <- PosunVektor(sur_x)
    sur_y <- PosunVektor(sur_y)
  }
}
```

Obr. 2.19: Funkcia `UlozGrafy_poSynch()` vo variante 2 a 3

2.4.4. GenerujNazov()

Úlohou funkcie je vytvoriť názov pre súbor, ktorý je vo formáte `.png` ukladaný do zložky nastavenej vo funkcii `UlozGraf()` (pozri 2.4.1). Je totiž potrebné, aby pre prípadnú vizualizáciu (pozri kapitolu 4) procesu synchronizácie mali názvy súborov poradie odpovedajúce krokom procesu synchronizácie.

Funkcia je súčasťou funkcie `UlozGraf()` a `UlozSynchGraf()` (pozri 2.4.2) a jej definícia je na obrázku 2.20.

```
GenerujNazov <- function(krok)
{
  if (krok < 10) {nazov <- paste('krok_0000',krok,'.png',sep='')}
  else if (krok < 100) {nazov <- paste('krok_000',krok,'.png', sep='')}
  else if (krok < 1000) {nazov <- paste('krok_00', krok,'.png', sep='')}
  else if (krok < 10000) {nazov <- paste('krok_0', krok,'.png', sep='')}
  else {nazov <- paste('krok_', krok,'.png', sep='')}

  return(nazov)
}
```

Obr. 2.20: Funkcia `GenerujNazov()`

Kapitola 3

Analýza výsledkov simulácie

V tejto kapitole využijeme naprogramovanú simuláciu procesu synchronizácie robotov, ktorú spustíme pre rôzne hodnoty vstupných premenných `pocet_casov` a `pocet_robotov`. Jej výsledky budeme následne analyzovať a prezentovať pomocou grafov typu barplot, boxplot a histogram.

V zadaní úlohy simulácie v podkapitole 2.1 sme poznamenali, že všetky časy/časové dieliky, ktoré môže mať robot na svojich hodinkách, sú rovnako pravdepodobné. Predpokladáme teda, že pokiaľ pre vybranú dvojicu `(pc,pr)` spustíme proces synchronizácie dostatočne veľakrát, náhodná veličina čas, predstavujúca čas/časový dielik, na ktorý sa všetky roboty v procese zosynchronizovali, by mala mať rovnomerné rozdelenie. Overiť tento predpoklad sme skúsili v nasledujúcej simulácii.

V simulácii sme uvažovali 12 časov/časových dielikov `(pc)` a rôzne množstvá robotov `(pr)`, konkrétne 10, 130, 250, 500, 750 a 1000, vstupujúcich do procesu synchronizácie. Pre každú dvojicu `(pc,pr)` sme proces synchronizácie spustili 1000-krát. Pri predpoklade rovnomerného rozdelenia a pri nastavení premennej `pocet_behov = 1000` by teda mal byť každý z 12 časov/časových dielikov zastúpený približne 83-krát v tisícke výsledkov každej dvojice `(pc,pr)`. Toto nastavenie vstupných premenných sme použili v každom variante zdrojového kódu simulácie a vidieť ho možno na obrázku 3.1. V prípade variantu 2 a 3 bola nastavená smerodajná odchýlka na hodnotu `0.05`.

Výsledky simulácií zobrazujú tri nasledujúce obrázky. Každý obrázok odpo-

```
pocet_behov <- 1000
pocet_casov <- c(12)
pocet_robotov <- c(10,130,250,500,750,1000)
smer_odch <- 0.05
```

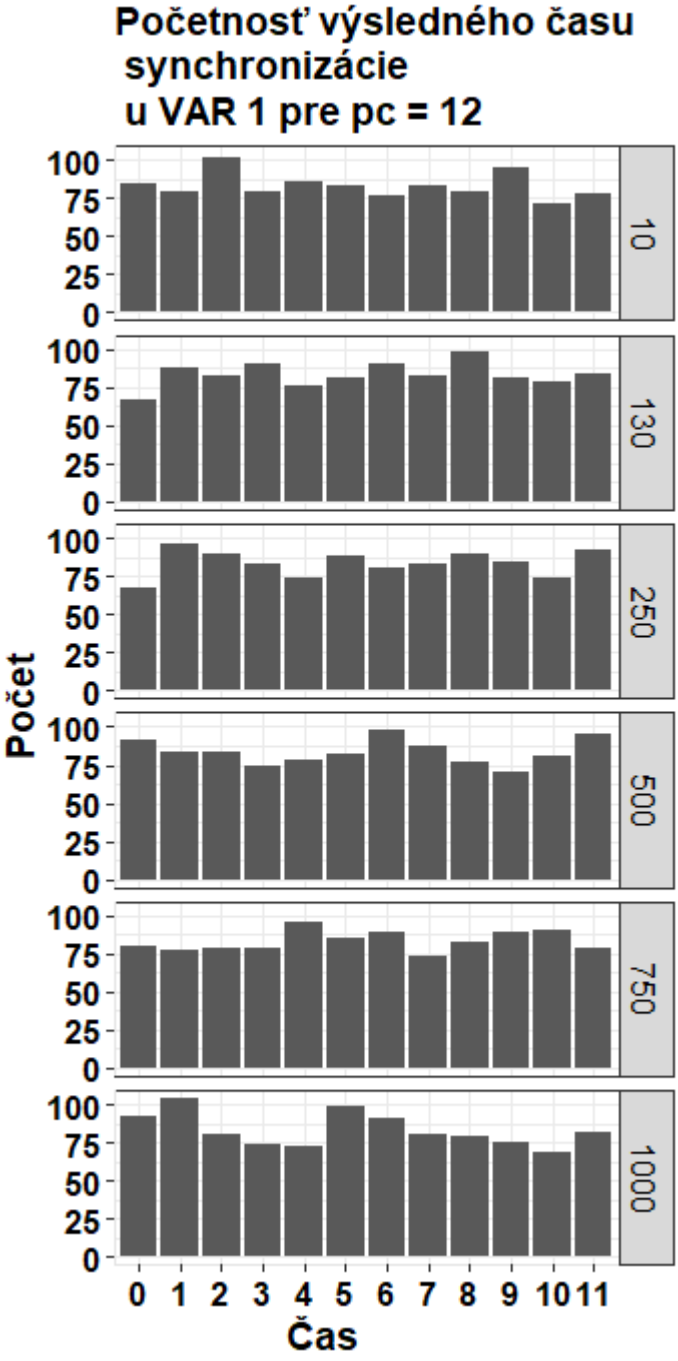
Obr. 3.1: Nastavenie vstupných premmených s `pc = 12`

vedá jednému variantu zdrojového kódu simulácie (pre variant 1 pozri obrázok 3.2, pre variant 2 obrázok 3.3 a pre variant 3 obrázok 3.4) a každý z nich obsahuje 6 grafov typu barplot, jeden pre každú dvojicu `(pc,pr)`. Vo všetkých prípadoch môžeme vidieť, že časy 0 až 11 sú zastúpené viac či menej blízko počtu 83, a tak rozdelenie náhodnej veličiny čas pripomína rovnomerné rozdelenie. Ešte presnejších výsledkov by bolo zrejme možné dosiahnuť zvýšením hodnoty vo vstupnej premennej `pocet_behov`.

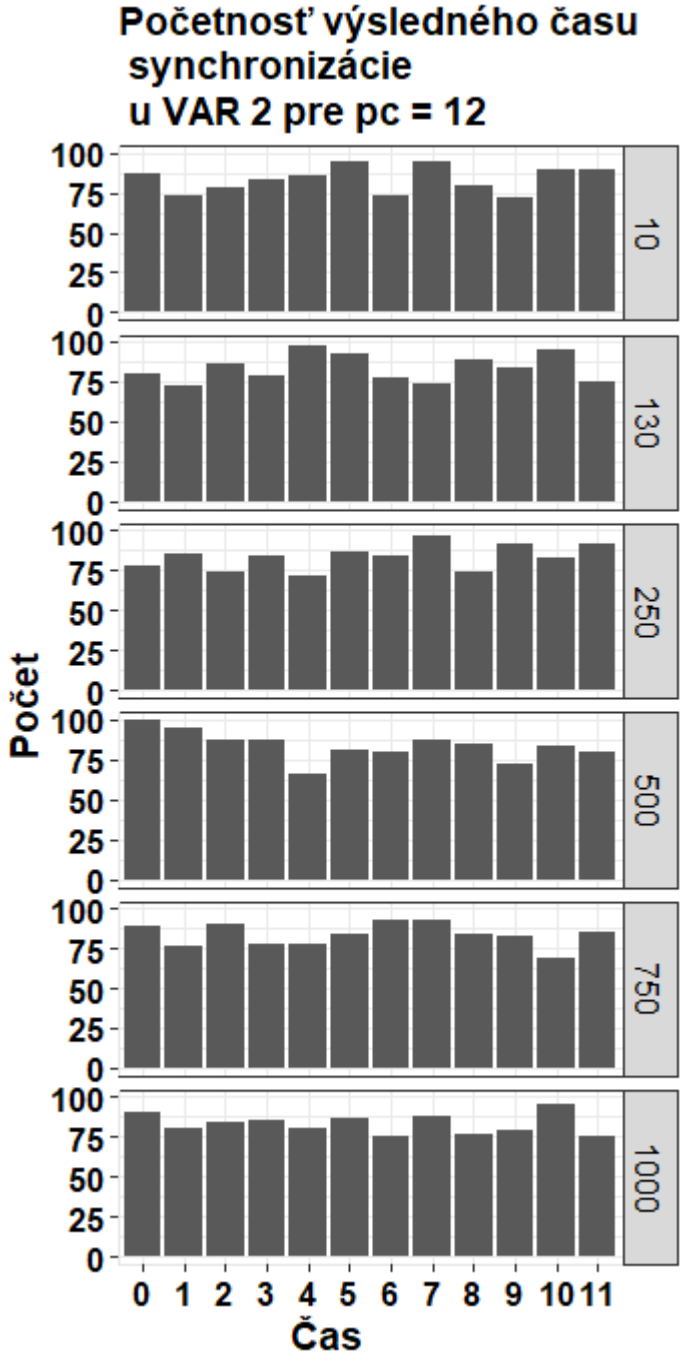
3.1. Závislosť doby synchronizácie na počte robotov

V nasledujúcej podkapitole sa pozrieme na ďalšie výsledky predošlých simulácií, kedy nastavenie vstupných premenných odpovedalo tomu na obrázku 3.1. Zvolili sme si teda jeden počet časov, a to `pc = 12`, ktorý budeme používať pri rôznych počtoch robotov `pr`, ktoré sú vlastne hodnotami vo vektore `pocet_robotov`. Nastavenie vstupných premenných bolo opäť pre všetky tri varianty zdrojového kódu rovnaké, môžeme tak nielen sledovať výsledky každého variantu zvlášť, ale aj porovnať varianty medzi sebou. Bude nás zaujímať predovšetkým to, ako sa bude meniť trvanie procesu synchronizácie pri rôznom množstve robotov, či už v podobe počtu krokov, ktorý bol potrebný k úspešnej synchronizácii všetkých robotov, alebo v podobe doby v sekundách.

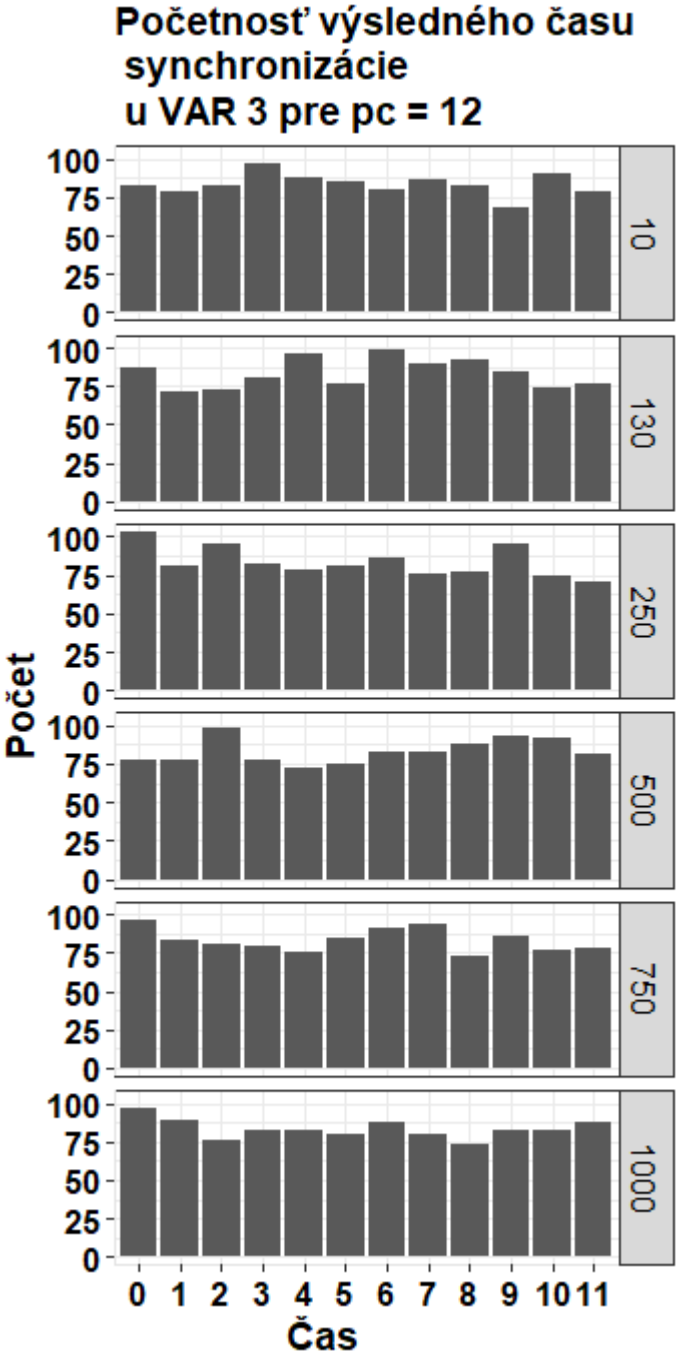
Na obrázku 3.5 vidíme grafy typu boxplot znázorňujúce rozsah výsledných počtov krokov, ktoré proces synchronizácie dosiahol pre dané `pr` pri nemennej hodnote `pc = 12`. Obrátený trojuholník v boxplote predstavuje aritmetický priemer hodnôt. Z grafov vidíme, že pre všetky tri varianty sa so zvyšujúcim sa počtom robotov zvyšuje i počet krokov potrebných k úspešnému zosynchronizo-



Obr. 3.2: Variant 1 - početnosť výsledného času synchronizácie



Obr. 3.3: Variant 2 - početnosť výsledného času synchronizácie

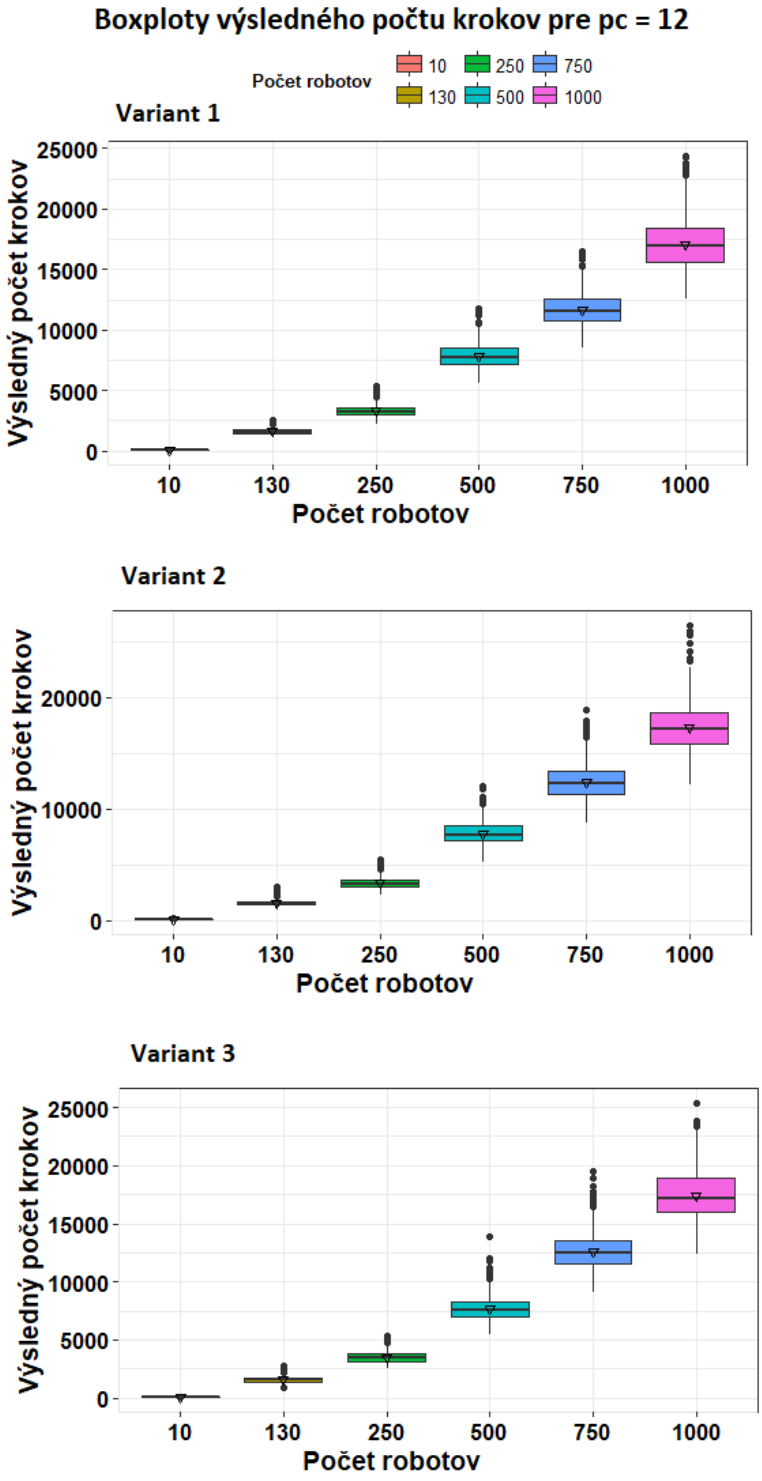


Obr. 3.4: Variant 3 - početnosť výsledného času synchronizácie

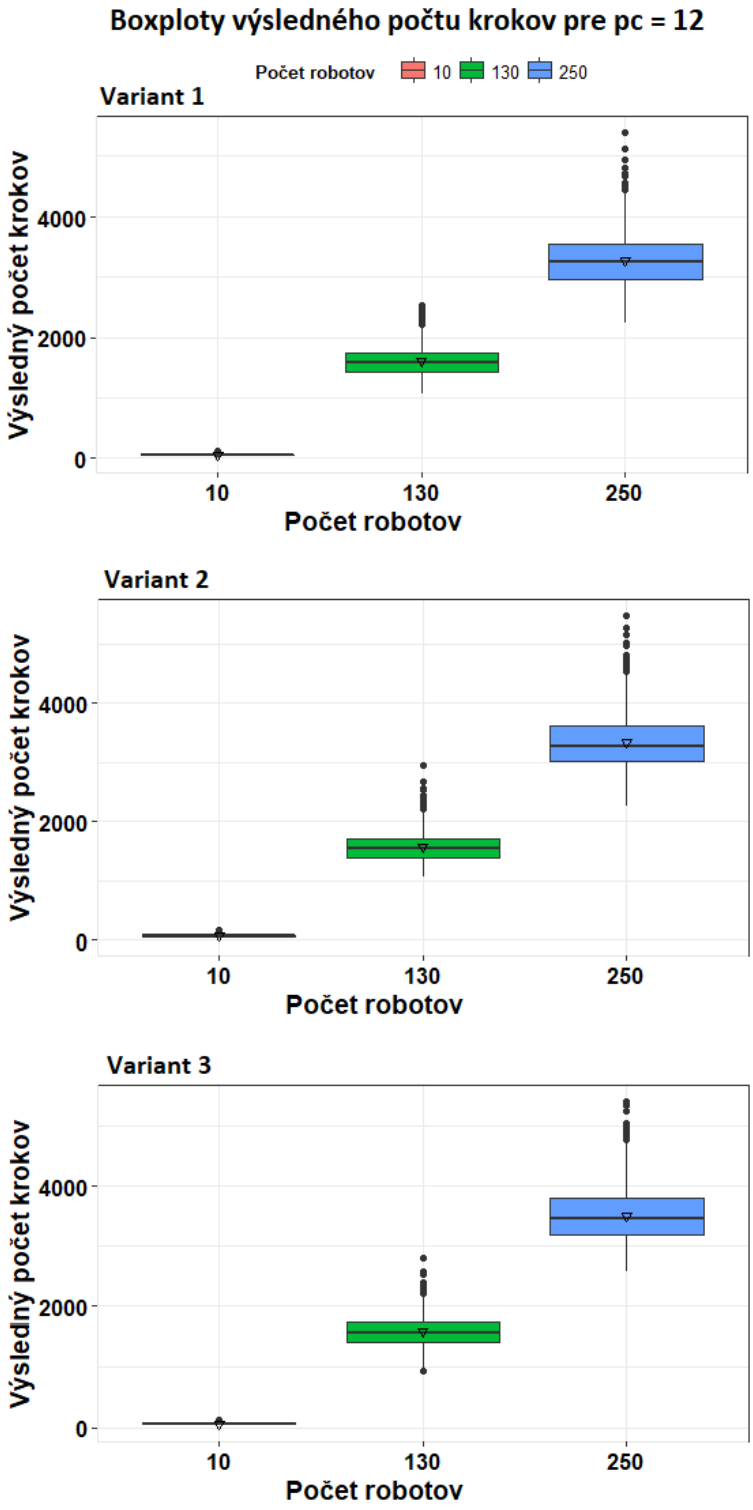
vaniu časov všetkých `pr` robotov. Dokonca v častiach, kde sa konštantne zvyšuje počet robotov, t. j. o 120 robotov medzi prvými troma hodnotami `pr` a o 250 robotov medzi druhými troma, resp. poslednými štyrmi hodnotami `pr`, by mohlo ísť o lineárnu závislosť. Tieto časti sú bližšie zobrazené na obrázku 3.6 a 3.7. Porovnaním variantov navzájom vzhľadom k niektorej z hodnôt počtu robotov vidíme, že ich výsledky sú dosť podobné. Na základe týchto výsledkov by zrejme bolo možné odhadnúť, koľko približne krokov bude mať proces synchronizácie. Napríklad pri `pr = 625` a `pc = 12` by sme mohli odhadnúť výsledný počet krokov okolo hodnoty 10 000 u každého variantu.

Na grafoch si môžeme tiež všimnúť, že u každého variantu sa nachádzajú odľahlé hodnoty, ale iba v hornej časti, proces synchronizácie vtedy skončil úspešne po väčšom počte krokov než obvykle. Čo sa však týka symetrie, vyzerá to vo všetkých prípadoch na približne symetrické rozdelenie, o čom svedčí ako medián nachádzajúci sa blízko stredu boxu, tak i dĺžky fúzov, ktoré sú približne rovnako dlhé, i to, že hodnota mediánu a hodnota aritmetického priemeru sa prekrývajú, a teda sú si veľmi podobné, ak nie zhodné. Približnú symetriu možno vidieť i na ďalších priložených obrázkoch obsahujúcich grafy typu histogram, pozri obrázok 3.8 a 3.9. Z nich, ale i z predchádzajúcich boxplotov je zrejmé, že v prípade `pr = 10` veľa behov procesu synchronizácie skončilo s veľmi podobným výsledným počtom krokov, oproti prípadom, kedy je počet robotov väčší, a tak je väčší aj rozptyl hodnôt pre výsledný počet krokov.

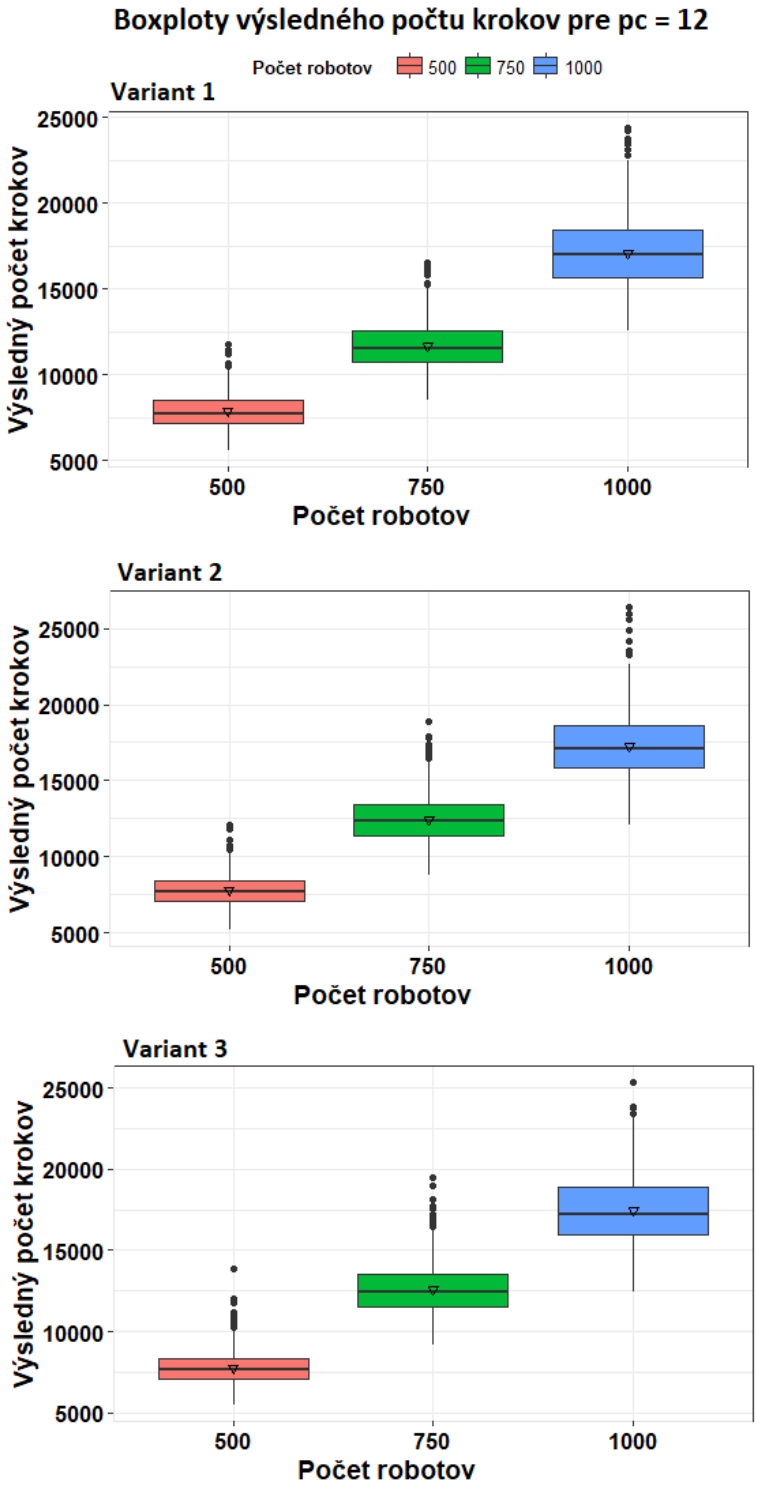
Ďalej sa pozrieme na dobu synchronizácie vyjadrenú v sekundách. Na obrázku 3.10 máme opäť grafy typu boxplot znázorňujúce rozsah trvania doby synchronizácie v sekundách pre dané `pr` pri nemennej hodnote `pc = 12`. Aritmetický priemer hodnôt je zobrazený vo forme obráteného trojuholníka. Na tomto obrázku však upozorňujeme na rozdielny rozsah osi y vykreslených grafov, ktorých výsledky môžu vyzerať sprvu veľmi podobne. Výsledky variantu 1 sa pohybujú od tých najnižších hodnôt blízkych nule pre `pr = 10` až po hodnotu 5 sekúnd pre `pr = 1000`. Vo variante 2 je to však až nad hodnotu 15 sekúnd pre `pr = 1000` a vo variante 3 po hodnotu 20 sekúnd pre `pr = 1000`, resp. až nad hodnotu 25 sekúnd



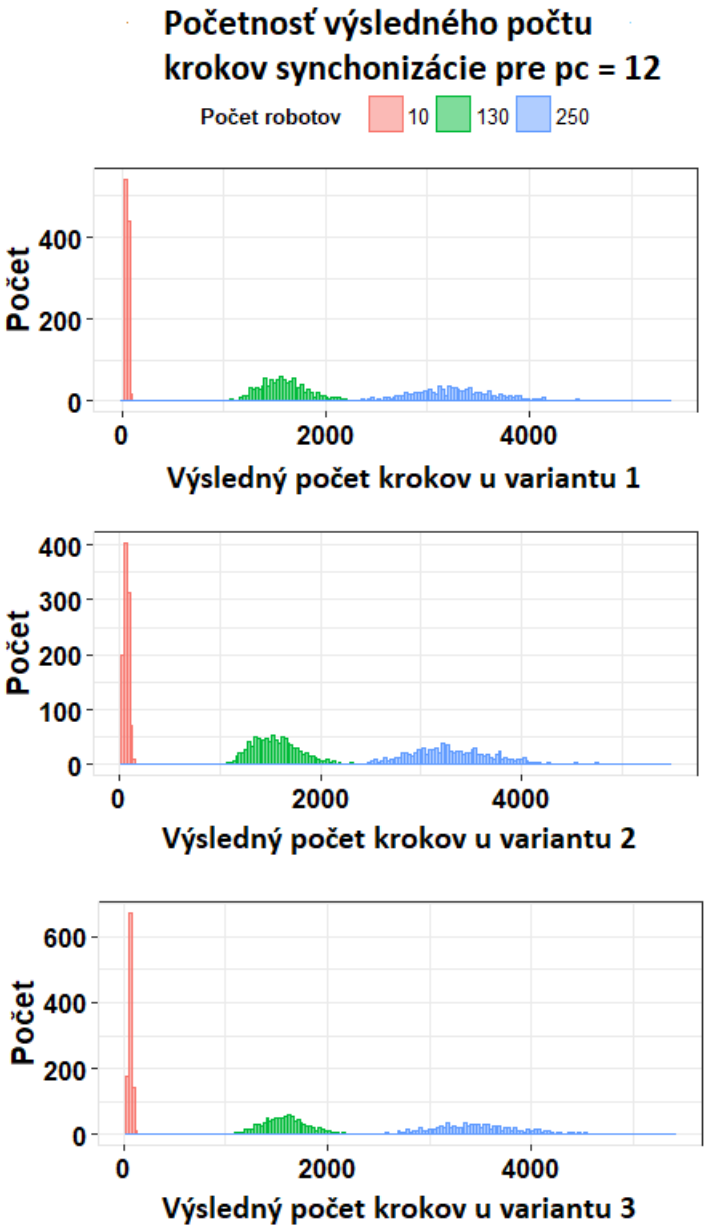
Obr. 3.5: Počet krokov pre každú hodnotu `pr` - boxplot



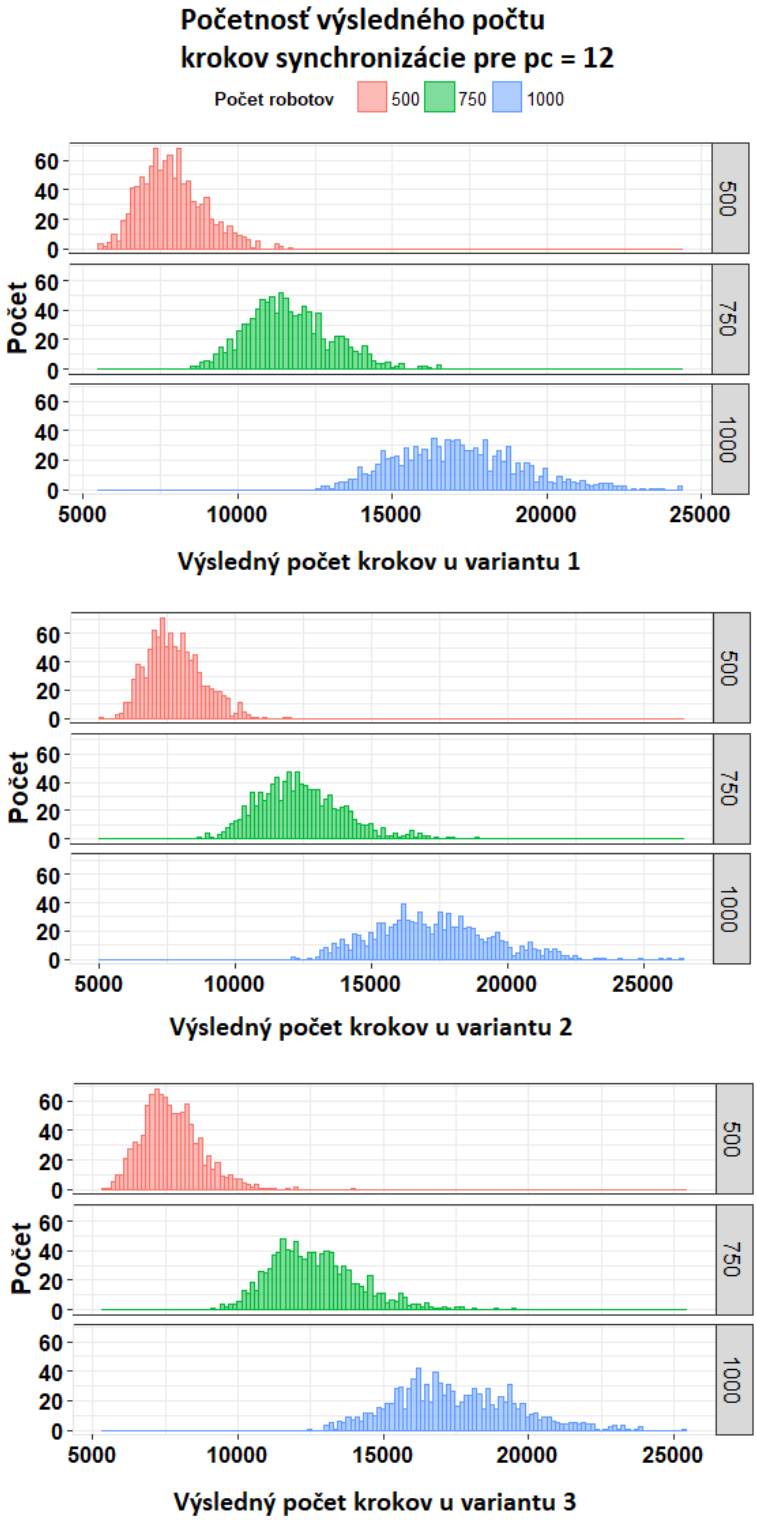
Obr. 3.6: Počet krokov pre prvé tri hodnoty `pr` - boxplot



Obr. 3.7: Počet krokov pre posledné tri hodnoty `pr` - boxplot



Obr. 3.8: Počet krokov pre prvé tri hodnoty `pr` - histogram



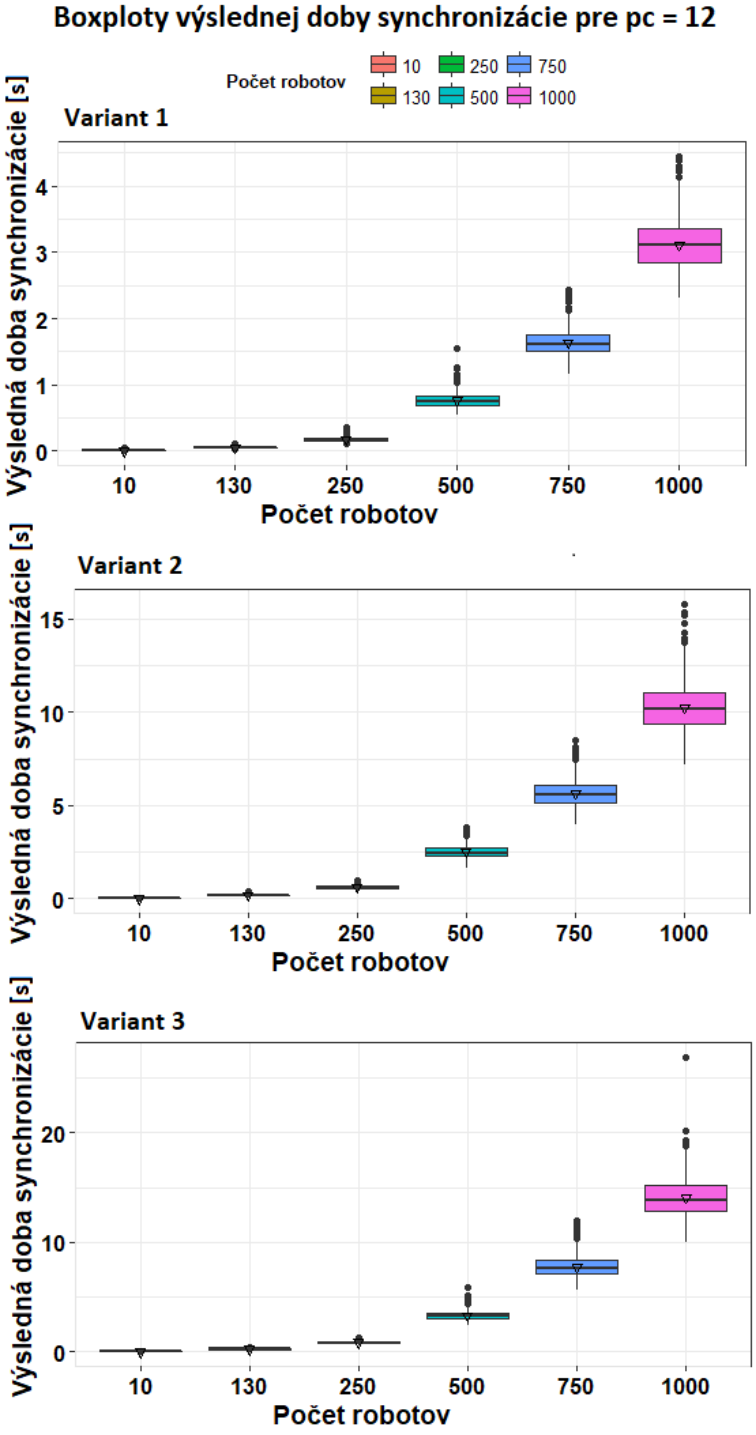
Obr. 3.9: Počet krokov pre posledné tri hodnoty `pr` - histogram

v prípade najväčšej odľahlej hodnoty. Z grafov je viditeľné, že u každého variantu sa pri zvyšujúcom počte robotov zvyšuje v sekundách vyjadrená výsledná doba synchronizácie procesu robotov. Varianty však majú pre daný počet robotov výsledky zreteľne rozdielne, čo v prípade doby meranej počtom krokov tak nebolo. Výsledky je lepšie vidieť na obrázku 3.11 pre prvé tri hodnoty `pr` a 3.12 pre posledné tri hodnoty `pr`. „Skoky“ medzi jednotlivými hodnotami `pr` variantu 1 nie sú tak výrazné ako u ostatných variantov. Variant 1 je teda zjavne najrýchlejší oproti zvyšným dvom variantom a to pri akomkoľvek počte robotov. Tento výsledok sa dal aj predpokladať, ak sa zamyslíme nad zložitosťou zdrojového kódu variantu 1 oproti variantu 2 alebo 3. Variant 1 nevykonáva toľko potrebných výpočtov ako variant 2 či 3, pretože ako vieme, výber dvojice robotov, ktoré budú synchronizovať svoje časy je vykonaný hneď bez nejakých pravidiel pre výber druhého robota do dvojice (pozri popis variantov v kapitole 2.2). Najväčšie „skoky“ medzi hodnotami `pr` môžeme pozorovať u variantu 3 a to najmä medzi hodnotou 130 robotov a 250 robotov. Zdá sa teda, že so zvyšujúcim sa počtom robotov sa bude doba synchronizácie variantu 3 zvyšovať najrýchlejšie.

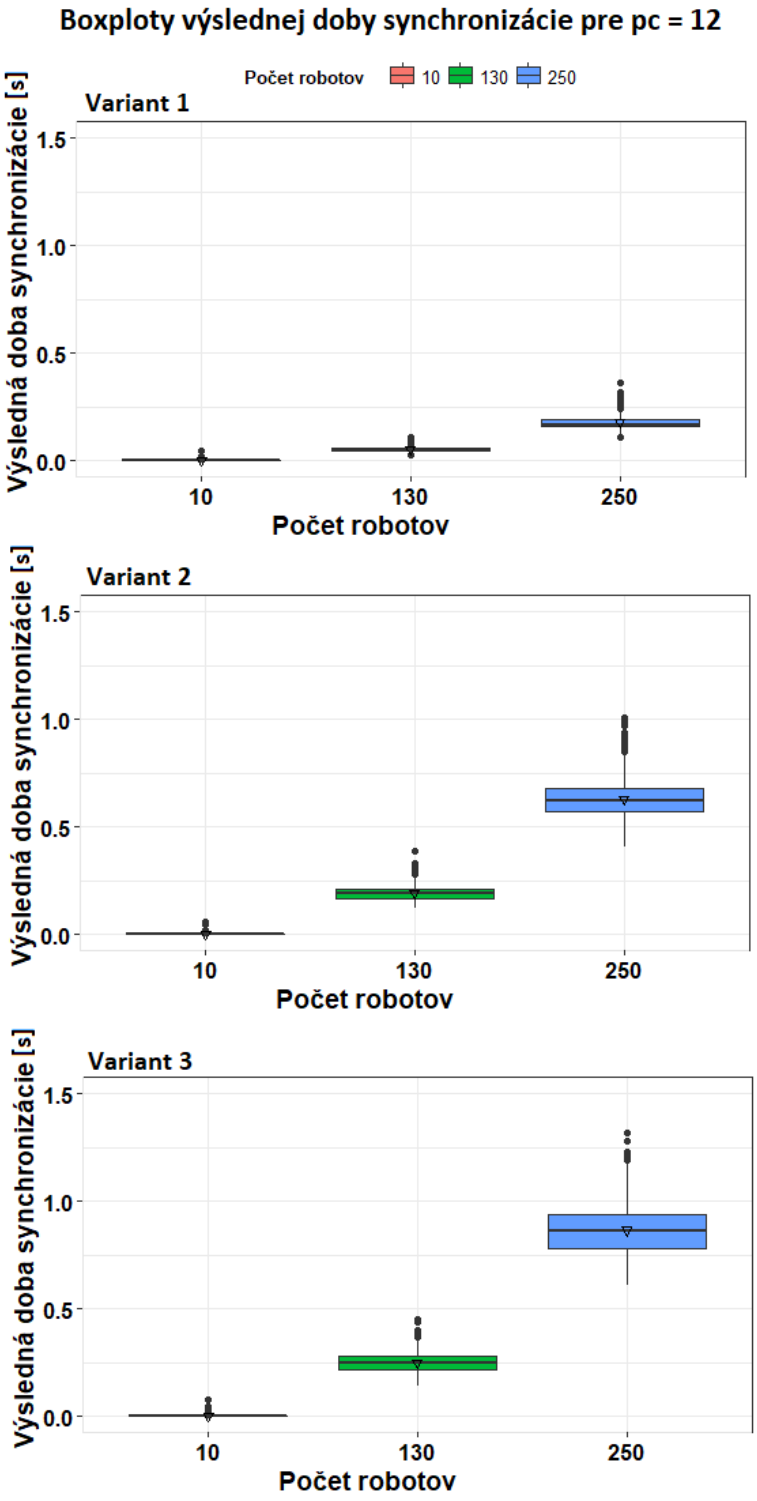
3.2. Závislosť doby synchronizácie na počte časov

K skúmaniu závislosti doby synchronizácie na počte časov/časových dielikov si zvolíme jednu hodnotu `pr` (počet robotov) a túto hodnotu budeme používať pri rôznych počtoch časov, ktoré predstavujú hodnoty vo vektore `pocet_casov`. Rovnaké nastavenie vstupných premenných pripravíme u každého z troch variantov zdrojového kódu, čo nám umožní navzájom porovnávať výsledky medzi variantmi. Znovu budeme sledovať predovšetkým to, ako sa bude meniť trvanie procesu synchronizácie pri rôznom počte časov a to v podobe počtu krokov a počtu sekúnd.

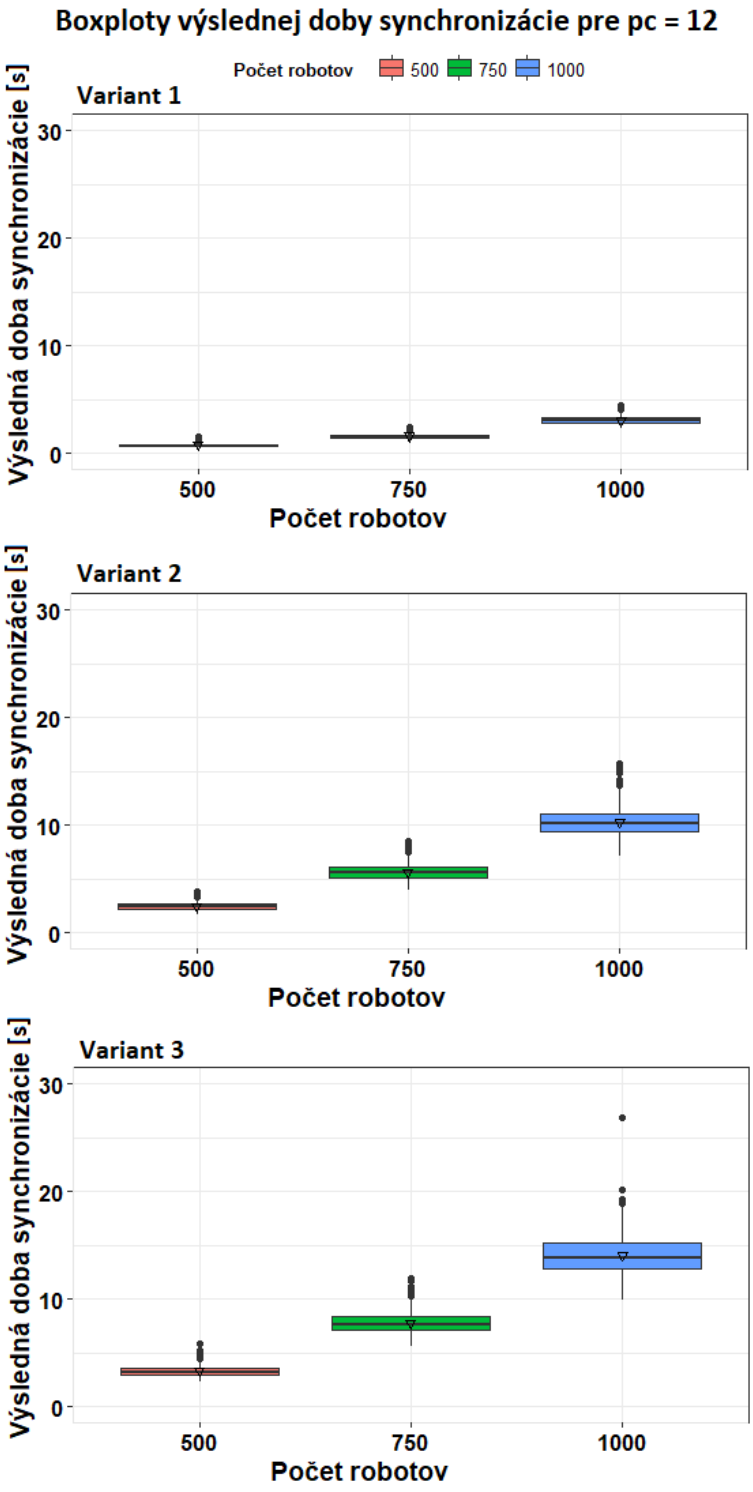
K simulácii sme zvolili 500 robotov `(pr)` a tieto rôzne hodnoty počtu časov `(pc)`: 10, 130, 250, 500, 750 a 1000. Pre každú dvojicu `(pc,pr)` sme proces synchronizácie spustili 1000-krát. V prípade variantu 2 a 3 bola nastavená sme-



Obr. 3.10: Doba synchronizácie pre každú hodnotu `pr` - boxplot



Obr. 3.11: Doba synchronizácie pre prvé tri hodnoty `pr` - boxplot



Obr. 3.12: Doba synchronizácie pre posledné tri hodnoty `pr` - boxplot

rodajná odchýlka na hodnotu `0.05`. Celé nastavenie možno vidieť na obrázku 3.13.

```
pocet_behov <- 1000
pocet_casov <- c(10,60,110,200,300,400)
pocet_robotov <- c(500)
smer_odch <- 0.05
```

Obr. 3.13: Nastavenie vstupných premmených s `pr = 500`

Na obrázku 3.14 môžeme pomocou boxplotov vidieť, ako sa menil výsledný počet krokov pri rôznych počtoch časov. Väčšinou pri zvýšení počtu časov sa zvýši i počet krokov u všetkých variant, ale u variantu 1 i variantu 2 si môžeme všimnúť, že napríklad v prípade zvýšenia zo 110 na 200 časov u variantu 1 alebo z 200 na 300 časov u variantu 2 je tento posun nahor až nepatrný. U variantu 1 sa to dokonca pri prechode z 300 na 400 časov javí ako malé (zrejme nevýznamné) zníženie počtu krokov. Naproti tomu u variantu 3 vidíme pomerne lineárne rastúci počet krokov. Je potrebné si však uvedomiť, že počet časov bol v prvej trojici hodnôt zvyšovaný o 50 časov, kým v druhej trojici to bolo o 100 časov. U všetkých troch variantov však platí, že s viac či menej rastúcou tendenciou sa všetky boxy, predstavujúce 50 % pozorovaní pre danú dvojicu `(pc,pr)`, nachádzajú v rozmedzí, ktoré začína približne na počte 7 000 krokov (pre `pc = 10`) a siaha k počtu približne 11 000 krokov (pre `pc = 400`).

Vzhľadom k symetrii usudzujeme, že vo všetkých prípadoch ide o približne symetrické rozdelenie a odľahlé hodnoty sa v drvivej väčšine vyskytujú v hornej časti. Ak tieto odľahlé hodnoty neberieme do úvahy, fúzy jednotlivých boxplotov sú dĺžkou veľmi podobné, takisto ako aj zobrazený medián a aritmetický priemer. Ilustratívne sú aj histogramy nasledujúce na obrázku 3.15.

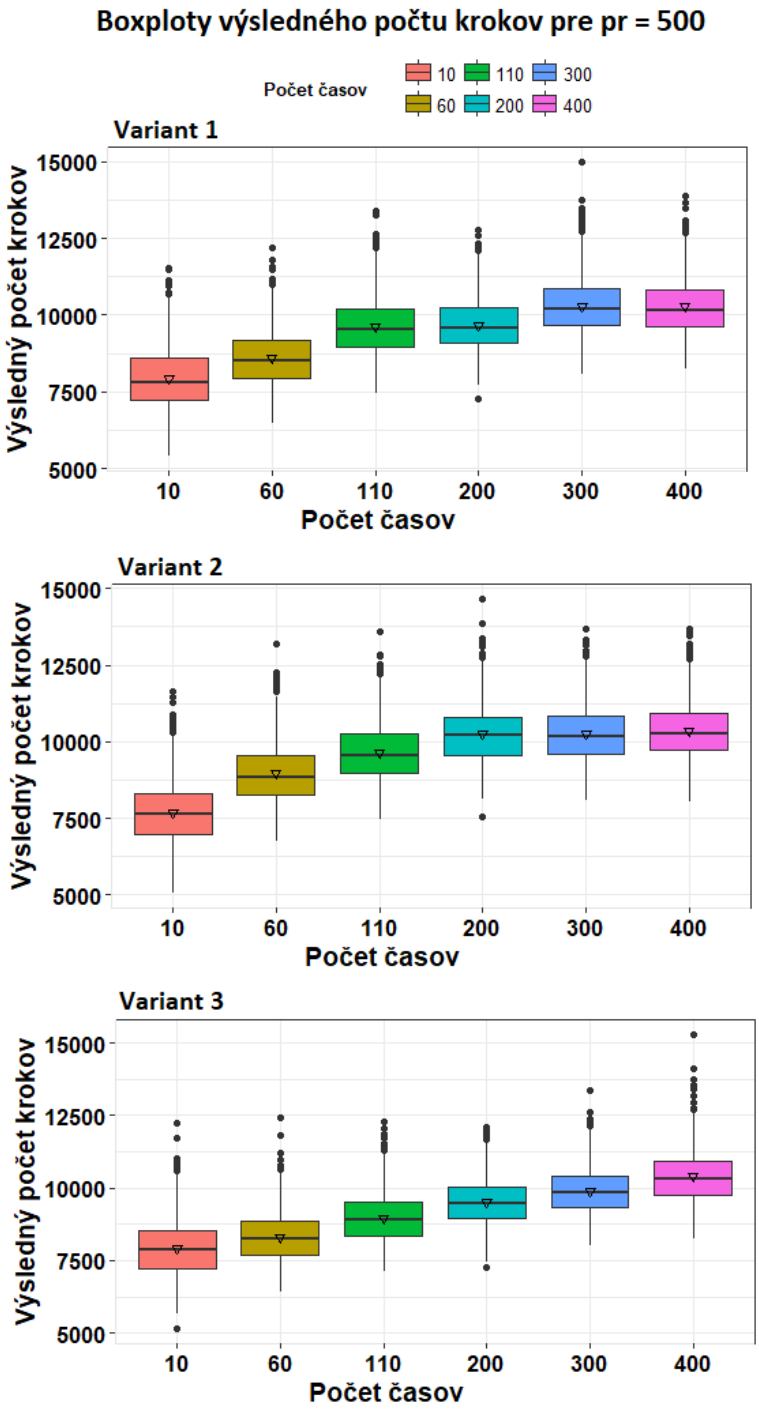
Na obrázku 3.16 sa nachádzajú grafy typu boxplot, ktoré znázorňujú, ako sa menila doba procesu synchronizácie vyjadrená v sekundách pri rôznych počtoch časov. U týchto grafov je však potrebné dať pozor na rozsah osi y u každého variantu. Priebeh každého variantu je dosť podobný priebehu v prípade doby meranej počtom krokov, ktorý je vidieť na obrázku 3.14. Najodľahlejšie pozorovanie výslednej doby synchronizácie variantu 1 dosahuje hodnotu približne do 1.65

sekúnd, kým u variantu 2 a 3 je to hodnota blízka 6.5 sekúnd. Boxploty i histogramy na obrázku 3.17 nám opäť zobrazujú, že vo všetkých prípadoch okrem jedného ide o približne symetrické rozdelenie. Len u variantu 2 s `pc = 110` sa javí, že ide o rozdelenie s kladnou šikmosťou. Medián je bližšie k dolnej časti boxu a je nižší ako aritmetický priemer. Takisto horný fúz je dlhší ako ten dolný. Ak sa pozrieme na histogram variantu 2 na brázku 3.17, môžeme vidieť, že pozorovania sa tiahnu až ďaleko k hodnote 6 sekúnd, narozdiel od ostatných prípadov tohto či iného variantu.

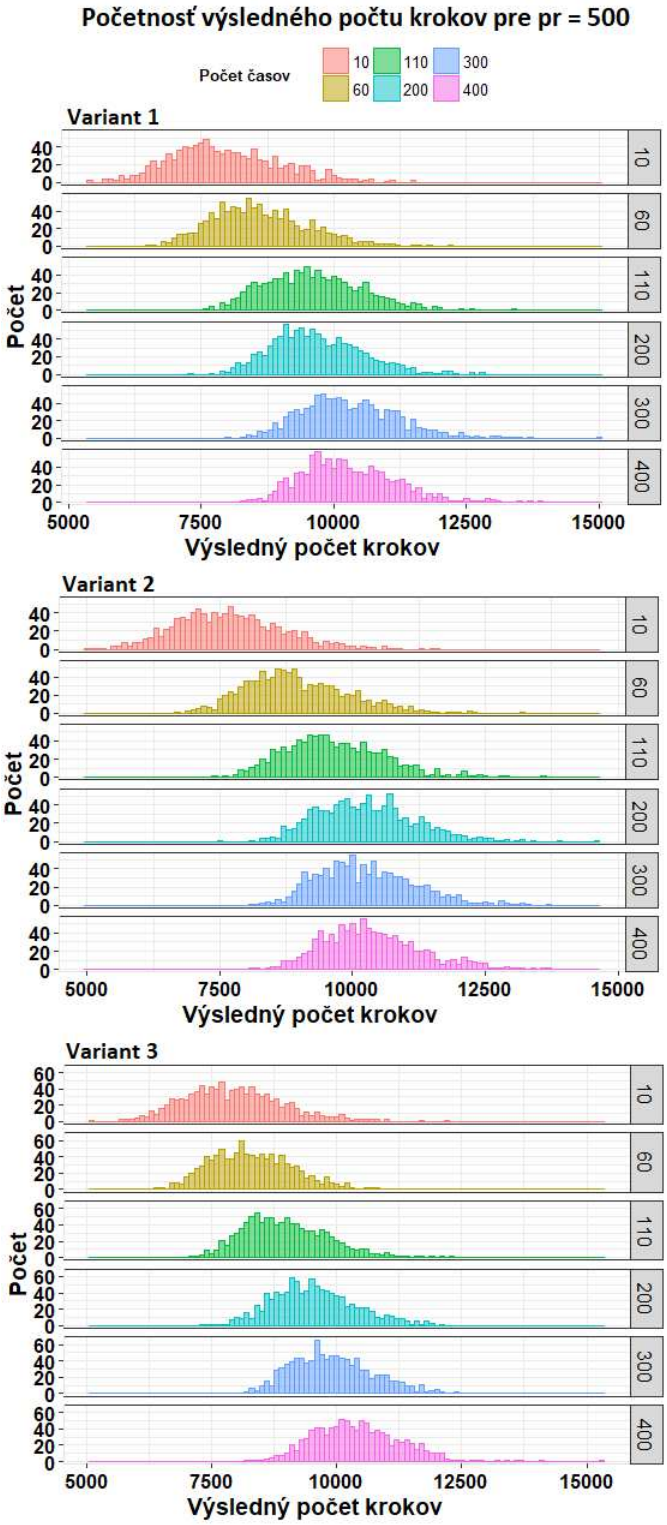
Zhrnutím našich výsledkov dochádzame k záveru, že varianty sú si veľmi podobné vo výsledných počtoch krokov pre danú dvojicu `(pc,pr)` či už v prípade meniaceho sa množstva robotov, alebo meniaceho sa množstva časov. Naše očakávania rátali s o niečo väčšími rozdielmi medzi variantmi, preto je tento výsledok pomerne prekvapivý. Napríklad u variantu 2, ktorý vyberá druhého robota s najmenšou vzdialenosťou od už vybraného prvého robota, by sa mohlo očakávať, že bude potreba viac krokov, kým sa všetky roboty zosynchronizujú. Vplyv na to určite malo aj nastavenie smerodajnej odchýlky. Pri veľmi malej odchýlke by sa roboty pohybovali menej a pravdepodobne by tak synchronizovanie robotov prebiehalo najprv v rámci zhlukov. A než by sa zosynchronizovali i tieto zhluky, vyžadovalo by to zrejme väčší počet krokov.

Ak ide o dobu meranú v sekundách, je vidieť jasný rozdiel medzi variantmi. Variant 1 dokončil proces synchronizácie najrýchlejšie pre akúkoľvek dvojicu `(pc,pr)`, kým variantu 3 to trvalo najdlhšie, čo sa dalo odhadovať, ak si uvedomíme podoby jednotlivých zdrojových kódov. Akurát v prípade 10 robotov a 12 časov sa to podľa grafov ťažko posudzuje. Proces vtedy skončil veľmi rýchlo u každého variantu a keďže výsledky doby v sekundách boli merané len na 2 desatinné miesta, tak to vyvoláva dojem, že proces trval rovnako u všetkých variantov, čo by tak nemuselo byť pri väčšom počte desatinných miest. Pri meraní doby procesu v sekundách je tiež vhodné uvedomiť si, že vplyv na výsledné hodnoty má aj výkonnosť počítača, na ktorom sme proces synchronizácie spustili.

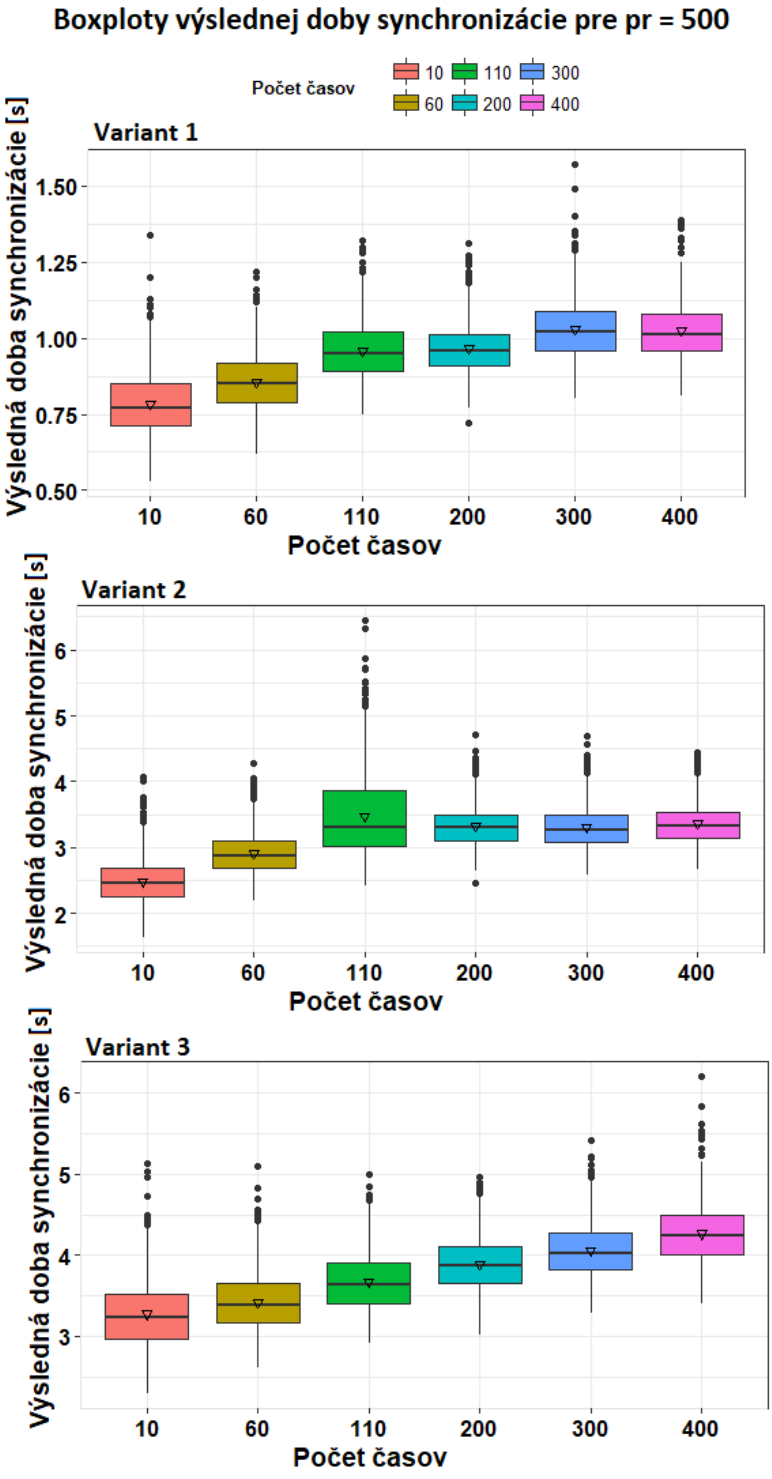
Zamyslime sa ešte nad jednou otázkou: čím je výsledný počet krokov procesu



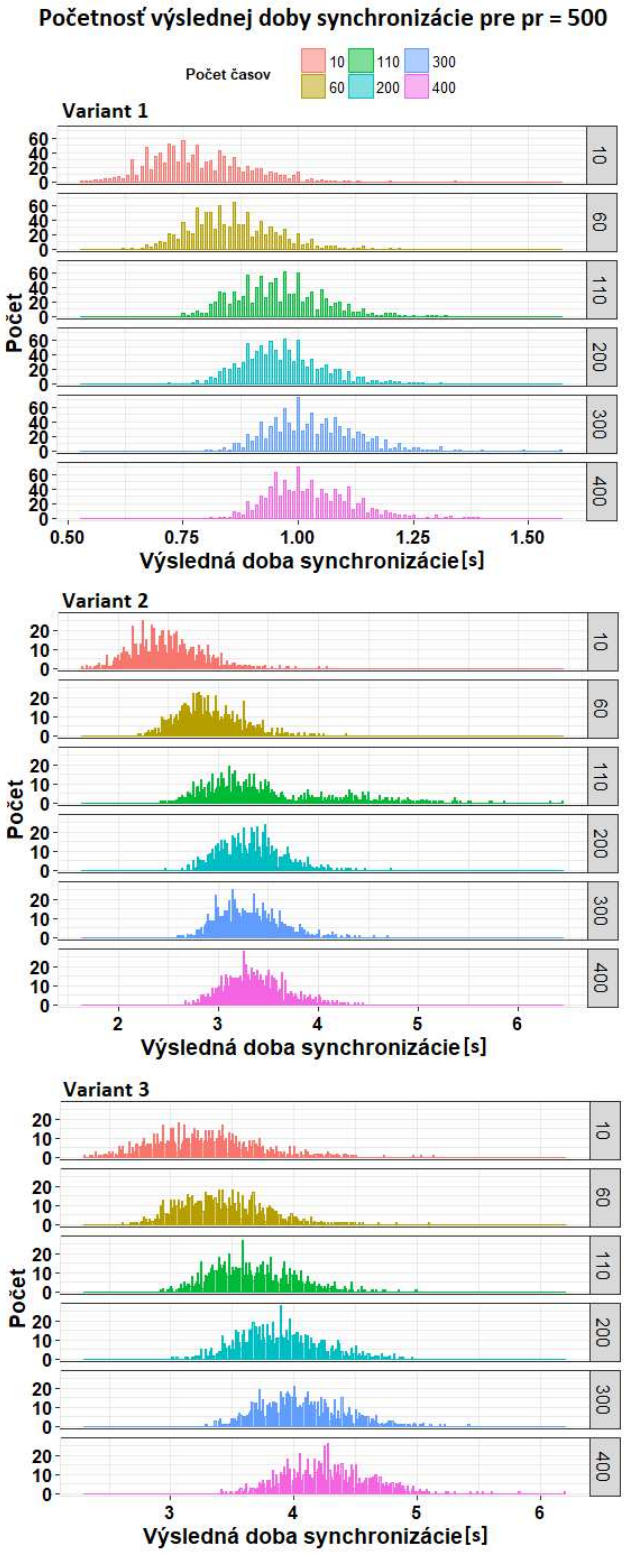
Obr. 3.14: Počet krokov pre každú hodnotu pc - boxplot



Obr. 3.15: Počet krokov pre každú hodnotu `pc` - histogram



Obr. 3.16: Doba synchronizácie pre každú hodnotu `pc` - boxplot



Obr. 3.17: Doba synchronizácie pre každú hodnotu pc - histogram

synchronizácie ovplyvnený viac? Počtom robotov alebo počtom časov? Pozrime sa spoločne na boxploty výsledného počtu krokov pre variant 2, t. j. na obrázok 3.6 a obrázok 3.14. Na prvom z obrázkov sa pri takmer dvojnásobnom zvýšení počtu robotov z hodnoty 130 na 250 zvýšil výsledný počet krokov odhadom z 1600 na 3300, čo je približne 2-krát väčšie množstvo. Na druhom obrázku sa pri takmer dvojnásobnom zvýšení počtu časov z hodnoty 60 na 110 zvýšil výsledný počet krokov približne z 8800 na 9400. V tomto prípade nárast nie je zďaleka taký výrazný ako pri zvýšení počtu robotov. Podobne to môžeme vidieť aj u ostatných variantov a napríklad pri prechode z 500 robotov na 750 na obrázku 3.7 a z 200 časov na 300 na obrázku 3.14. Náš záver teda znie, že počet robotov má podstatne väčší vplyv na výsledný počet krokov doby procesu synchronizácie než počet časov.

Kapitola 4

Vizualizácia procesu synchronizácie

Nasledujúca kapitola je v podstate stručným návodom ako z uložených grafov v podobe súborov vo formáte *.png*, prípadne iného typu, vytvoriť vizualizáciu procesu synchronizácie robotov ako súbor *.mp4*. Vizualizáciu sme sa rozhodli zvoliť vo forme pripomínajúcej blikanie svätojánskych mušiek. Inšpiráciou nám bol aj článok [2]. Najprv si však v pár bodoch zhrnime, čo je potrebné vykonať, než spustíme simuláciu s ukladaním grafov. Nasledujúce kroky platia pre ktorýkoľvek z troch variantov:

1. odkomentovať príkaz pre načítanie vopred nainštalovaného balíčka `ggplot2` nachádzajúceho sa na začiatku každého z našich R skriptov,

2. nastaviť premennú `pocet_behov` na hodnotu 1 a premenné `pocet_casov` a `pocet_robotov` na vektory dĺžky 1 (inak dôjde k prepisovaniu súborov v zložke z kroku 5),

3. vo funkcii `Synchronizuj()` odkomentovať vytváranie premenných `bliknutie` a `dataset` a volanie funkcie `UlozGraf()`,

4. vo funkcii `SynchPreDvojicuPrAPc()` odkomentovať volanie funkcie `UlozGrafy_poSynch()`,

5. na disku `C` vytvoriť zložku s názvom `SYNC`. Názov zložky či jej umiestnenie je možné zmeniť, následne je však potrebné toto nastavenie premietnuť aj

do funkcií `UlozGraf()` a `UlozSynchGraf()`. Zároveň odporúčame zložku pred každým spustením simulácie s ukladaním grafov vyprázdniť.

Po splnení všetkých uvedených bodov stačí naraz spustiť celý R skript. Je však dobré si uvedomiť, že vykresľovaním grafov a ich ukladaním je simulácia výrazne spomalená, a tak i doba celého procesu synchronizácie je podstatne dlhšia. Z tohto dôvodu odporúčame spúšťať simuláciu s ukladaním grafov pre menší počet robotov. Pre príklad: v prípade 100 robotov a 20 časov býva do zložky uložených zväčša 1000 až 2000 súborov a simulácia je ukončená obvykle do 3-4 minút.[1]

Keď máme po skončení simulácie uložené všetky *.png* súbory, môžeme z nich vytvoriť vizualizáciu ako *.mp4* súbor. K tomu použijeme free software *FFmpeg* [1], ktorý vie narábať s multimediálnymi dátami. Nasledujúci postup, ktorého detailnejšiu verziu možno nájsť v [19], nám pomôže tento software nainštalovať (vyskúšané pre operačný systém *Windows*).

1. Stiahneme poslednú statickú verziu softwaru *FFmpeg* z webovej stránky https://ffmpeg.zeranoe.com/builds/ odpovedajúcu nášmu operačnému systému.[2]

2. V prípade, že nemáme už nainštalované, stiahneme open source software *7-Zip* z webovej stránky http://www.7-zip.org/.

3. Pomocou *7-Zip* rozbalíme stiahnutý súbor z 1. kroku do samostatnej zložky.

4. Na disku v mieste, kde sa nachádza zložka *Windows* či *Program Files*, zvyčajne je to disk *C*, vytvoríme novú zložku s názvom *ffmpeg*.

5. Všetky súbory zo zložky z 3. kroku skopírujeme a vložíme do vytvorenej zložky *ffmpeg*.

[1] Simulácia bola spustená na počítači s procesorom Intel Core i5 2.30 GHz, RAM 8 GB a 64-bitovým operačným systémom.

[2] CD príloha tejto diplomovej práce obsahuje nami stiahnutú a použitú statickú verziu (20171111-3af2bf0) softwaru *FFmpeg* pre Windows 64-bit. V prípade, že chcete použiť túto, prejdite priamo na krok 4.

6. V okne *Systém* (*Ovládací panel\Systém a zabezpečenie\Systém* alebo pravým klikom na *Tento počítač* a následne *Vlastnosti*) klikneme na *Rozšírené systémové nastavenia* a v okne *Vlastnosti systému* na *Premenné prostredia* (pozri obrázok 4.1).

7. V časti *Používateľské premenné* vyberieme *Path* a klikneme na tlačidlo *Upraviť*. V prípade, že tam premenná *Path* nie je, tak ju najprv vytvoríme.

8. Do poľa pre hodnoty premennej za čímkoľvek, čo je tam prípadne už napísané, vložíme `;c:\ffmpeg\bin` a zmenu uložíme.

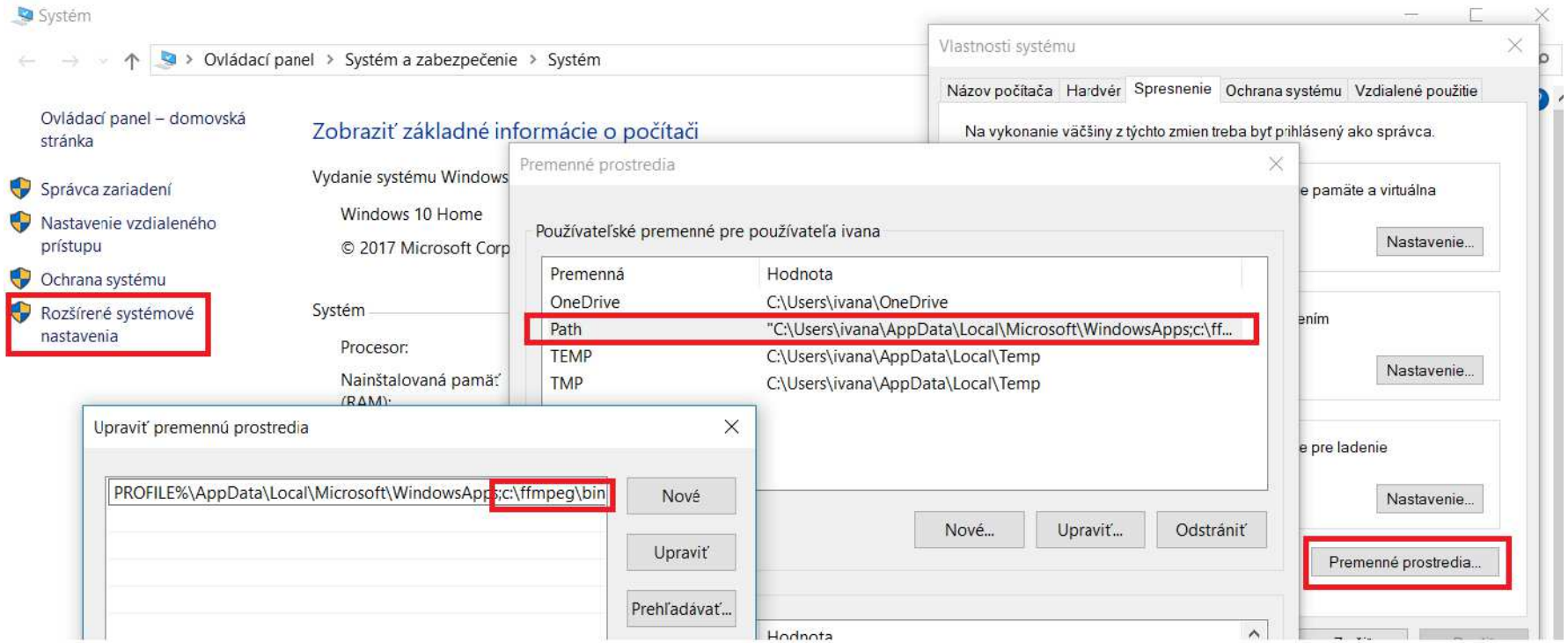
9. Otvoríme *Príkazový riadok* a spustíme príkaz `ffmpeg -version`, ktorý by nám v prípade úspešnej inštalácie mal vypísať verziu softwaru *FFmpeg*. Software je tak pripravený a môžeme ho kedykoľvek použiť.

Po zdarnom splnení týchto krokov nám chýba už len málo k samotnému vytvoreniu *.mp4* súboru.

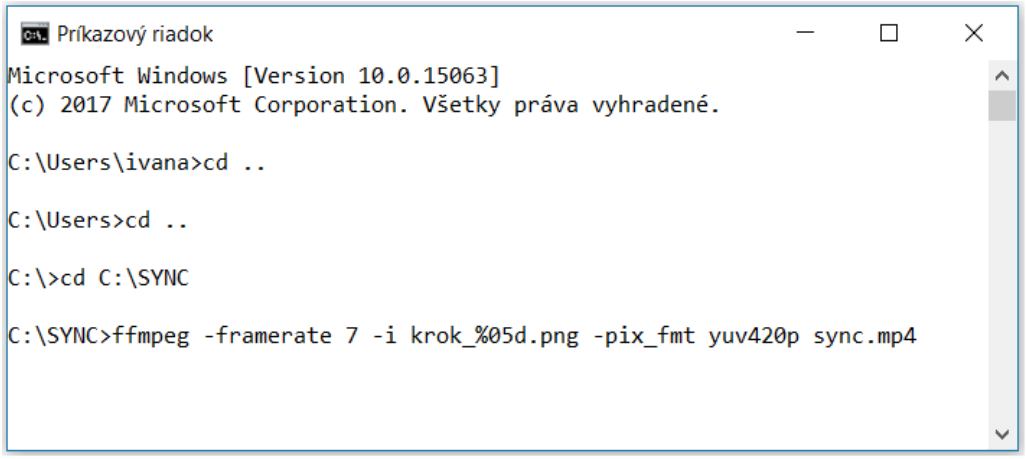
1. V príkazovom riadku najprv nastavíme adresár na práve ten, ktorý máme vo funkciách `UlozGraf()` a `UlozSynchGraf()`, tzn. `C:\SYNC`. Preto spustíme príkaz `cd ..` toľkokrát, až kým nebude zobrazené iba `C:\>`, a vzápätí nato príkaz `cd C:\SYNC` (pozri obrázok 4.2).

2. Nakoniec vložíme príkaz `ffmpeg -framerate 7 -i krok_%05d.png -pix_fmt yuv420p sync.mp4` (je možné upraviť podľa vlastných preferencií) a spustíme. Po chvíli je naša cielená vizualizácia procesu synchronizácie robotov vytvorená v zložke *C:\SYNC* ako súbor *sync.mp4*.[3]

[3]Pre každý variant procesu synchronizácie je na CD prílohe tejto diplomovej práce pripravený takýto *.mp4* súbor ako názorná ukážka tejto vizualizácie, a to pre 100 robotov a 20 časov.



Obr. 4.1: Povolenie *FFmpeg*



Obr. 4.2: Spustenie *FFmpeg* v *Príkazovom riadku*

Záver

Cieľom tejto diplomovej práce bolo vytvoriť simuláciu procesu synchronizácie robotov. Predtým však bolo potrebné oboznámiť sa so samotným pojmom synchronizácia. Preto sme si tento pojem v kapitole 1 stručne vysvetlili a upozornili, že synchronizácia, ktorou sa budeme v ďalších kapitolách zaoberať, bude tzv. *spontánna*, u ktorej proces synchronizácie prebieha automaticky. Jej účastníci sa organizujú sami a nie na základe „pokynov“ nejakého vedúceho prvku. Pre lepšiu predstavu a pochopenie sme si následne opísali niekoľko príkladov spontánnej synchronizácie a to z rôznych oblastí života.

Kapitola 2 už pojednáva o našej vytvorenej simulácii, ktorú sme sa rozhodli naprogramovať v prostredí jazyka R. Po úvodnom zoznámení sa s tým, ako chceme aby proces synchronizácie prebiehal, sme objasnili odlišnosti troch variantov simulácie. Počas práce na programe simulácie sme totiž narazili na rôzne možnosti, ako docieliť výber dvojice robotov, ktoré majú zosynchronizovať svoje časy. Z tohto dôvodu vznikli spomínané tri varianty simulácie, a teda i tri R skripty s čiastočne odlišujúcim sa zdrojovým kódom. U variantu 1 je totiž dvojica robotov vybraná náhodne, u variantu 2 je druhý robot z dvojice vybraný ako ten najbližší k prvému robotu a u variantu 3 je druhý robot vybraný podľa vypočítaných pravdepodobností, ktoré sú úmerné vzdialenostiam. Ďalšie časti tejto kapitoly vysvetľujú samotné naprogramované funkcie obsiahnuté v R skriptoch.

Po naprogramovaní simulácie bolo ďalším cieľom využiť simuláciu k získaniu dát a tie potom analyzovať. Zaujímala nás najmä doba procesu synchronizácie a jej závislosť na počte robotov, resp. na počte časov/časových dielikov. Dobu

procesu synchronizácie sme merali v sekundách a v počte krokov, ktoré bolo potrebné vykonať k úspešnej synchronizácii všetkých robotov, pričom v jednom kroku sa synchronizuje jedna dvojica robotov.

Z výsledkov simulácie, ktoré sme si graficky zobrazili, je zrejmé, že doba synchronizácie vyjadrená počtom krokov sa pri zvyšujúcom sa počte robotov takisto zvyšuje. Varianty sa však pri rovnakých vstupných nastaveniach líšia vo výslednom počte krokov len mierne, čo je celkom prekvapivý výsledok. Odhadovali sme totiž, že napríklad variant 2 bude k dosiahnutiu zosynchronizovania všetkých robotov potrebovať viac krokov vzhľadom k tomu, že pracuje s minimálnou vzdialenosťou zo vzdialeností medzi prvým vybraným robotom do dvojice a zvyšnými robotmi. Otázkou je, či by sa tieto len mierne rozdiely medzi variantmi zmenili v prípade, že by sme upravovali veľkosť smerodajnej odchýlky, ktorá by tak ovplyvňovala, ako veľmi by sa roboty na štvorcovej ploche pohybovali. Tento pokus žiaľ diplomová práca už neobsahuje.

Čo sa týka doby procesu synchronizácie vyjadrenej v sekundách, takisto platí, že so zvyšujúcim sa počtom robotov sa zvyšuje aj doba procesu. Rozdiely medzi variantmi, opäť pri rovnakých vstupných nastaveniach, sú však v tejto situácii väčšie ako v prípade počtu krokov. Z výsledkov nám vyplynulo, že ako najrýchlejší variant sa javí variant 1 a najdlhšie proces synchronizácie trval variantu 3. Tento záver sme odhadovali vzhľadom na to, ako funkčne náročné sú jednotlivé varianty.

V ďalšej časti analýzy venovanej závislosti doby procesu synchronizácie na počte časov sme získali výsledky, ktoré už nie sú tak jednoznačné ako v prípade závislosti doby procesu synchronizácie na počte robotov, resp. počte časov. Simulácia pracovala s 500 robotmi a so šiestimi rôznymi hodnotami počtu časov v rozmedzí od 10 do 400.

Varianty sa pri rovnakých vstupných nastaveniach nelíšia vo výslednom počte krokov nejak závratne. Avšak pri prvých troch hodnotách počtu časov sa doba procesu synchronizácie vyjadrená počtom krokov u každého variantu zvyšovala, kým pri ďalších troch hodnotách počet krokov mierne kolísal alebo sa takmer vyrovnal. To neplatilo pre variant 3, u ktorého sme mohli vidieť stále naras-

tajúci počet krokov v celom rozsahu, t. j. od 10 do 400 časov. Na základe týchto výsledkov preto nie je možné jasne určiť vzťah medzi dobou procesu synchronizácie v podobe počtu krokov a počtom časov. Bolo by žiaduce tento pokus opakovať, prípadne skúsiť voliť i iný počet robotov. Domnievame sa totiž, že určitú rolu hrá i pomer medzi zvoleným počtom robotov a počtom časov.

V prípade závislosti doby procesu synchronizácie v sekundách na počte časov vykazuje variant 3 v spomínanom rozsahu 10 až 400 časov rastúcu tendenciu a je jasne najpomalší. Varianty 1 a 2 majú však od druhej polovice rozsahu opäť kolísajúcu až takmer vyrovnanú dobu procesu v sekundách, ako tomu bolo podobne i pri dobe vyjadrenej počtom krokov. Jednoznačné však je, že variant 1 je najrýchlejší.

Analýza závislosti doby procesu synchronizácie na počte časov by si očividne vyžadovala viac experimentovania. Z našich výsledkov sa môžeme len domnievať, že pri pevne danom počte robotov sa so zmenšujúcim sa pomerom počtu robotov a počtu časov doba procesu synchronizácie vyjadrená v sekundách či v počte krokov vyrovnáva u variantu 1 a 2, u variantu 3 rastie. Otázkou ale je, aké výsledky by sme dostali, ak by bol tento pomer menší ako 1, tzn. keď je časov viac ako robotov.

Taktiež sme sa v krátkosti zamysleli nad vplyvom počtu robotov, resp. počtu časov na výsledný počet krokov. Výsledky simulácií jednoznačne ukazujú na väčší vplyv počtu robotov. Pri jeho zmene sa počet krokov zmení o dosť výraznejšie než pri zmene počtu časov.

Pre zaujímavosť sme sa počas spracovávania diplomovej práce rozhodli pokúsiť sa i o vytvorenie vizualizácie procesu synchronizácie, čo sa nám aj podarilo a postup, ako si takúto vizualizáciu zhotoviť zahŕňa práve posledná kapitola. Samotná vizualizácia každého variantu je k dispozícii na priloženom CD.

Literatúra

[1] FFmpeg Developers. (2016): *ffmpeg tool (Version 3af2bf0).* [software]. dostupné z: http://ffmpeg.org/.

[2] Fürst, T.: *Světlušky, larvy, hadi ... dostanu já někdy normální nápad?* [online]. 2013, [cit. 2017-04-20]. dostupné z: http://furstovi.blogspot.cz/svetlusky-larvy-hadi.html.

[3] Gerr, M.: *Unit 12, InSync.* [online]. 2017, [cit. 2017-10-06]. dostupné z: https://www.learner.org/courses/MathIlluminated_12_txt.pdf.

[4] *Kilobot.* [online]. dostupné z: https://www.k-team.com/kilobot.

[5] Nadis, S.: *All together now.* [online]. 2003, [cit. 2017-09-17]. dostupné z: https://static1.squarespace.com/static/all-together-now.pdf.

[6] Néda, Z., Ravasz, E.: *The sound of many hands clapping.* [online]. 2000, [cit. 2017-09-20]. dostupné z: http://regan.med.harvard.edu/pdf/publications/Clap_Nature.pdf.

[7] Peterson, I.: *Step in Time.* [online]. 1991, [cit. 2017-09-16]. dostupné z: https://static.squarespace.com/static/step-in-time.pdf.

[8] R Core Team (2017): *R: A language and environment for statistical computing.* R Foundation for Statistical Computing, Vienna, Austria. dostupné z: https://www.R-project.org/.

[9] Rubenstein, M., Nicholas, H., Nagpal, R.: *Kilobot: A Low Cost Scalable Robot System for Collective Behaviors.* [online]. 2015, [cit. 2017-09-16]. dostupné z: https://iatranshumanisme.files.wordpress.com/2015/08/kilobot.pdf.

[10] Stewart, I.: *All together now ...* [online]. 1991, [cit. 2017-09-16]. dostupné z: https://static.squarespace.com/static/all-together-now-1991.pdf.

[11] Strogatz, S.: *Spontaneous synchronization in nature.* [online]. 1997, [cit. 2017-09-16]. dostupné z: http://citeseerx.edu/spontaneous-synchronization-in-nature.pdf.

[12] Strogatz, S.: *Sync : The Emerging Science of Spontaneous Order.* Hyperion, New York, 2003.

[13] Strogatz, S.: *The science of sync.* [online]. 2004, [cit. 2017-04-18]. dostupné z: https://www.ted.com/talks/steven_strogatz_on_sync/transcript.

[14] Strogatz, S., Stewart, I.: *Coupled oscillators and biological synchronization.* [online]. 1993, [cit. 2017-09-16]. dostupné z: https://squarespace.com/coupled-oscillators-and-biological-sync.pdf.

[15] *Synchronization of Metronomes.* [online]. 2017, [cit. 2017-09-20]. dostupné z: https://sciencedemonstrations.harvard.edu/synchronization-metronomes.

[16] Třešňák, P.: *Záhada lidského souznění.* [online]. 2017, [cit. 2017-10-06]. dostupné z: https://www.respekt.cz/tydenik/2017/30/zahada-lidskeho-souzneni.

[17] Vencálek, O., Fürst, T.: *Pravděpodobnostní modely synchronizace.* Česká statistická společnost, Praha, 2015.

[18] Wickham, H.: *ggplot2: Elegant Graphics for Data Analysis.* Springer-Verlag New York, 2009.

[19] *wikiHow to Install FFmpeg on Windows.* [online]. dostupné z: https://www.wikihow.com/Install-FFmpeg-on-Windows.