

UNIVERSITY OF TARTU  
Institute of Computer Science  
Computer Science Curriculum

Janno Jõgeva

# Software Infrastructure and Course Design of a Robotics Course

Master's thesis (30 ECTS)

Supervisor: Eno Tõnisson, PhD

Tartu 2019

## **Software Infrastructure and Course Design of a Robotics Course**

**Abstract:** This thesis covers the design and development process of a robotics course. The course in question serves as an entry point into the field of robotics in the University of Tartu. The design of the thesis draws on the ideas of educational action research and the infrastructure of the course on the software development and information technology operations practices (DevOps). Cycles of improvements, which are core to both of these practices, are in tune with the rapid evolution required of a modern robotics course. Infrastructure as code, version control systems, and incentive towards continuous integration are all used in the development process. The two runs of the course covered by the thesis have advanced the knowledge of more than a hundred students in some of the modern tools and technologies used in robotics. Altogether, the development of the course has helped to reinforce the line-up of students and instructors available for projects and more advanced courses in the Institute of Technology and the university as a whole. The author, being the lecturer in charge leading the development the course and also pushing for the use of new technology and maintaining chosen solutions. The thesis serves as an organised documentation on the evolution process and reasoning for the choices made.

**Keywords:** Course Design; Course Scalability; Robotics; Action Research.

**CERCS (Common European Research Classification Scheme):**

1. T125 - Automation, robotics, control engineering
2. P175 - Informatics, systems theory
3. S281 - Computer-assisted education.

## Õppeaine „Roobotika“ tarkvaraline taristu ja kursuse disain

**Lühikokkuvõte:** Käesolev töö käsitleb robootika kursuse disaini ja arenduse protsessi. Käsitletav kursus on Tartu Ülikoolis üheks võimalikuks lähtepunktiks robootika valdkonda sisenemiseks. Disaini protsessis on kasutatud elemente tegevusuuringu ülesehitusest ning *DevOps* nime kandvast süsteemiarenduse kultuurist. Mõlemale praktikale omane tsükliline paranduste tegemine harmoniseerub hästi tänapäevase robootika kursuse jaoks vajaliku kiire arenguga. Taristu kui kood (*infrastructure as code*), versioonihaldussüsteemid ja püüd rakendada pideva integratsiooni (*continuous integration*) põhimõtteid — on kõik osa kursuse arengust. Roobotika kursus on teemaks oleva kahe toimumiskorra jooksul aidanud arendada enam kui saja tudengi teadmisi mõningatest tänapäevases robootikas kasutatavatest töövahenditest ja tehnoloogiatest. Töös kajastatud kursuse arendusprotsess on toetanud robootikaalaste teadmistega tudengite ning nende juhendajate järelkasvu, mis loob aluse keerukamate kursuste arendamiseks ja kasvatab tehnoloogiainstituudi — ning ülikooli üldisemalt — võimekust olla koostööpartner robootika valdkonnas. Autor kandis vastutava õppejõu rolli, juhtides kursuse arendust, uute tehnoloogiate kasutamist ning juurutamist. Töö eesmärk on olla läbitud arengu, tehtud valikute, ning nende põhjenduste süstematiseeritud dokumentatsioon.

**Võtmesõnad:** Kursuse koostamine; Kursuse skaleeruvus; Roobotika; Tegevusuuring.

**CERCS** (Common European Research Classification Scheme):

1. T125 - Automatiseerimine, robootika, control engineering
2. P175 - Informaatika, süsteemiteooria
3. S281 - Arvuti õpiprogrammide kasutamise meetoodika ja pedagoogika.

# Table of Contents

1. Introduction .....	1
2. Abbreviations and Acronyms .....	3
3. Course Overview .....	5
3.1. Bird's Eye View .....	5
3.2. Linked Courses .....	8
3.3. Links to Computer Engineering Curriculum .....	9
3.4. Target Audience .....	10
4. Course Design .....	15
4.1. Methodology .....	15
4.2. Grading System and Results .....	17
4.3. Lab Manual Release Schedule .....	25
4.4. Instructors Workload .....	30
5. Student-Facing Solutions .....	34
5.1. Moodle Dashboard .....	34
5.2. Software Stack .....	36
5.3. OpenCV Adaptations .....	41
5.4. Version Control .....	45
6. Internal-Facing Solutions .....	53
6.1. Instructors' Repository .....	53
6.2. Lab Manual Template .....	54
6.3. Google Docs .....	55
6.4. Configuration Management .....	57
6.5. Resource Management System .....	59
7. Summary .....	65
Acknowledgement .....	66
References .....	67
Licence .....	70

# 1. Introduction

This thesis covers the design and development process of a robotics course. The course in question serves as an entry point into the field of robotics in the University of Tartu (UT) as envisioned by the Institute of Technology (TUIT). The field in question—Robotics is a research field centred around robots making use of cross- and interdisciplinary methodologies. For the purposes of the course, a robot is a programmable machine that performs autonomous decisions and actions in solving a task or parts of it. Robotics is leading much of what is considered possible in the modern world. Rapid advances of the past decades in the field of robotics have also raised some concerns, mainly the socio-economic risk created in many sectors [1]. Some of the risks are counterweighed by enabling the population on the local, European, and global level to solve problems at scale or environment otherwise unreachable or impractical. The amount of possible applications for robotics creates a high demand in the job market for specialists with robotics background. The course aims to help the supply of some of these specialists.

The course's design draws on the ideas of educational action research and the infrastructure of the course on the software development and information technology operations (DevOps). DevOps practices used are: infrastructure as code (IaC), version control systems (VCS) and incentive towards continuous integration (CI). The concept of IaC is applied via Ansible [2] configuration management system's Playbooks, describing desired system states and policies, which can be applied to the target hosts [3]. The incentive towards CI is supported by using the AsciiDoc language [4] and the AsciiDoctor toolchain [5] for document development coupled with VCS to enable rapid updates of the course content. Altogether action research is a widely researched educational methodology and DevOps a widely used collection of best practices for systems development with inherent benefits of scalability and sustainability. Both of those schools of thought are based on the idea of rapid development cycles and the drive to improve the processes within via review and reflection. These concepts are in tune with the ever-changing field of robotics and automation.

This thesis covers the continuous period starting from planning for the 2017/2018S (S - spring; A - autumn) semester and ending with some bits of the ongoing planning for 2019/2020A semester. This translates to two runs of the course and a part of the planning stage for the third run. The author does have previous experience with conducting this course, which is outside of the scope of this work. In the 2017/2018S and 2018/2019A semesters combined the course has had 14 instructors led by the author for laboratory practicals and support from various employees of the TUIT, Information Technology Office

(ITO), and the University of Tartu in general. The main contributions of the author are leading the design and development of the course, managing the VCS and various templates, delivering lectures, pushing for the use of new technology and maintaining chosen solutions.

Although the thesis is focused on the author's work, it would not be possible to give a clear and concise overview without mentioning some contributions from other people. The author will indicate where such overlap occurs. The contributions considered for this paper are mostly related to the infrastructure and high-level design of the course. Conducting all of the lectures was also part of the author's responsibilities in the course but not all of the topics covered are part of this thesis. Thus, the description's focus is on a template for a lab manual, not on the individual labs. The following section will expand on the topics covered by the thesis.

[Chapter 2](#) lists all of the field-specific abbreviations and acronyms used in the thesis. [Chapter 3](#) gives a general overview of the robotics course, its placement in the curriculum and the target audience. The chapter on general overview is followed by [Chapter 4](#) explaining how the lab manuals are released to the students, how the overall grading system works, instructor workload throughout the semester and the methodology used for compiling this paper. The following two sections continue to describe the diverse set of tools used for the course. First of the two, [Chapter 5](#) is centred around tools that the students would be in direct contact with while enrolled — Moodle dashboard, software on classroom computers, Open Source Computer Vision (OpenCV) modifications and the version control system. Second of the two, [Chapter 6](#) explains the technology used behind the scenes, to run the course. The chapter starts off with VCS and after which it continues to the lab manual template that is maintained within. Subsequent sections cover: Google Docs based solutions, configuration and resource management, and the server which was set up by the author for the purposes of the course. Finally, [Chapter 7](#) concludes the paper.

## 2. Abbreviations and Acronyms

Abbreviation/Acronym	Full Term
API	Application Programming Interface
AR	Action Research
AR-cycles	Action Research <b>cycles</b>
armhf	<b>ARM</b> Hard Float
CE	Computer Engineering
CERCS	Common European Research Classification Scheme
CI	Continuous Integration
DevOps	<b>Development</b> and Information Technology <b>Operations</b>
DTD	Digital Timing Diagram
DVCS	Distributed Version Control Systems
ECTS	European Credit Transfer and Accumulation System
EOL	End-of-Life
FPU	Floating-Point Unit
IaC	Infrastructure <b>as Code</b>
IO	Input-Output
ITO	Information Technology Office
JSON	JavaScript Object Notation
Lab00	<b>Laboratory</b> Practical number <b>00</b>
Lin	<b>Linux</b>
LMS	Learning Management System
LTS	Long Term Support
np	<b>not present</b>
OpenCV	<b>Open</b> Source Computer Vision
OS	Operating Systems
PID	Proportional-Integral-Derivative
PKI	Public Key Infrastructure

<b>Abbreviation/Acronym</b>	<b>Full Term</b>
PoC	<b>P</b> roof of <b>C</b> oncept
PR	<b>P</b> ull <b>R</b> equest
RDP	<b>R</b> emote <b>D</b> esktop <b>P</b> rotocol
RPi	<b>R</b> aspberry <b>Pi</b>
ROS	<b>R</b> obot <b>O</b> perating <b>S</b> ystem
SaaS	<b>S</b> oftware- <b>a</b> s- <b>a</b> - <b>S</b> ervice
S&T	<b>S</b> cience <b>a</b> nd <b>T</b> echnology
SIS	<b>S</b> tudy <b>I</b> nformation <b>S</b> ystem
SSH	<b>S</b> ecure <b>S</b> hell
VNC	<b>V</b> irtual <b>N</b> etwork <b>C</b> omputing
TUIT	<b>T</b> artu <b>U</b> niversity, <b>I</b> nstitute of <b>T</b> echnology
UI	<b>U</b> ser <b>I</b> nterface
UPS	<b>U</b> ninterruptible <b>P</b> ower <b>S</b> upply
UT	<b>U</b> niversity of <b>T</b> artu
UX	<b>U</b> ser <b>eX</b> perience
VCS	<b>V</b> ersion <b>C</b> ontrol <b>S</b> ystems
Win	Microsoft <b>W</b> indows



# 3. Course Overview

This chapter explains why and how this course fits into the broader area of studies in Computer Engineering (CE) in the University of Tartu. Starting with the target audience and how it has evolved over the years. Continuing to how the course fits into the curricula of CE and Science and Technology (S&T) designed by the Institute of Technology. Finally, there is a brief characterisation of the students who have enrolled and passed the course in the applicable period. The majority of sections will be divided into subsections based on AR-cycles to better display the development process. There are a couple of exceptions on topics that have stayed fairly static over the covered period. Example, the title of a subsection "Plan (2017/2018A)" refers to the planning stage for the 2017/2018A semester happening before that semester started. [Section 4.1](#) gives a detailed overview of the methodology used in this thesis.

## 3.1. Bird's Eye View

The course yields 6 European Credit Transfer and Accumulation System (ECTS) points on successful completion and is a part of regular studies in UT. That and other characterising features for the course are presented in [Table 1](#) for improved readability. The table is based on the official course info available in Study Information System (SIS) [\[6\]](#).

*Table 1. Bird's Eye View of the Course*

Area	Description
Course type	Regular course
Grading	Differentiated (A, B, C, D, E, F, not present) - more in <a href="#">Section 4.2.2</a>
Forms and volume of study in hours	6 ECTS points - $6 \times 26h = 156h$ of work <ul style="list-style-type: none"><li>• Practical classes (Labs) - <math>64h</math></li><li>• Lectures - <math>32h</math></li><li>• Independent work (incl. e-learning) - <math>60h</math></li></ul>

Area	Description
Levels of study	<ul style="list-style-type: none"> <li>• Bachelor's studies</li> <li>• Master's studies</li> <li>• Doctoral studies</li> <li>• Bachelor's and master's integrated studies</li> </ul>
Goal	The goal of this course is to give basic theoretical and practical knowledge in robotics.
Brief description of content	Introduction into robotics systems using Raspberry Pi and Arduino in combination with GoPiGo platform.
Assessment (see <a href="#">Section 4.2</a> for more details)	<ul style="list-style-type: none"> <li>• Labs</li> <li>• Presentation</li> <li>• Project</li> <li>• Many specialisation categories</li> <li>• Oral exam</li> </ul>
Languages of instruction	The whole course can be completed in English. Estonian is available in all verbal communication and at the final exam.

These characteristics have been rather stable over the period covered by the thesis. In addition to the compulsory activities visits have been organised by the author to companies giving students first hand experience with practical applications of robotics. These optional activities have been popular amongst students and there are already some plans for the next semester. One ECTS converts to 26 academic hours of work for the student in UT and a full-time student in regular studies is expected to gather 30 ECTS points per semester [7]. The lectures, lasting 1.5h and running every week, are conducted by the author. The labs last 4h and are taught by two instructors per group. Students are expected but not limited to attend one lab group per week. Two additional optional lab consultation times were added in the 2018/2019A semester.

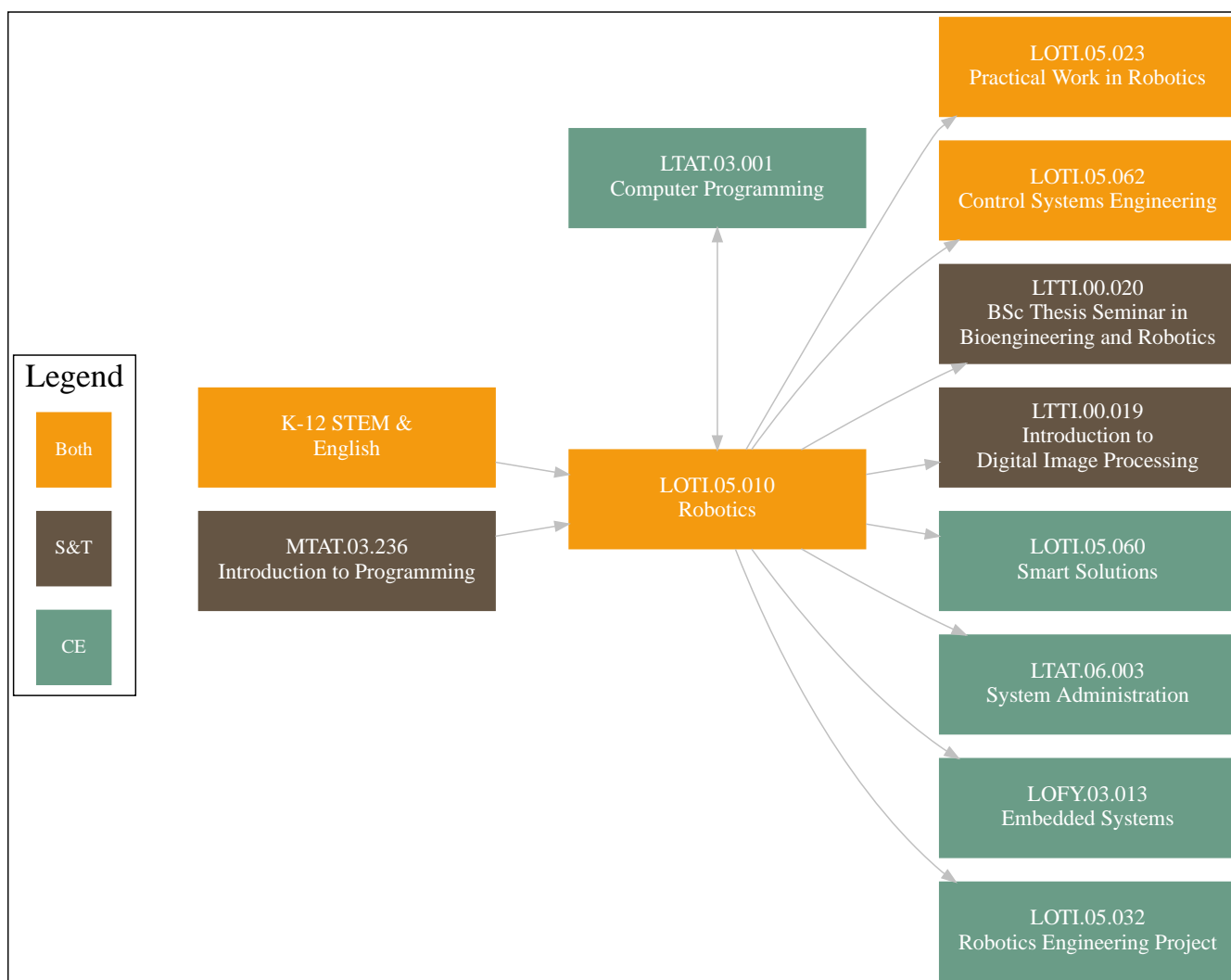
The course follows the common semester system in use at UT. The course runs for the whole 16-week semester: either September → December or February → May with exams taking place in the corresponding examination session in January or June.

The thesis mentions some modules of curricula. These are defined as follows for bachelor's studies curriculum in the study regulation of UT [7]:

1. two base modules, 24 ECTS each;
2. two field modules, 24 ECTS each, one of which may be replaced by a speciality module;
3. two speciality modules, 24 ECTS each, one of which may be replaced by a field module;
4. at least one elective module, 12 ECTS;
5. optional courses, 0–18 ECTS; and
6. bachelor's exam or graduation thesis, 6–12 ECTS.

## 3.2. Linked Courses

This course is considered as the entry point to the area of robotics in University of Tartu. The course focuses on coding for hardware and introducing various areas of robotics. After completing this course the students can enrol in courses specialising in sub-areas of robotics. [Figure 1](#) shows the dependencies between the courses and includes the unique identifiers used in the university. The course in question is "LOTI.05.010", the first letters "LOTI" before full stop specify the structural unit within UT responsible for the course and the two sets of numbers organise the course within the structural unit. The course makes heavy use of the knowledge students gain in the introductory Computer Programming course.



*Figure 1. Courses supporting, and supported by the robotics course, to develop students knowledge in robotics*

These courses have mostly stayed the same over the action research cycles (AR-cycles) considered and thus, are covered as a single item. First tests of cooperation with the programming course were made in the 2018/2019A semester and there are plans for collaboration with the introductory mathematics course.

### 3.3. Links to Computer Engineering Curriculum

As with [Section 3.2](#) the course has stayed in the same module over all of the AR-cycles and the division to cycle steps is not applied here. The course is part of the "Specialisation Module in Computer Engineering" optional module which is part of the Computer Engineering curriculum. This module while optional is what the majority of students opt for. The general objective of this module is "To give student specialised knowledge in two distinct fields of Computer Engineering." [8]. This information is publicly accessible via the universities SIS. The module contains 5 courses:

1. LOTI.05.062 | Control Systems Engineering 3 ECTS;
2. LOFY.03.013 | Embedded Systems 3 ECTS;
3. LOTI.05.023 | Practical Work in Robotics 6 ECTS;
4. LOTI.05.010 | Robotics 6 ECTS;
5. LTAT.06.003 | System Administration 6 ECTS.

Each curricula has a public description of its objectives. Targets listed in the description of SIS [8] have also been taken into account when designing this course. [Table 2](#) lists each of these objectives with an explanation if and how the course contributes towards these targets.

*Table 2. Course Contribution Towards "Computer Engineering" Curriculum Objectives*

<b>Curriculum objectives (copy from SIS)</b>	<b>Course contribution</b>
After completion, the student	
Has a systematic understanding of the fundamental concepts, theoretical principles, and methodology inherent to CE.	Has written code for an electromechanical system, including fine tuning and debugging the solution to make it work in a controlled environment.
Can independently gather and critically analyse information in the field of CE and communicate observations and conclusions in an appropriate form (e.g. oral or written presentation) in Estonian as well as in English.	Has presented a recent news article to all of the other students via a presentation in the lecture or a 2 minute news clip (video) that get's shown in the lecture. This work is done in English. Most of the summative feedback in labs is provided in Estonian.

<b>Curriculum objectives (copy from SIS)</b>	<b>Course contribution</b>
Is able to write computer code using assembler and at least two higher-level programming languages.	Has used Python and Arduino C++. Use of assembler is not covered in this course.
Understands the roles of individual computer components, is able to select an appropriate configuration (considering both price and performance) as well as assemble this computer.	Has gathered experience in sensors, actuators, other input-output (IO) devices and communication. Has participated in a project finding suitable system configuration and balance between using software and hardware for the final solution.
Understands the organisation and architecture of computers and other electronic devices on a level that allows the conception, design, and troubleshooting of electronic devices.	The course focuses on the ability to assemble pre-fabricated development boards and sensors. Students get a very basic introduction to breadboarding. Electronics design is not covered in this course.
Has the ability to work independently or as a part of a larger team on a Computer Engineering project or task.	Has solved several hands-on multistep tasks that should help to develop independent work skills. In addition participated in a group project with assigned partners to help the students work on a larger project with colleagues of the same field.

### 3.4. Target Audience

The target audience of the course has changed over time. Initially it was a course aimed at students further in their Bologna Process conformant 3-year bachelor's studies in UT. Later it was moved earlier in study plan for Computer Engineering students. It was also assigned part of the Science and Technology curriculum, which is a similar 3-year bachelors programme. It is a very new curriculum as the first students enrolled in 2016/2017A [9], meaning they had yet to reach their third year by the 2017/2018S semester. The course is compulsory for students specialising in "Bioengineering and Robotics". The following sections also cover the enrolment statistics as it is dual in nature to the target audience.

### 3.4.1. Plan (2017/2018A)

When first invited to run this course the author had a meeting with the programme manager who identified that the majority of students would be from the second and third year of the Computer Engineering curriculum and previous experience showed that there would be some students from other curricula as well. This was used to set initial expectations on the background of the students i.e. most of them would have taken more than one programming course before enrolling in this course. Three lab groups with twelve student capacity each were opened to cater for the estimated 35 students.

### 3.4.2. Act and Observe (2017/2018S)

The final number of students takes some time to stabilise every semester. This paper considers students who got their final grade in the course as these students influence the course most considerably. They are listed in [Table 3](#).

*Table 3. Students with Final Grade (2017/2018S) - Division by Curricula*

<b>Curriculum</b>	<b>Level</b>	<b>2017/2018S</b>
Computer Engineering	1st bachelor's	3 (~10%)
	2nd bachelor's	14 (~47%)
	3rd bachelor's	6 (~20%)
Science and Technology	bachelor's	0 (0%)
Erasmus	bachelor's	1 (~3%)
Physics	doctorate	1 (~3%)
Computer Science	bachelor's	2 (~7%)
Conversion Master in IT	master's	0 (0%)
Physics, Chemistry and Material Science	bachelor's	1 (~3%)
Software Engineering	master's	1 (~3%)
Estonian Aviation Academy	bachelor's	1 (~3%)
<b>Total</b>	<b>All</b>	<b>30 (100%)</b>

The table has some rows with zero students—this is to make the comparison with future semesters more convenient. Computer Engineering students are listed per year as they form the majority of the students and all years are represented. The author acknowledges that

there might be useful information in students who only enrol for a short period of time, but this topic is out of scope for this thesis.

### 3.4.3. Reflect and Plan (Summer 2018)

By the end of the course 30 students received a final grade. The staff of the course were informed mid-semester that the course would be moved to the first semester of Computer Engineering. It was previously known that the first third year students of Science and Technology curriculum specialisation module would also take this course in the autumn. All this added up to a prognosis of approximately 100 students enrolled in the beginning of the semester with a dissimilar background to the spring students. This meant that more teaching staff would be needed and some of the tasks in the labs needed to be fitted to the students experience. These changes are discussed further in [Section 4.4.3](#).

### 3.4.4. Act and Observe (2018/2019A)

From [Table 4](#) we can see that around half of the students completing this course in the autumn were Computer Engineering first year students.

*Table 4. Students with Final Grade (2018/2019A) - Division by Curricula*

Curriculum	Level	2018/2019A
Computer Engineering	1st-year bachelor's	40 (~49%)
	2nd-year bachelor's	18 (~22%)
	3rd-year bachelor's	3 (~4%)
Science and Technology	bachelor's	11 (~14%)
Erasmus	bachelor's	4 (~5%)
Physics	doctorate	2 (~2%)
Computer Science	bachelor's	1 (~1%)
Conversion Master in IT	master's	1 (~1%)
Physics, Chemistry and Material Science	bachelor's	1 (~1%)
Software Engineering	master's	0 (0%)
Estonian Aviation Academy	bachelor's	0 (0%)
<b>Total</b>	<b>All</b>	<b>81 (100%)</b>



First year Computer Engineering students were taking an introductory 6 ECTS programming course in parallel. Students from Science and Technology curriculum had passed a 3 ECTS programming course in their first year and then a couple of courses that made use of MATLAB. Other students with a few exceptions had more programming experience.

### 3.4.5. Reflect and Plan (2018/2019S)

The students of Science and Technology had less background in programming than initially expected. Most of the misjudgement can be attributed to the fact that this was the first ever class of third year Science and Technology students in UT the rest is on the author not having a deeper look into the programming courses they had taken. First year Computer Engineering students did benefit from the changes made in the course. Some additional support on applying the divide → conquer → combine methodology is needed.

Table 5 is a result from discussions with programme managers, students already registered to the course, and the expectation that the number of students from less represented curricula will -on average- stay the same. All in all the table shows comparable expected enrolment to that of 2018/2019A semester with the exception of Computer Science students who have already shown increased interest. Science and Technology bachelors enrolment increased in 2018/2019A from 25 to 35 students [9]. Some of these extra students will reach this robotics course in 2020/2021A semester.

*Table 5. Students prognosis (2019/2020A) - Division by Curricula*

<b>Curriculum</b>	<b>Level</b>	<b>2019/2020A (prognosis)</b>
Computer Engineering	1st bachelor's	40-50
	2nd bachelor's	5-10
	3rd bachelor's	1-3
Science and Technology	bachelor's	12-15
Erasmus	bachelor's	2-5
Physics	doctorate	0-1
Computer Science	bachelor's	10-15
Conversion Master in IT	master's	0-1
Physics, Chemistry and Material Science	bachelor's	1-4
Software Engineering	master's	0-1

<b>Curriculum</b>	<b>Level</b>	<b>2019/2020A (prognosis)</b>
Estonian Aviation Academy	bachelor's	0-1
<b>Total</b>	<b>All</b>	<b>73-106(100%)</b>

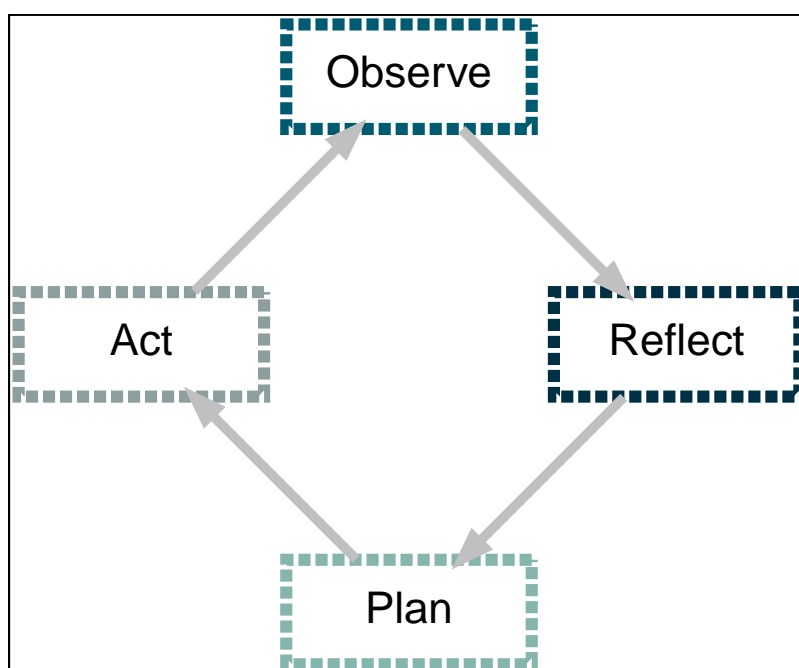
Students from other curricula are planned to be given more attention to make sure the course would cater to their needs as long as it doesn't strongly affect the core of the students. More support in programming is planned for the students of Science and Technology curriculum.

## 4. Course Design

Iterations of the course included in this thesis are based on the broader view from the [Chapter 3](#) and feedback from programme manager and other institute members gathered in discussions. This information is maintained in the instructors repository that will be introduced in [Section 6.1](#). The following sections first introduce the methodology used in the design process and continue to explain how it was applied in the development of selected areas of the course.

### 4.1. Methodology

The design of the thesis follows some of the core ideas of action research (AR). Action research cycles help to trace and explain the improvements implemented in the course. The basic AR-cycle consists of four key steps as described by [10, 11, 12] and shown in [Figure 2](#).



*Figure 2. Basic Action Research Cycle*

This thesis covers two complete AR-cycles and at times uncovers some of the planning activities for the third cycle. The AR concept is applied on two levels. There are semester-long cycles and many sub-cycles within each semester usually aligned with the weekly rhythm of the course. Semester officially starts with an opening meeting for the staff and closes with a staff reflection meeting at the end of the semester. There are also weekly meetings and multiple chat groups to help solve more immediate questions. All of these cycles are visualised in [Figure 3](#).

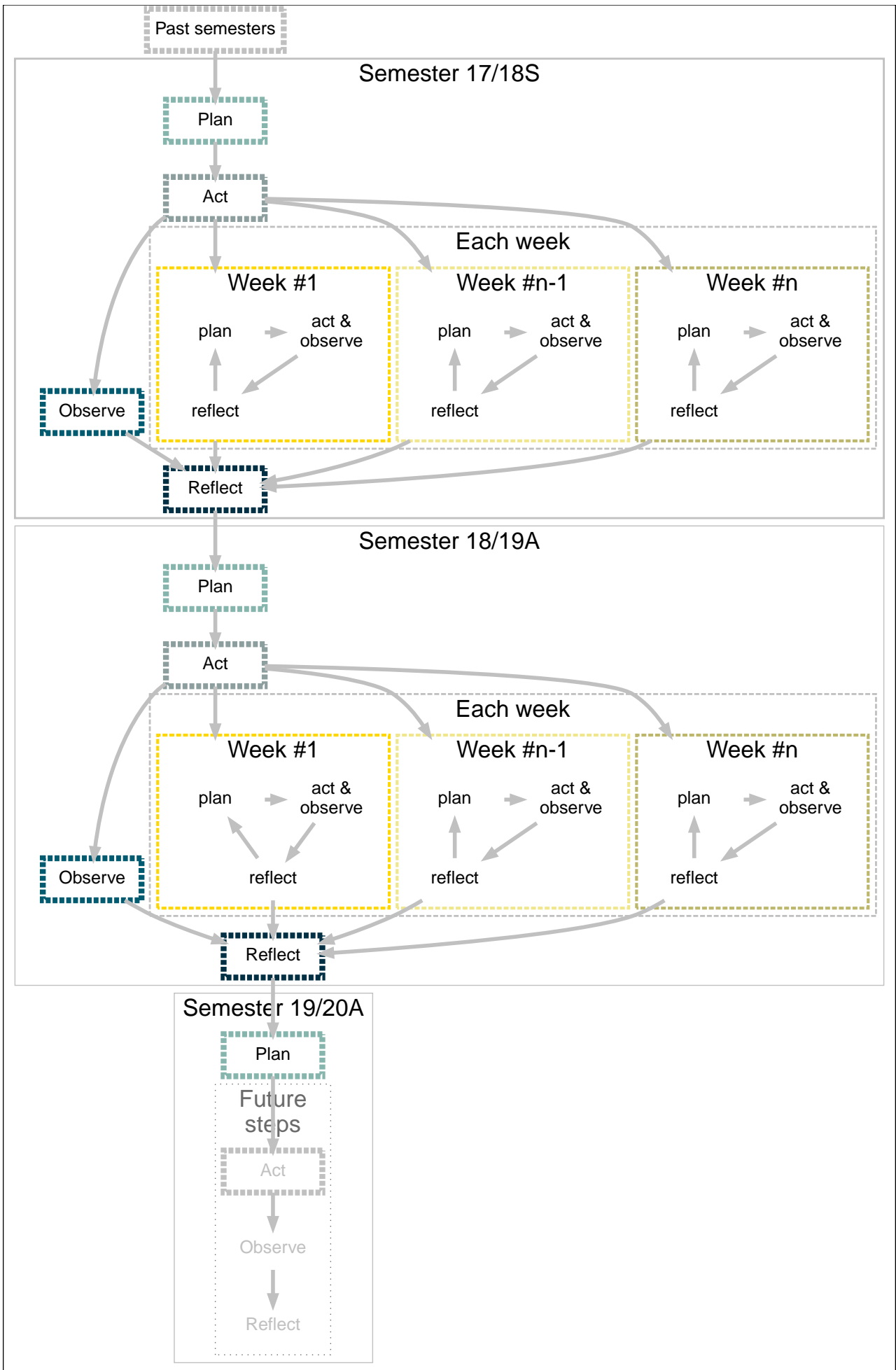


Figure 3. Actual feedback cycles in the course

Starting with 2018/2019A semester official minutes of meetings are available to all instructors. Some more static sections that do not follow this structure are also present in the thesis. The minutes were initially taken by one instructor to develop the general style of the notes, but this task was rotated in the second half of the semester. In the previous semester, notes were taken individually and reminders on the decisions taken were sent out via e-mail as needed. This would not scale well with the increase in the number of instructors and there was no good way for an instructor who was missing to get the gist of the meeting.

The majority of the thesis is structured first by the topic and second by the AR-cycle steps. In the description "Act and Observe" and "Reflect and Plan" steps are coupled together as they withhold a lot of linked content. The common subsection structure is:

- Plan (2017/2018A);
- Act and Observe (2017/2018S);
- Reflect and Plan (Summer 2018);
- Act and Observe (2018/2019A);
- Reflect and Plan (2018/2019S).

The time specification inside () shows the time of the work happening. Plan (2017/2018A) denotes preparations for the 2017/2018S semester happening in 2017/2018A semester. The course has seen and will see a lot of adjustments including to the tasks proposed, hardware used, and theory covered through the employed AR-cycles. This will help to maintain an up-to-date course in a fast growing and rapidly evolving area of robotics.

## 4.2. Grading System and Results

The course uses a grading system based on multiple criteria. One of the core ideas of grading is that for an introductory course we need to cover various talents with our grading scheme as students have a wide variety of backgrounds. Ranging from students who have little to no programming experience to students who have several years of programming experience and have taken part in robotics competitions. Comparable levels of diversity in backgrounds have also been observed in electronics, mechanics, and mathematics all of which support the comprehension of the topics covered in the course. Most of the assessment is summative but some formative assessment is done in labs, lectures, and for the course project.

### 4.2.1. Plan (2017/2018A)

As explained in [Section 3.4.1](#) it was expected that there would be ~35 students participating in the course in the spring of 2018. This is one of the factors considered in the grading and feedback design as they both need time from the staff. During the course design, it was decided that the staff would distribute the activities needing feedback or a grade over the semester? This was done to: firstly, motivate the students to invest at least some time in to the course each week, secondly extending the useful period of formative feedback—helping the student to improve every week, and finally, to make sure students get all of the summative feedback with a shorter delay. The importance of distributing the grading and feedback over the whole course is also supported by researchers Jones, and Gorra, who found: "To conclude, the data suggest that offering all students the detailed feedback on summative work is not resource efficient for academic staff and institutions based on the low number of students requesting and actually accessing detailed individual feedback." [[13](#)].

All of the points awarded in the course are divided into two categories as shown in [Figure 4](#). Each of these categories contains tasks, some of which are covered by the further sections of the thesis. The first category "Base skills" are the items that are expected from all of the students, including:

- **Laboratory Practicals** - 10 compulsory labs (Lab01..Lab10),
- **Course Project** - participating in a 50h/student project conducted in pairs,
- **Week in the News** - a 5 minute presentation given by each student at a lecture about a recent news item in robotics,
- **Exam** - an individual oral examination carried out at the end of the course and based on students own solution code for the labs.

Second one "Specialisation" covers excellence giving students with above average academic abilities an opportunity to express their talents. These include:

- **Advanced tasks** - extra challenges in labs usually extending one of the compulsory tasks,
- **Early Bird Presenter Reward** - extra points for students who take the lead and give their presentations early in the semester,
- **Bug Bounty** - credit system for students who report bugs in the course materials, anything from typos to bugs in example code,

- **Perfect Attendance** - extra points for students who are able to attend all of the lectures — university regulations apply for exemptions, such as health issues.

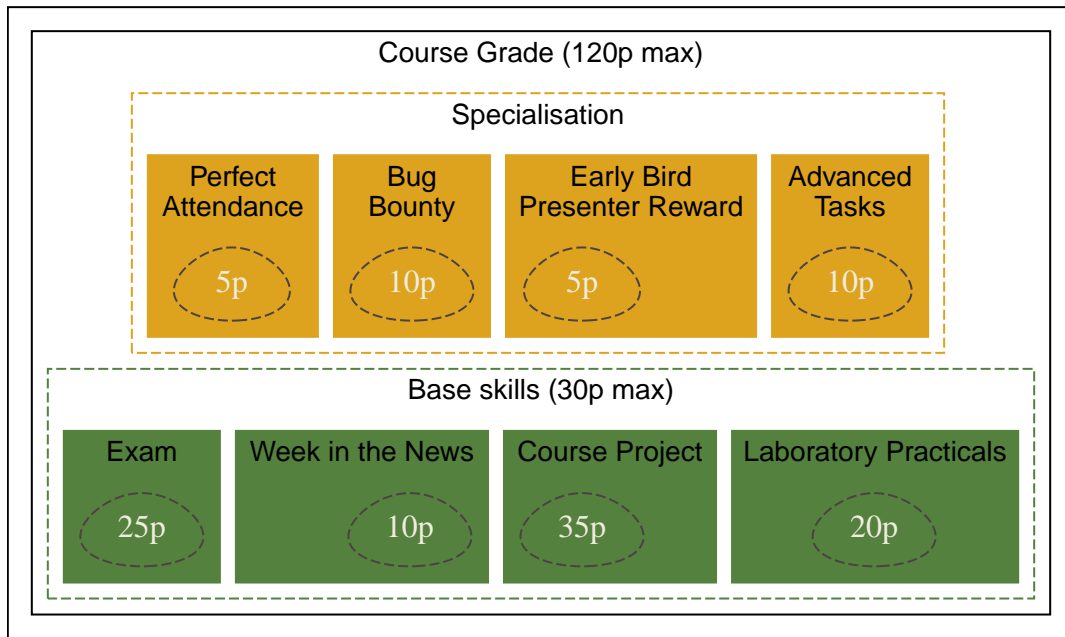


Figure 4. Score categories used for the semester (2017/2018S)

Each of these categories contains tasks, some of which are covered by the further sections of the thesis.

The score system, amongst other things, aims to motivate students to submit their work as soon as possible, even if they have missed a deadline. This is achieved via initiating some of the point categories, especially for the course project, with a half-life and an exponential decay process.

Let's consider the following example. The students have a deadline for presenting the proof of concept (PoC) for the course project on May 3rd 2018 23:59. The maximum number of points a PoC can be awarded is 5 points denoted by  $P(0)$ . There is a missed deadline penalty of 1.5 points denoted by  $MDP$ . The points remaining after missing the deadline are  $P(0) - MDP = 3.5$  points and will be denoted  $N_0$ . These points are listed to have half-life of 168h which is 7 days denoted by  $half-life$ . There is a lower bound of 0 points for each task. The number of hours past deadline is rounded up and denoted by  $t$ . The points remaining after  $t$  hours have passed are denoted by  $P(t)$  and calculated as follows:

$$P(t) = \max \left\{ \frac{N_0}{2^{\frac{t}{half-life}}}, 0 \right\}.$$

Let's suppose a student misses the deadline by 50 hours—the points remaining for this task are:

$$P(50) = \max\left\{\frac{3.5}{2^{\frac{50}{168}}}, 0\right\} = \sim 2.85p.$$

Points gained from laboratory practicals do not hold the exponential decay property. They just have two deadlines which [Section 4.3](#) discusses in more detail. Missing the first results in losing half of the points and missing the midterm results in 0 points. [Figure 5](#) shows that as time passes, less and less points are on offer for the students. This means that the course will get gradually harder to pass if no action is taken by the student. At the same time it is still theoretically possible to get the grade A if the student starts work in April. It is just considerably harder than when they would have started in the beginning of the semester.

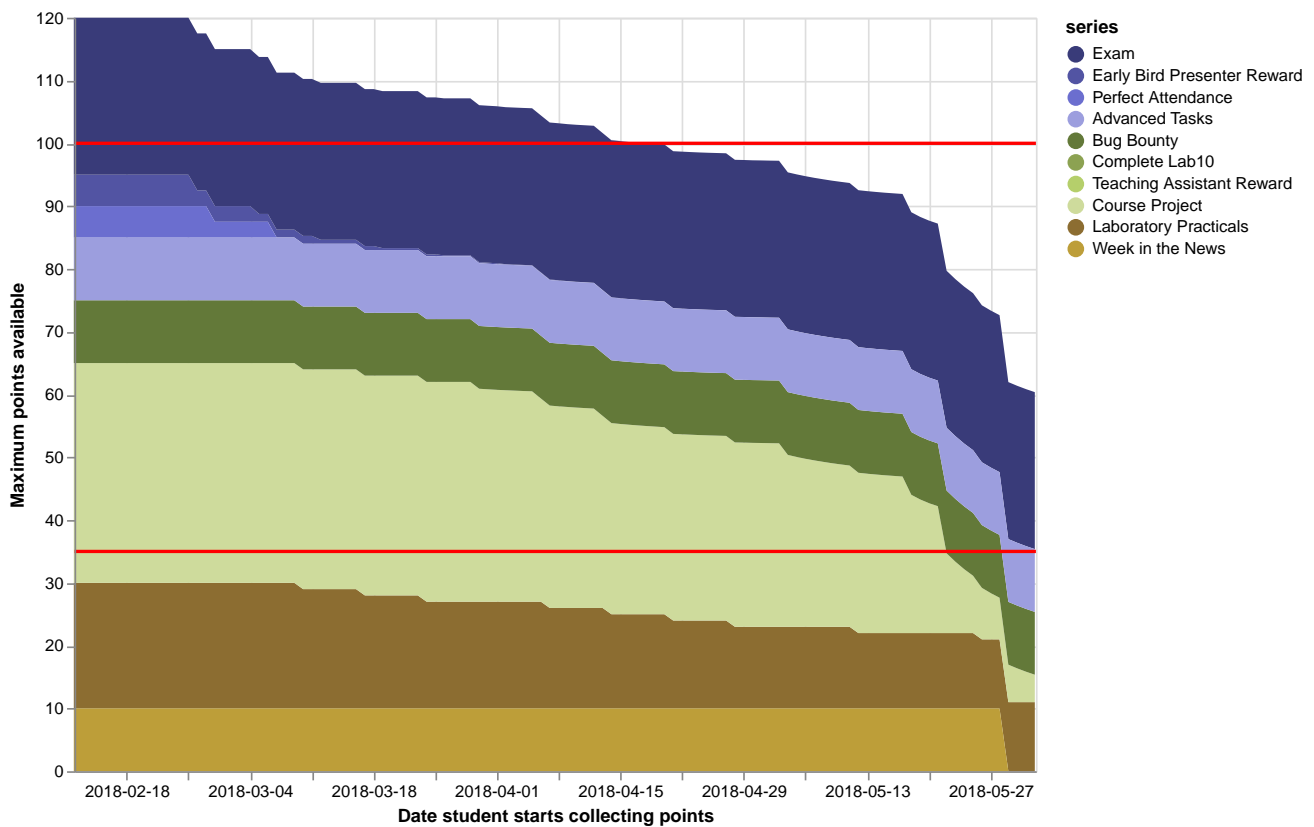


Figure 5. Points decay over the semester assuming that the student has not taken action until that point in time (2017/2018S)

#### 4.2.2. Act and Observe (2017/2018S)

Grading system was introduced in the opening lecture and parts of it covering the course project, midterm, and exam were explained multiple times over the course of the semester. Students are presented a simplified Python function visible in the [Code Example 1](#) to show how the points are converted at the end of the course. The students were also explained that



this conversion would only be done if the student has met all of the minimum requirements to pass the course. This format was initially chosen to remind students of Python syntax and to help communicate the amount of programming waiting for them in the course.

#### Code Example 1

```
1 def grade(points): ①
2     if not isinstance(points, int): ②
3         return 'Contact instructor'
4     elif points >= 91:
5         return 'A'
6     elif points >= 81:
7         return 'B'
8     elif points >= 71:
9         return 'C'
10    elif points >= 61:
11        return 'D'
12    elif points >= 51:
13        return 'E'
14    else: ③
15        return 'F'
```

- ① Input is the total number of points the student has at the end of the course
- ② To explain that input to a function is not always what one would expect it to be
- ③ Define grade F in an `else` statement to cover the whole range

An individual oral exam was conducted with all students who passed the 35 point bare minimum before the exam. The exam was based around students randomly choosing two tasks from the labs and explaining their solution code to a panel of three: two instructors, and the author. Tasks were categorised before the exam to lessen the effects of chance. All of the tasks in the labs were divided into three categories for the exam: tasks that were short and easy (Group A), tasks that were tricky and/or long (Group B), and tasks that were not suitable. The division of tasks was based on a majority vote with each instructor having one vote. All instructors graded all tasks and the lecturer broke ties where necessary. Each student would draw one task from Group A and one from Group B as part of the exam.

### 4.2.3. Reflect and Plan (Summer 2018)

The initial wording of the score categories were confusing to a couple of students according to overall course feedback in the SIS. The problem seemed to revolve around the use of word "bonus" in some, but not all of the specialisation level activities. These names were updated

for the next semester to avoid possible confusion amongst future students. The final grade distribution, based on the criteria described in Section 4.2.1, is presented in Figure 6. It shows that half of the students got the grade A or B and that one in five students did not manage to pass the course achieving an F or "not present" (np). The figure contains one grade per student, for students who participated in the resit the grade is from the resit.

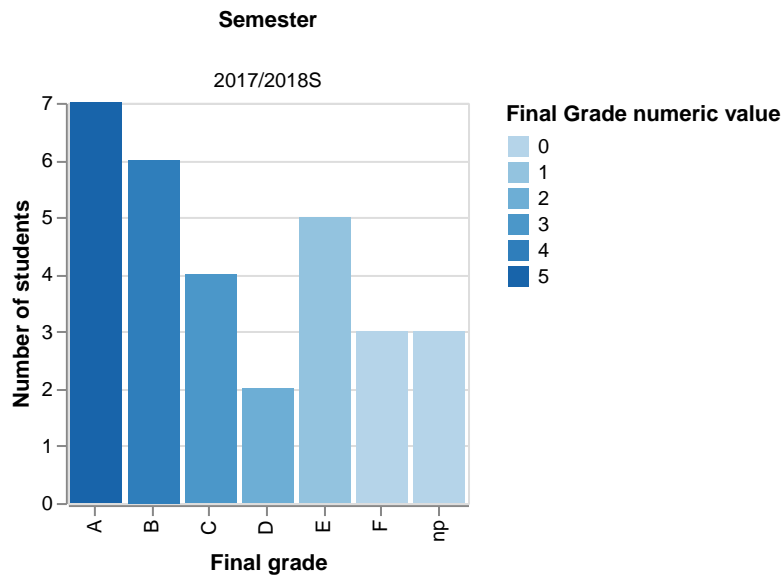


Figure 6. Students' final grade division (2017/2018S)

Figure 7 lists the categories planned and used for the 2018/2019A semester.

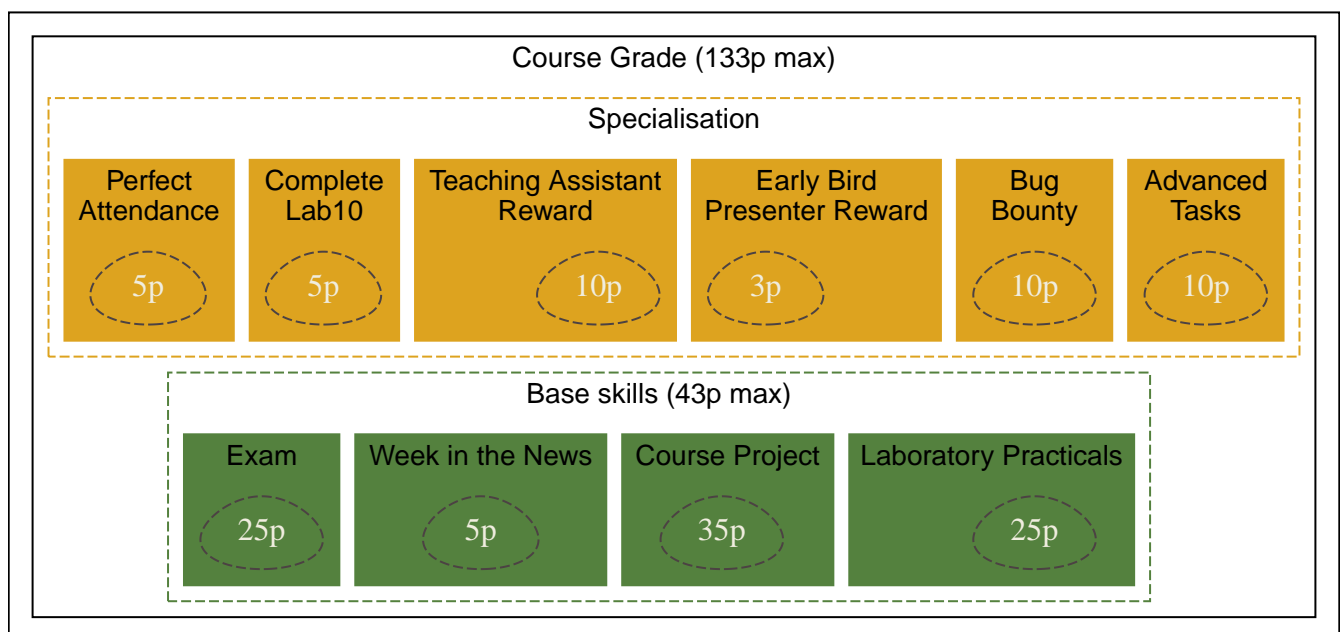


Figure 7. Score categories used for the semester (2018/2019A)

The course needed an update to the grading as the level at which students entered the course was expected to drop. This expectation was based on the course moving to the first semester from the regular fourth semester for Computer Engineering (CE) students. An introductory

Laboratory Practical number 00 (Lab00) was added, it included hands-on experience with breadboarding, usage of Ubuntu Linux operating system [14], use of Git distributed version control system (DVCS) [15], and an introductory programming task. Lab10 was moved to the specialisation tier, as it required implementing algorithms more advanced than other labs and this kept the number of compulsory labs at a total of ten.

Teaching assistant reward was added to the course to engage more experienced students in learning by helping others in their lab. This process was supervised by the corresponding instructors in the lab. This was an invite only opportunity for students who excelled in their labs to join another lab group and assist the instructors in answering students questions, they would have no role in the assessment process which was managed by the regular instructors. For assisting one 4h lab the students would get 2p. It was hoped to reduce the classroom pressure on the teaching staff with entry level questions from first year students. The Early Bird Presenter Reward points were reduced from five points to three points. Five points were transferred from the Week in the News presentation to the Laboratory Practicals to better reflect the expected input for each activity.

The overall idea of rewarding timely, or as soon after the deadline as possible, seemed to work well based on the feedback from the instructors. Some modifications were made to introduce the idea into new areas of grading, such as the midterm described in [Section 4.3](#). [Figure 8](#) shows that the decay in points available is greater than in the previous semester—being a direct result from the changes made.

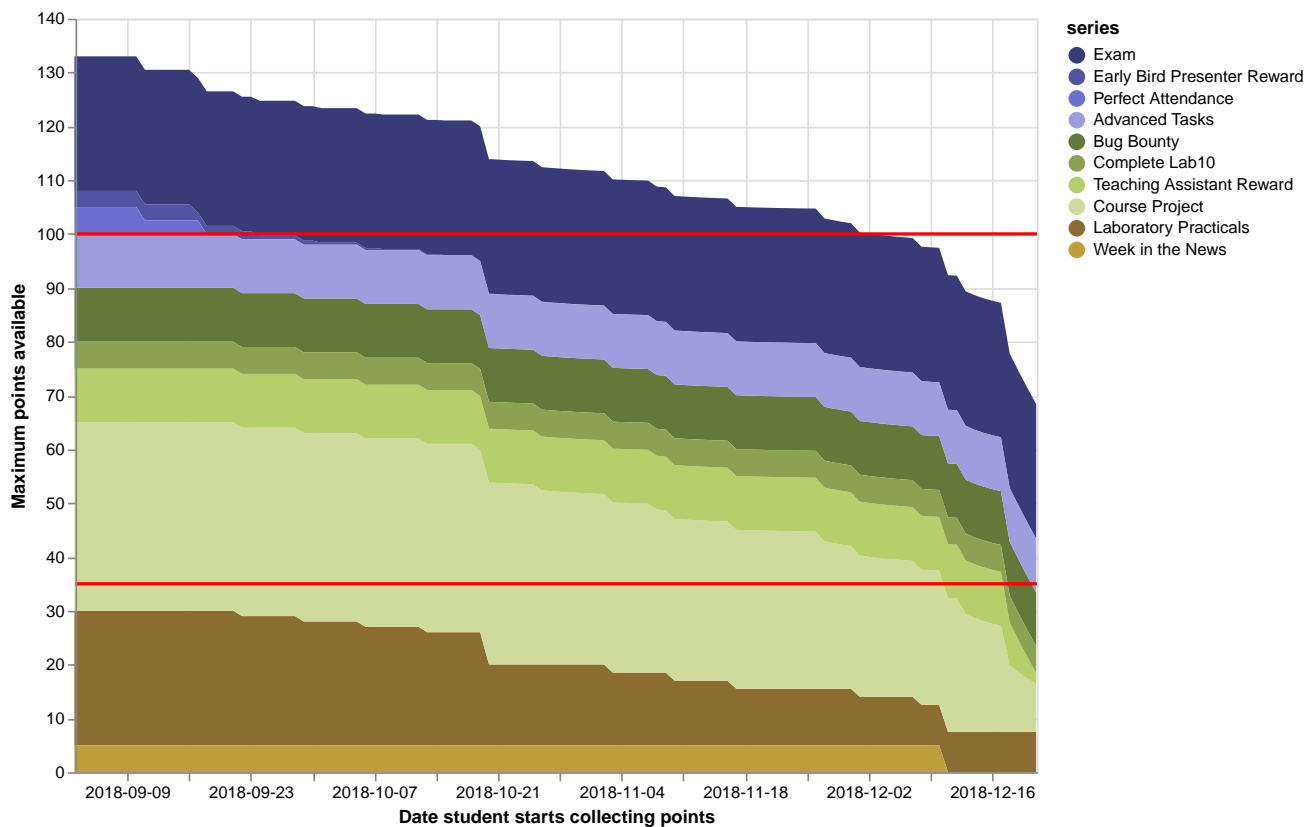


Figure 8. Points decay over the semester assuming that the student has not taken action until that point in time (2018/2019A)

It is also worth noting that the maximum number of points increased from 120 to 133. The points added were all part of the specialisation tier. All this combined means that a student would still be able to get the A grade when starting in second week of December.

#### 4.2.4. Act and Observe (2018/2019A)

The instructors reported at a weekly meeting that Lab00 was of suitable complexity for first year's students, and was considered easy by more experienced students. A total of eleven students were offered the opportunity to gain additional points via the Teaching Assistant Reward. Eight students accepted the offer and a total of 24 labs were assisted. Two students achieved the maximum result of 10p in the category by assisting on five occasions And two of the students decided to help out on one occasion. Other four students assisted on two to four occasions.

#### 4.2.5. Reflect and Plan (2018/2019S)

Course grades seem to indicate the expected decrease in grades as we transitioned to having many first year students. The division of grades is shown in [Figure 9](#).

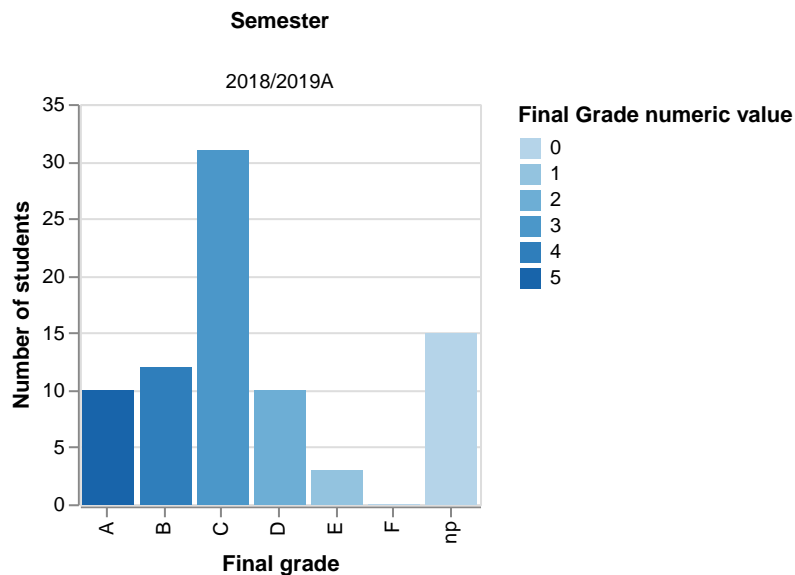


Figure 9. Students' grades division.

Only one of the students who passed the 35 point minimum requirement for taking the exam failed to pass the exam after the resit had taken place by not showing up at the resit. Some students sought additional help on learning how to read their own code between the two exams. This might suggest that an additional activity in the labs were students would need to explain their code might be beneficial. This would also help to push the students to write more of their own code.

## 4.3. Lab Manual Release Schedule

This section covers the process of releasing lab manuals as one of the main sources for weekly tasks for the learner. There have been several modifications to the releases mostly guided by student progress. Digital Timing Diagram (DTD), a common visualisation method for representing timing relations in digital logic, is now used to convey the information to the students within the semester. These diagrams are produced by a tool named WaveDrom from WaveJSON compact exchange format based on JavaScript Object Notation (JSON) language syntax which has become the de facto tool for building DTDs [16].

### 4.3.1. Reading Digital Timing Diagrams

A representation for the time-step is needed. This is very straightforward — a clock signal is used for this as represented in Figure 10. The numbers on top of the rising-edge of the clock represent the start of the corresponding academic week. Period of the signal is chosen to be one week to make sure that all of the weeks look the same instead of using two week periods that might communicate that there is a difference between high and low clock signal weeks. The use of clock signal also conserves the appearance of traditional DTDs which helps the

students to make a habit out of referencing all readings they take to the clock.

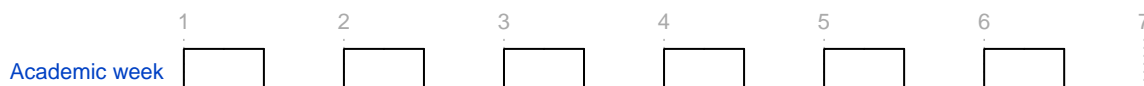


Figure 10. Release schedule - 6 academic weeks

A representation of the points available for a particular lab is also needed. Another signal is used for this purpose as shown in Figure 11. High state of the signal representing maximum points for the lab, low state: zero points, and middle state: reduced points.

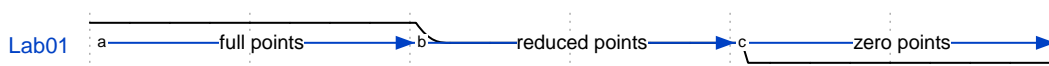


Figure 11. Release schedule - points available

Figure 12 displays a signal showing that at first Lab01 is released then there is a week long pause after which two more labs are released. This information can also be derived from Figure 11 but not as conveniently on more complex diagrams. The slightly curved tip at the end of Lab01, and Lab03 as opposed to the blunt tip at the end of Lab02 does not hold any special meaning for our application.



Figure 12. Release schedule - labs released

DTDs have been used since 2018/2019A to represent information about the labs in this course. DTD for the previous semester has been retroactively created for the benefit of making comparison between semesters more convenient.

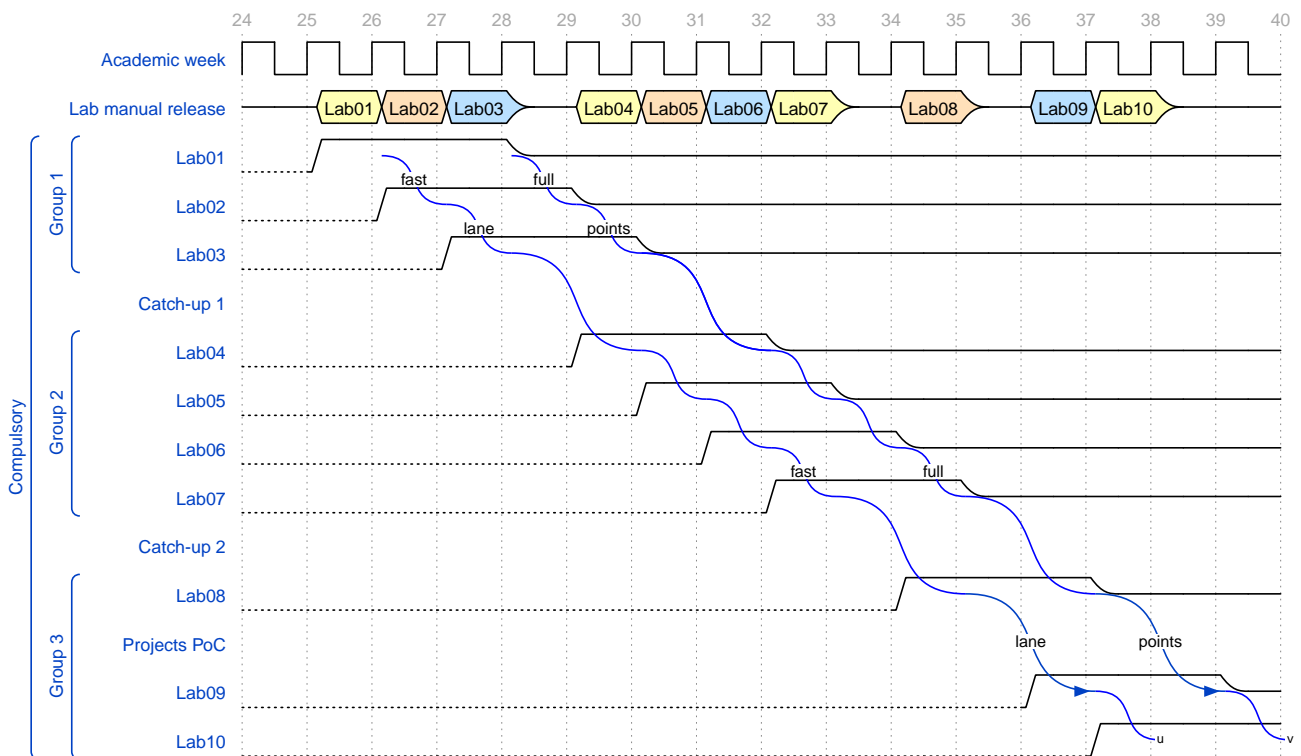
### 4.3.2. Plan (2017/2018A)

When the planning process started there was some feedback available from the students who had participated in the course on the previous year. One of the things that stood out was that most of the manuals were just there. The students were asked to complete them in a particular order but there was not much other structure around them. None of the staff of six involved in the planning processes had been involved in teaching this course on the previous semester, but three had recently participated as students. It was decided that the upcoming semester would use step-by-step release schedule for the lab manuals. There were multiple reasons for this approach. From the students perspective the main one was to spread out the workload to the whole semester giving the students a guideline on how the course progresses. From the instructors perspective it helped to spread the workload of

improving and redesigning the lab manuals. It also helped to adjust the level of difficulty based on the students progress in the earlier labs. This also meant that the development cycles could be easily repeated where the need arose as there was always at least one instructor who had recently worked on improving the manual in question. It was agreed that each instructor would be responsible for two lab manuals as we had five instructors and ten lab manuals to cover. The author would work with each one of them to fit the lectures to the labs, on the manual template covered in [Section 6.2](#), and to act as an extra pair of eyes to read through each manual before release.

### 4.3.3. Act and Observe (2017/2018S)

During the semester the lab manuals followed the release schedule shown in [Figure 13](#). This diagrams has two additional lines marked with **fast lane** and **full points**. Fast lane is for students with more background in the area wanting to challenge themselves by submitting solutions to tasks on the week at which they were released. Full points is a reference for students who are aiming to gain all points from the labs.



*Figure 13. Lab manuals' release schedule and points availability as played out in the 2018/2019S semester (retroactively added visualisation).*

The staff needed to be flexible in adding supporting text in the manuals as problematic areas became apparent as one or two instructors testing the manual was not sufficient to find all of the problems. Some alterations were made in the release dates as the course progressed and the comfortable pace for the students became more apparent. This was also the first

semester for considerable amount of the subtasks and even for some complete labs. Student progress review conducted at the weekly instructor meetings played a key role in the process of setting a suitable pace.

#### **4.3.4. Reflect and Plan (Summer 2018)**

In the end of semester meeting the staff collectively agreed that the students seemed to lose interest in completing a lab if they had already missed the soft deadline and half of the points were deducted from the maximum available for the lab with the remainder being available until the end of the semester. This issue was analysed and a mid-term deadline was proposed to motivate students in going through the labs in a timely manner. Feedback from the instructors meeting indicated that the catch-up week worked well the first time. It was proposed that the reason for this was that the extension was unexpected to the students. The instructors also felt that the second catch-up did not have such a clear impact on the students course progress as many opted to miss the lab on the catch-up week. It was decided that a compact representation of release dates and deadlines might benefit the course. The author started work with DTDs to present this information.

Another point made at the instructors meeting was that as the manuals became more advanced they also got more dependencies outside the basic AsciiDoctor install for compiling the documents. This together with the increase in the number of instructors described in [Section 4.4.3](#) meant that there would be more overhead in making small fixes to the manuals. It was decided that a CI server building manuals on each push would help the reduce this overhead. Setting up a CI server would need a server and a change in the branching model used for the instructors repository. These changes were added the roadmap but it was unlikely that all this could be done for the upcoming semester.

#### **4.3.5. Act and Observe (2018/2019A)**

[Figure 14](#) was created and introduced in the opening lecture and reiterated and improved throughout the semester. There was some initial confusion with reading the diagram as many students had no prior experience with this type of diagrams and they are not commonly used for this sort of information. It was still beneficial for the students to practice reading these as they would need to use sensors that had documentation on usage represented by DTDs e.g. the ultrasonic sensor [17].



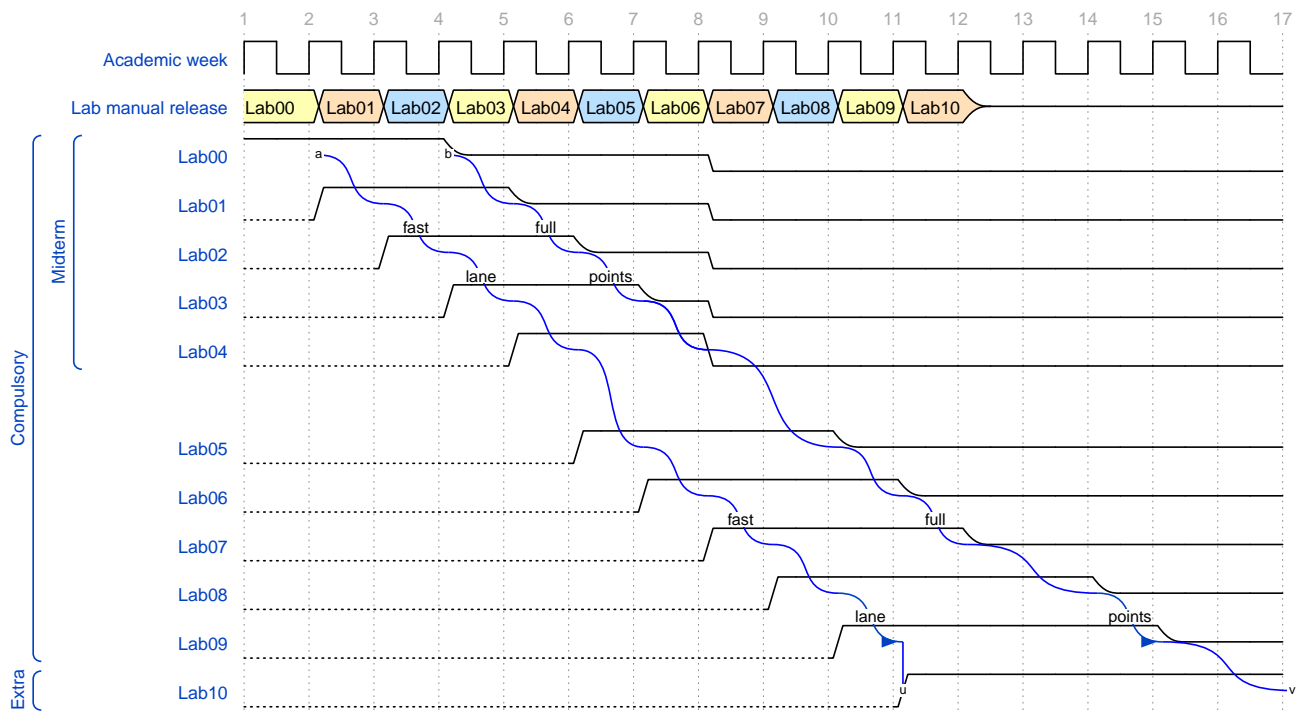


Figure 14. Lab manuals' release schedule and points availability as played out in the 2018/2019A semester.

Catch-up weeks were not explicitly used. Instead the deadlines for the labs after the midterm were extended by a week or two. The midterm was confirmed to be a success by the instructors as the majority of the students (72 out of 81) pushed to get the first five labs completed.

#### 4.3.6. Reflect and Plan (2018/2019S)

In this feedback session it was agreed that considerable amount of effort has been put into structuring the student side of the release schedule and it has shown positive results. Lead by this it was agreed that the team will aim to improve the instructors side of the release schedule as well. The approach of pairing up instructors for the review process is going to be tested. The midterm ended up being a make-or-break type barrier for some students. Four of the nine students missing the deadline did not present any work after this deadline. There were also two students who decided to leave the university before the semester ended—meaning they did not receive a final grade and are not part of the grades in Figure 9. Nine students out of eighty-one failing the midterm was within expected margin according to the programme manager.

## **4.4. Instructors Workload**

### **4.4.1. Plan (2017/2018A)**

The main aim for this semester was to hit the ground running as all of the teaching staff was replaced. Continuity still prevailed as some of the instructors had recently participated in the course as students two of them in the previous semester. The author also had previous teaching experience with the course. A list of all of the tasks that needed to be done in order for the course to succeed was created at one of the pre-semester meetings—it was acknowledged that we will not be able to catch them all but an attempt was made.

An example of larger tasks that needed to be solved as soon as possible were listed (smaller tasks not listed for brevity):

1. initiate staff Git repository;
2. lab manual;
3. lab manual release schedule;
4. hardware inventory;
5. instructor division to lab groups;
6. laptops for the classroom;
7. deciding on the operating system used for the labs;
8. exam outline.

### **4.4.2. Act and Observe (2017/2018S)**

The list of things that needed to be done was reviewed at weekly meetings and was not exhausted by the end of the semester. One of the list items after a couple of weeks was to create a separate list for improvements that were collectively agreed to be worth to consider but too time consuming to implement within the semester.

### **4.4.3. Reflect and Plan (Summer 2018)**

Looking back the staff concluded that a lot had been learned and that we needed to implement a better system to spread out the work over the semester to smooth out some of the peak demand at deadlines. It was also apparent that the number of students will grow in the next semester giving another reason to space out the work.

It was agreed that the upcoming semester will mainly focus on scaling the course and implementing as many of the improvements listed on the previous semester as possible. The course moved from having 5 instructors to 12 instructors and from 30 students to nearly a 100 students registered by the end of August. Out of the five instructors teaching in the spring semester two were unavailable for the autumn semester, this further complicated the situation and meant we had to recruit and train nine new instructors over the summer. It was decided within the institute that eight groups would be opened for the students. The course is designed to give a lot of direct experience to each and every student via Laboratory Practicals, Course Project, Week in the News presentation and use of a large number of new tools and technologies. This in turn means that the students also need a considerable amount of one-on-one support adding to the importance of spreading out the work over the whole semester to avoid overloading the teaching staff.

### **4.4.4. Act and Observe (2018/2019A)**

The author created a spreadsheet to divide all of the tasks that needed to be done between the instructors. This helped to better divide the work between 12 instructors who each had their specific needs and contracts. New tasks were assigned to the instructors that had less responsibilities and an appropriate background for the tasks at hand. Some instructors got special tasks that ran through the whole semester e.g. looking after the mechanics of the robots or keeping track of various spreadsheets that needed weekly updates.

2018/2019A semester

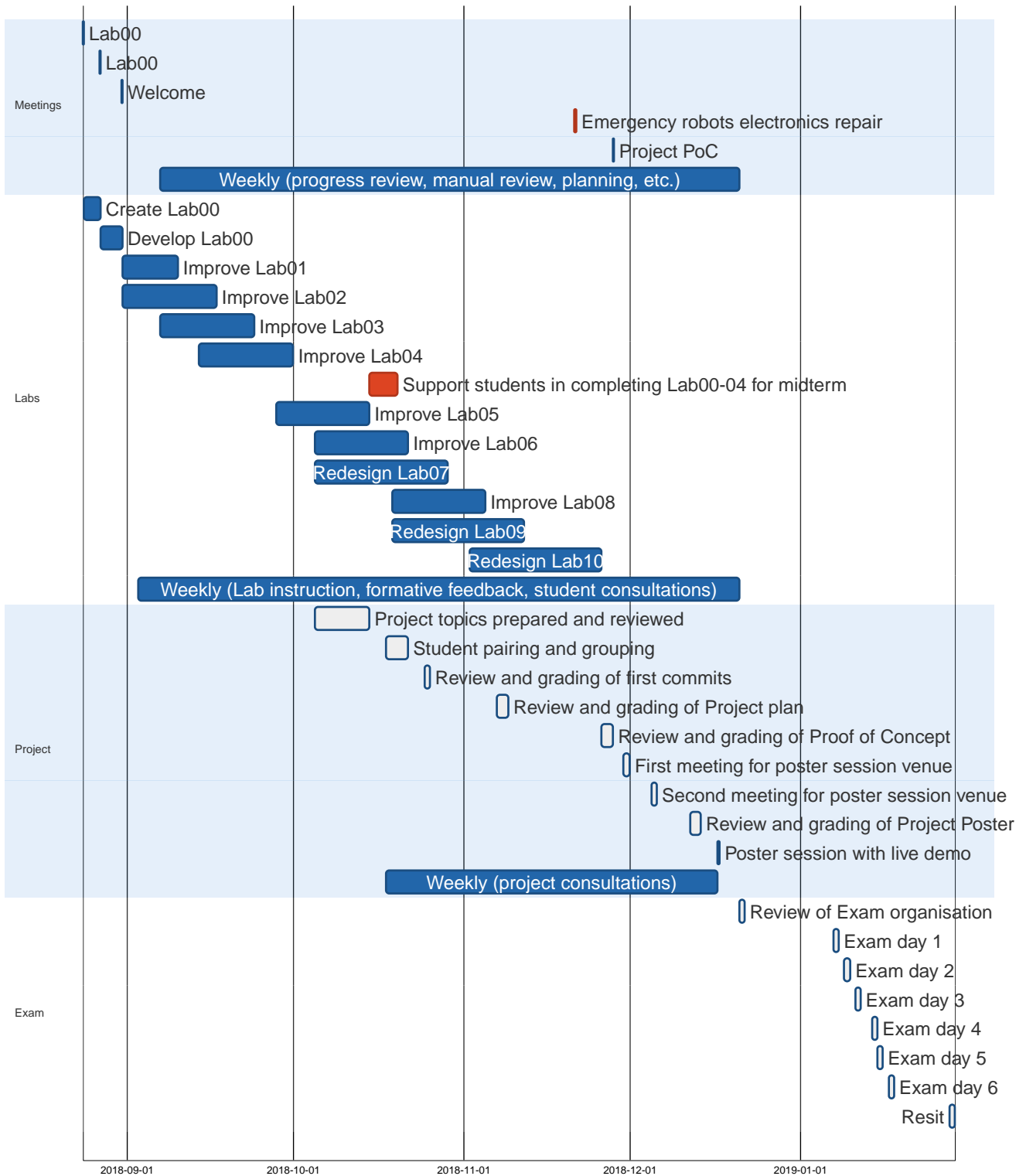


Figure 15. Instructors' workload distribution in the 2018/2019A semester.

#### **4.4.5. Reflect and Plan (2018/2019S)**

From [Figure 15](#) one can observe that there are activities happening all the time but there are still some very busy times — such as the midterm or the end of November when the projects are taking shape. Another problem was the unplanned failure of robots electronics which peaked in the second half of November. Although anticipated that we will reach the expected lifetime of some components under the heavy usage, it was still hard to plan for the failures. It was agreed that unplanned maintenance and repair will be considered as a task for one or two of the instructors from the beginning to make sure that we have someone to quickly respond to the issues. These and many others suggestions for improvements will be organised into a priority queue and assigned a responsible instructor at the next staff meeting in June 2019.

# 5. Student-Facing Solutions

The term "student-facing" is derived from business software development where terms customer-facing and internally-facing are used to differentiate between internal and non-internal applications. The following sections cover various technology that students made use of in this course. Firstly the course used Moodle as the central dashboard for all of the information for the students. Secondly the students mostly used laptops provided by the staff to solve labs and tasks within. The course introduced students to a considerably large set of tools which are required for completing the course. These ranged from languages and libraries to development tools and operating systems (OS). Thirdly one of the tools (OpenCV) is covered that needed effort from the author to enable the students to use more recent versions than provided by the OS repositories. Finally, the use of version control systems is explained and justified.

## 5.1. Moodle Dashboard

The course uses multiple platforms and technologies. Learning Management System (LMS)--Moodle [18] is one of these technologies. It is used as a central hub for all of the course related information and resources for the students. Moodle is the de facto LMS in UT. This section will not follow the regular structure of having separate subsection per AR-cycle. Most of the content and structure was added early in the planning stage and improvements followed the weekly cycles described in [Section 4.1](#) rather than the semester long ones. Most of the updates were time critical and needed less than one hour of work from the lecturer. The only updates that followed the long cycles were updates to grading system described in [Section 4.2](#)

The content in Moodle is organised into six sections. In general the idea is that sections that are more often needed will come before the others e.g. section "Laboratory Practicals" is before the section "Lecture Slides" as they regularly need to access lab manuals. The exception being the very first section which we want the students to notice rather than hope that they know to look for—sign-up sheets. Students also get their points and feedback through Moodle although other systems are used internally to track and assess students. This is discussed in [Section 6.3](#) on Google Docs.

Table 6. The Six Sections of Moodle

#	Name	Section content
1	The Main Panel	<ul style="list-style-type: none"> <li>• Forums</li> <li>• Registration forms for presentations, open seats in labs, etc.</li> <li>• Syllabus info</li> </ul>
2	Laboratory Practicals	<ul style="list-style-type: none"> <li>• Lab manuals</li> <li>• Timing diagram for points availability</li> <li>• Manual release DTD <a href="#">Figure 14</a></li> </ul>
3	Lecture Slides	<ul style="list-style-type: none"> <li>• PDF version of the slides</li> <li>• Meta information on the slides (last edit date, size, format)</li> </ul>
4	Week in the News	<ul style="list-style-type: none"> <li>• Information on rules and deadlines</li> <li>• Presentation Datadrop</li> <li>• Previous presentations</li> <li>• Example sources for presentations</li> </ul>
5	Course Project	<ul style="list-style-type: none"> <li>• Link to topics, rules, schedule and points</li> <li>• Course project registration form</li> <li>• Poster template</li> </ul>
6	External Sources	<ul style="list-style-type: none"> <li>• Pages that are useful but not managed by the staff</li> <li>• Documentation</li> <li>• Proportional–Integral–Derivative (PID) controller loop simulator</li> <li>• Linux terminal usage</li> </ul>

Although Moodle contains many tools it is still only used as a central dashboard for information as in many cases there are more convenient tools for the task. Grade management is one example. Marking completion of many large tasks is not time efficient

with the user interface (UI) provided by Moodle. There is a proposal from the instructors from the 2017/2018S semester to develop a tool using Google Spreadsheet application programming interface (API) and Moodle API to enable semi-automated transfer of awarded points to Moodle. This has yet to be implemented.

## 5.2. Software Stack

It was clear from the beginning that the course would make use of a set of software for the students to solve the task with and base their solutions on. Operating Systems, development tools and libraries would be at the core. Part of the software was OpenCV which took a lot of separate work and is mostly covered in [Section 5.3](#)

### 5.2.1. Plan (2017/2018A)

The course started off with using almost four operating systems, some very closely related. First two used on the Raspberry Pi (RPi) controlling the robot being Raspbian and Raspbian for Robots which is a Raspbian with a customisation and integration layer developed by Dexter Industries [19]. Raspbian is an operating system highly optimised for ARM chips found in the RPi [20]. The other two being Ubuntu 16.04 Long Term Support (LTS) and Microsoft Windows 10 Enterprise available as a dual-boot on the laptops in the classroom.

The connecting link between all but one of these OS's is Debian Linux [21]. Both Ubuntu and Raspbian belong to the Debian family of Linuxes and follow some version and developments while adding value to their distributions. Ubuntu has releases every six months, with LTS releases published every two years [22]. Debian documentation explains the following: it uses an approach with having three different release branches. Every Debian release follows the stages: → unstable → testing → stable. The release in unstable branch is always code-named Sid, when the release is mature enough and sufficient time has passed from previous release, it moves to the testing branch and gets a dedicated code name. After all release critical issues found in the testing process have been fixed in about 7 months the release reaches stable branch. Current stable release is Stretch and the next one in testing is Buster [23]. Raspbian is directly bound to the Debian version with many optimisations to ARM.



### 5.2.2. Act and Observe (2017/2018S)

Windows 10 was offered as the well known alternative to Ubuntu on the classroom laptops. The students were free to choose one operating system for the laptops and one for their individual Micro-SD card. Many of the students opted to use Windows on the laptops and it was about half and half on using pure Raspbian with the GoPiGo2 library or the Raspbian for Robots. [Figure 16](#) visualises the software used on the laptops, including the operating system, development tools and libraries. Python 2.7 is considered a thing of the past and legacy systems in software development and has been given an End Of Life (EOL) by 2020 a long time ago [24]. Some hardware libraries still depended on Python 2.7 so students were advised to use this branch.

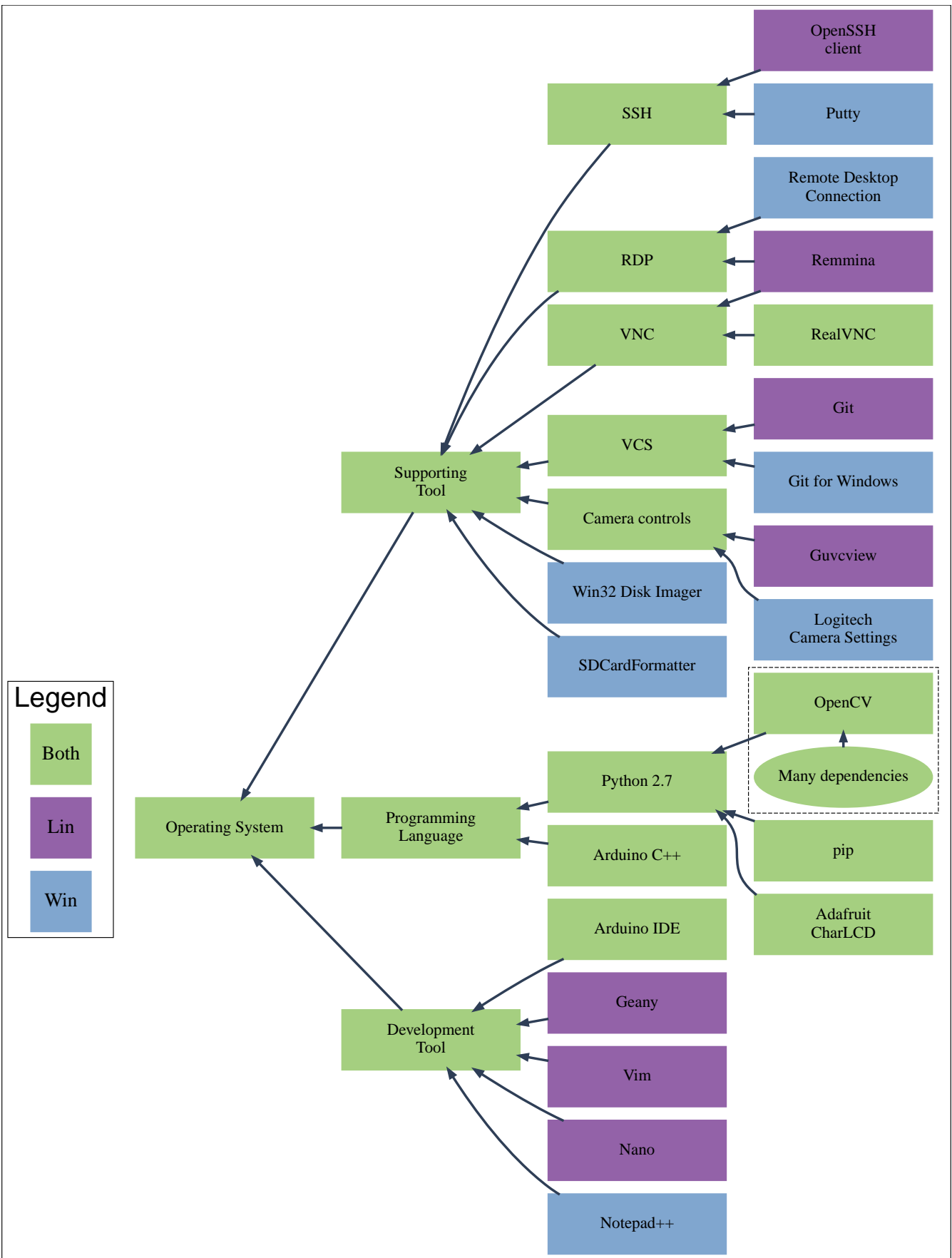


Figure 16. Software stack on laptops (2017/2018S)

### 5.2.3. Reflect and Plan (Summer 2018)

Both virtual network computing (VNC) and remote desktop protocol (RDP) were used in the labs as a later manual needed VNC. Students did have some VNC connectivity issues with the initial server used on RPi so this was changed in the middle of the semester. The freedom to choose between Windows and Linux ended up being highly skewed towards Windows. Students had lower than expected level of experience in using the Windows platform for more advanced tasks which did put extra stress on the teaching staff. There were also situations where the solutions did not end up being OS-agnostic meaning that two separate solutions were needed. It was decided after a lot of discussion that the course would officially drop support for Windows platform to give students a more homogeneous experience. This change was in line with the objective to make the course easier to follow considering the target audience change discussed in [Section 3.4](#).

The need to use Python 2.7 created a bit of confusion especially because students had other Python versions installed for other courses. The lecturer helped two groups who had a problem where they had installed libraries using various methods. In one case the student had 3 different Python installations collectively containing various tools necessary for their project—although an extreme example it exemplifies the problem of using multiple Python versions. Using RDP and two different VNC servers was too much overhead — it was decided that all of the labs would migrate to a single VNC server.

## 5.2.4. Act and Observe (2018/2019A)

Figure 17 visualises the software used for this semester.

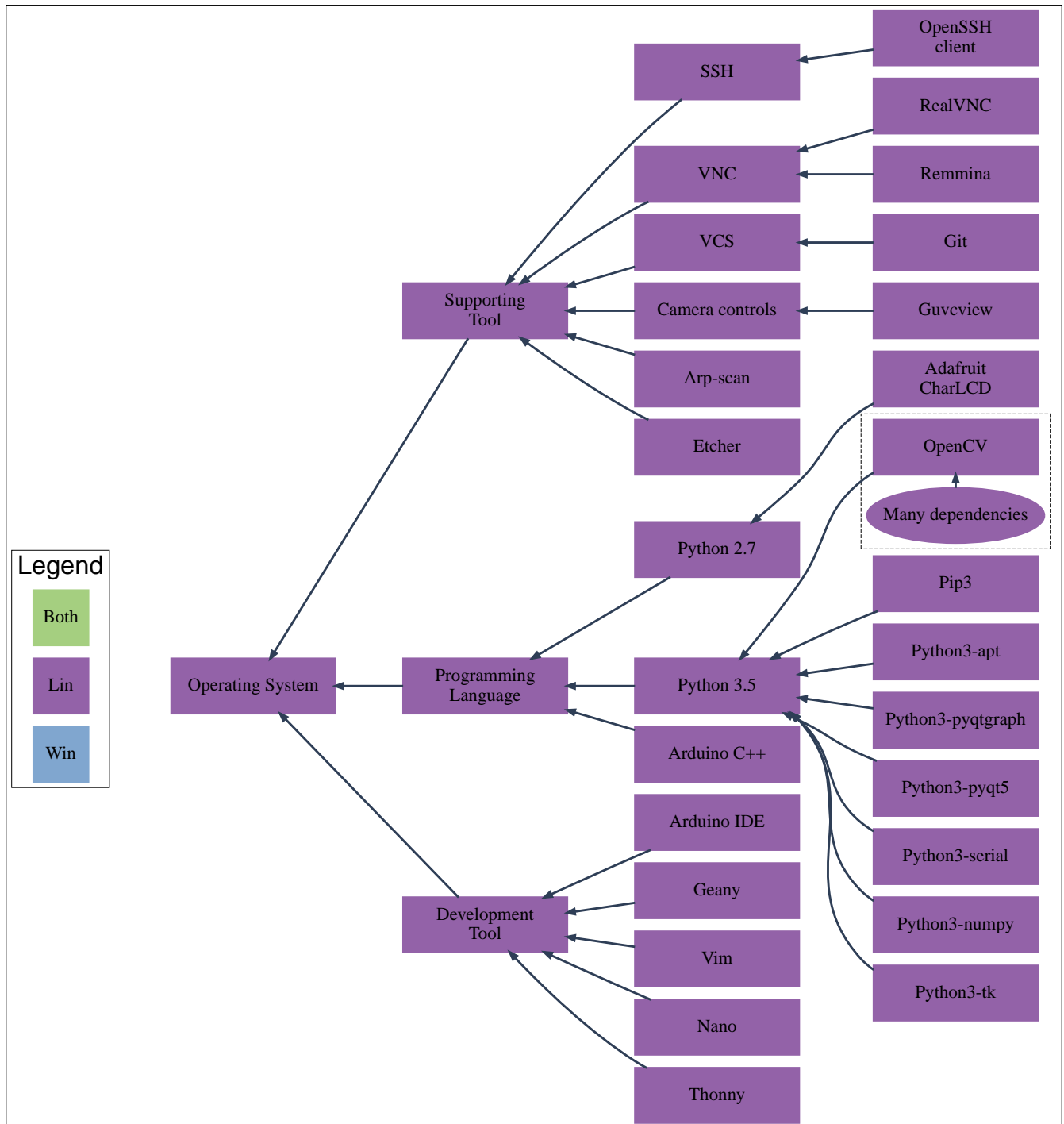


Figure 17. Software stack on laptops (2018/2019A)

Windows is not used for this semester and the transition from Python 2.7 is in progress with only one library remaining for Python 2.7.

### **5.2.5. Reflect and Plan (2018/2019S)**

The software stack has evolved considerably over the two semesters of teaching. Dropping official support for the Windows platform gave the students a more uniform experience in class and one less variable to consider when debugging. Ubuntu UI was introduced in Lab00 to make the transition process easier for the mostly Windows centred students.

The last Python 2.7 dependent library Adafruit Python CharLCD has been deprecated as of November 2018 and replaced by a driver for CircuitPython library [25]. CircuitPython is minimal Python version for microcontrollers [26]. The usability will be evaluated before the next semester and with high probability put to use. This, if successful, will complete the transition from Python 2.7.

## **5.3. OpenCV Adaptations**

Open Source Computer Vision (OpenCV) is the go-to image processing library developed on C/C++. The library has interfaces to Python, Java, MATLAB supporting Linux, MacOS, Windows, iOS, and Android platforms [27]. It is also the main image processing package used by the Robot Operating System (ROS) that students will use further in their studies in courses such as LOTI.05.023 Practical Work in Robotics or LOTI.05.057 Robotics Technology. It is the most sophisticated library that students work with in the robotics course. It has taken a considerable amount of effort for the author to organise installation and base usage principles, and the staff takes extra time to help students get acquainted with its base functionality. The following sections focus on the installation and customisation efforts by the author.

### **5.3.1. Plan (2017/2018A)**

OpenCV had already been introduced into the course at its inception and although the way it was used had fluctuated a bit it was still going to be the tool used for image processing. To simplify the software management process and installation on personal laptops it was decided that we would use the OpenCV 2.4 branch that is still supported and is the version available in Ubuntu 16.04, and Raspbian Stretch repositories. There were slight differences between the Windows and Ubuntu version used (2.4.13 and 2.4.9 respectively) but most of the functionality need worked near identically. Laptops in the classroom were preloaded with corresponding versions for both Ubuntu and Windows. Each student would install OpenCV to their RPi version 2B or 3B.

### 5.3.2. Act and Observe (2017/2018S)

Some students opted to install a version from the 3.4 branch to their laptops which yielded different results than the version used on their robot which demanded some additional debug time. Students were given links to the OpenCV documentation and the freedom to use the functions they preferred to use for solving the object detection tasks in relevant labs. Students had some problems with the RPi overheating when running OpenCV.

### 5.3.3. Reflect and Plan (Summer 2018)

OpenCV 2.4 branch would continue to get supported at least by the end of the Ubuntu 16.04 LTS maintenance updates period—April 2021 [22]. OpenCV 2.4.9 version supported was released in April 2014 [28] which in terms of image processing was a long time ago, even considering that the students would not use the most advanced features of the library. Real world application of some detector functions gave better performance for fellow students and instructors alike. A direct benefit of deciding to use only one operating system on the laptops, discussed in [Section 5.2](#) was that we could focus on having better support for the two remaining operating systems. It was proposed at a meeting by the author that the course would move to OpenCV version 3. The same version would be used, both on the laptop and the RPi to ensure uniformity between the two environments. This would take one variable out of the development and testing cycle applied by the learners. This led the author to two separate tasks.

The author had several newer versions and a couple of strategies to choose from to make this work. These options are organised into [Table 7](#). The option to upgrade the whole operating system as the laptops were also used by other courses needing Ubuntu 16.04 and more importantly Raspbian Stretch was already the most up-to-date release.

Table 7. Upgrade Paths Considered for the OpenCV Library

Option	+ (Positive)	– (Negative)	Conclusion
Third party repository that would contain compatible packages with a newer OpenCV version.	Someone has already taken the time to create the packages. Students would be able to install the packages in the labs and at home alike with minimal effort.	The package maintainers quality of work is going to determine much of the problems that one might encounter.	The official Debian packages list and the far wider Packages Search site were consulted. There were no good options in August 2018.
Incorporate compiling OpenCV from source into the lab manuals.	Being able to choose what libraries get included and the code is optimised for our specific chip.	The downside is that compiling OpenCV with all of its dependencies will take over 3 hours on a RPi. Taken that the user knows exactly what to do.	This could have been very educational but taking into account that many of the students had not had a course covering or explicitly using compilers or toolchains, the author decided that this would not be a very engaging lab.
Backport newer OpenCV package from a repository for a newer release of the operating system.	The package compatibility with the operating system quirks is likely to be less of a problem. Fairly easy to distribute.	Being limited by the number of OpenCV versions to choose from.	This might be a viable option if the need arises—which it did.
Compile and package OpenCV from scratch.	A lot of freedom on choosing the exact configuration and release to base the packages on.	All of the responsibility in getting the configuration and packaging exactly right.	This might be a viable solution in the long run but it seemed disproportionate to invest this amount of time to solving this one task at that time.

Debian packages [29] and Packages Search [30] are very common sites for packages search.

### 5.3.4. Act and Observe (2018/2019A)

A solution for RPi was needed. After going through the options for the upgrade [Table 7](#) the author decided to pursue the backport option. The author checked if the official backports repository contained the required packages—it didn't. This semester the course mainly concentrated on RPi model 3B+ with the occasional 3B serving as backup units. The model 3B+ uses a chip made by Broadcom (BCM2837B0) that runs at a maximum of 1.4GHz and a metal heat spreader [31]. Debian Buster was still in development with no sight of a release happening any time soon. But they were already building packages for the ARM Hard Float (armhf) architecture. Which is the version to use for RPi 3B+ as it uses the built in floating-point unit (FPU) of the Broadcom chip. Buster development included OpenCV version 3.2.0. The experimental branch of Debian had newer OpenCV packages available but this a bleeding edge branch and not suitable for production use. The author decided to backport these Buster packages to Stretch and use the version 3.2.0 for the 2018/2019A semester. In the backporting process the author recompiled and packaged OpenCV on a RPi 3B+ and distributed the packages to be used in the labs.

The second platform to support where the classroom laptops. This was needed to enable testing on laptops and completion of some project work and Lab10. The OpenCV version used on the RPi now dictated the version we needed on the laptops. The obvious solution would have been to backport the packages from next Long Term Support (LTS) Ubuntu release "18.04" which had the same version (3.2.0) as Debian Buster. This was unsuccessful due to an unresolved bug in the official Makefile for the OpenCV package. Instead of finding the fix for the bug the author was forced due to time constraints to leverage the configuration management already set-up via Ansible as described in [Section 6.4](#). Although this was initially discarded from the viable options list it had the benefit of straightforward process and it would lay some base for creating fully customised packages for the course. The compilation process was also considerably faster on the laptops when compared to the RPi platform. It did have the downside of complicating the removal process.

### 5.3.5. Reflect and Plan (2018/2019S)

Improved thermal management of the RPi 3B+ allowed the students to use more computational power with some mitigation of overheating the whole system when running image processing or other high load tasks. The use of OpenCV 3.2 enabled students to benefit from new functionality for their projects. The manually compiled and installed OpenCV on the laptops did prove to be a bit of work to get rid of after the semester ended to make room for the next course using ROS and avoid OpenCV version conflicts between ROS and the host



system. OpenCV 4.0 was released in November 2018 and has already been followed by a 4.1 release. The author is in the process of integrating OpenCV 4.0 into the course but the majority of this will be carried out in the summer.

## 5.4. Version Control

Distributed version-control system Git is one of the core technologies supporting the course. Using such a versioning tool has many benefits some of which are listed in the first subsection. Each student worked on two separate repositories in the course, one for the labs and one for the course project. Both of which have dedicated sections. Bitbucket was chosen to host all of the courses repositories as it has a suitable Academic plan for the courses purposes. The reasoning behind not using universities internal Git hosting service is to give students experience with one very common hosting platform.

### 5.4.1. System benefits

The following list shows some of the benefits from students perspective.

1. Backup in case of failure
  - a. Device failure on physical or filesystem level
  - b. Loss of the device
  - c. Accidental deletion of file
  - d. Accidental format by fellow student
2. Reverting changes after fixing code has not worked out as planned (fallback)
3. Keeping track of their own progress
4. Finding out who (and in some cases why) introduced a bug
5. Finding out what has changed in a specified amount of time
6. Improved collaboration

In addition to these benefits there are others that are clearer when viewed from the teachers perspective. There are some mutually beneficial areas with the students, but alternative reasoning applies. These are in the following list.

1. Students learn good software development practises earlier—for the majority of students this is the first course to use code versioning systems. Having experience with

VCS is a highly requested skill by the employees.

2. Students are very unlikely to accidentally lose large portions of their work
3. Code base to use for the exams
  - a. Identical revisions and history of changes — simplifies the grading process
4. Simplifies plagiarism detection — more versions of code exist for inspection

### 5.4.2. Students' Personal Repository

A personal Git repository is created to each student at the beginning of the course. There is a template repository and then each student's repository is a fork for this repository. This fits into the workflow of publishing new base code to students on a weekly basis. See the workflow listed below. Bitbucket API is used to automate large portions of this process. Atlassian is going through a transition phase from APIv1 [32] to APIv2 [33] with Bitbucket at the moment. APIv1 is already deprecated and APIv2 does not yet have the support for all of the fields, so some manual configuration steps are required.

#### *Repository update workflow*

1. A lab manual is reviewed and tested by the instructor in charge.
2. At least one other instructor or the lecturer reviews the changes.
3. Fixes are implemented in the manual and base code for students.
4. Base code is pushed to the master branch in student template repository.
5. Each student's repository gets changes from upstream merged to the master branch of their repository by a semi-automated process using Kantu for Chrome Automation Plugin [34].
6. Student follows the process of pulling and merging new code into their development branch.
7. Student solves tasks in the lab.
8. Student creates a pull request (PR) after each laboratory practical.

#### 5.4.2.1. Plan (2017/2018A)

It was decided at one of the very first meetings that this course would make use of Git VCS. The author had positive previous experience with introducing a VCS in courses and elsewhere. Caution was expressed by some members of the staff on Git being too

complicated of a tool to teach as an extra in a course without dedicating considerable amount of time in the process, this was noted but did not reverse the introduction of Git. It was decided by the author that each student would have their personal repository instead of having one central repository for all students. This helps to somewhat isolate the amount of damage the students can do to each-others repositories and helps to some extent avoid unintentional sharing of solutions with other students. There was also a need for releasing new base code with almost every lab. It was decided that this would also be done via the students' repositories.

#### 5.4.2.2. Act and Observe (2017/2018S)

Version-control systems were introduced in the second lecture and after that in small portions over several consecutive lectures to help divide the amount of new concepts the students had to learn each week. With a couple of exceptions the students did not indicate any or much previous experience when prompted in the lecture. This was confirmed at the weekly meetings by the instructors. The Atlassian Bitbucket platform was chosen to be used for hosting the repositories for two main reasons. Firstly the author, who was responsible for the setup had previous experience with this Software-as-a-Service (SaaS) stack. Secondly Atlassian offered free academic accounts for students who signed up with a '@ut.ee' address. The repositories for each student would be created by the author to make sure that they get named and set-up uniformly across all students. This would help to take some of the complications away from getting started with Git in this course. Students were asked to complete the registration process on Bitbucket and send their real and user name via e-mail to the instructor.

*A student template repository was created.*

1. Read and write access was granted to all of the instructors.
2. A fork of this repository was created for each student.
3. Master branch write access was taken away from everyone but the teaching staff.
  - a. This was done to make sure that the branch would hold the same structure as the template repository, helping to avoid conflicts when merging weekly updates.
4. Access was granted to the student and to their two corresponding instructors.

All of the steps up to the point where access was granted to the student and instructors was done manually. Options as listed in [Table 8](#) were explored to introduce automation to the process of managing 30+ repositories. In the end the third option was chosen as the API was in an unfavourable state of transition from version 1 [33] to version 2 [32] and there was

time pressure in kick-starting the semester. One step that was semi-automated by the author was the generation of a bookmark list for all of the students repositories. This list was organised into a folder on the Chromium web browser and this gave a open all repository dashboards at the correct location with a single click solution which took the weekly update time from around 20 minutes to 6 or 7 minutes.

*Table 8. Options Considered for Repository Management*

<b>Option</b>	<b>+ (Positive)</b>	<b>- (Negative)</b>	<b>Decision</b>
Use a script (shell, make, etc.) for managing repositories locally	Another copy of all of the repositories; can work with them locally	Limited by Git functionality, unable to manage Bitbucket cloud options; Creates more traffic on updates; Needs disk space for each repository;	Being limited by no control over cloud options means that this option could only be a part of the solution
Use Bitbucket API	Support for editing Bitbucket Cloud settings.	Atlassian was in the process of transitioning from APIv1 to APIv2. APIv1 was already deprecated and APIv2 was missing a lot of functionality.	If V1 would have been used then it would have needed to be replaced by a new implementation on the following semester. Quite a few manual configuration steps would have been required when going with V2
Manually go through all of the steps for each repository	Easy to implement	Does not scale well; Prone to errors;	This solution does not have considerable benefits if considered for more than one semester

#### **5.4.2.3. Reflect and Plan (Summer 2018)**

The compromise of not automating a lot in the repository management process seemed reasonable even after the semester ended. More time was spent, but it was spent evenly over the semester helping to reduce the workload in the beginning of the semester. That said there was a need for automation as the prognosis was that the 2018/2019A semester would triple the number of students in the course. Bitbucket API had evolved considerably over the

half a year and had some useful functionality from the course perspective. The author decided to start the process of moving over to using the API where possible. Another tool (a Chromium extension) Kantu Browser Automation [34] was also introduced to automate a couple of clicks not exposed by the API.

#### 5.4.2.4. Act and Observe (2018/2019A)

The API was used via curl command line utility. The following reduced example in [Code Example 2](#) creates a fork from the template repository for all of the students in the list.

##### Code Example 2

```
1 student_id=(b1 b2 b3) ①
2 arraylength_id=${#StudentId[@]}
3 for (( i=0; i<${arraylength_id}; i++ )); ②
4 do
5     echo
6     echo "Working with: " ${student_id[$i]}
7     curl -X POST --netrc-file ~/passwordfile
https://api.bitbucket.org/2.0/repositories/account_name/robotics-loti.05.010-18-19a-student-template/forks -H
'Content-Type: application/json' -d '{"name": "robotics-loti.05.010-18-19a-student-${student_id[$i]}"}' ③
8     echo
9     sleep 2 ④
10 done
```

- ① List of student ID's to create the fork for.
- ② For each student.
- ③ Actually construct the query and call the REST API. It is a POST query to the forks endpoint that has the request included as an HTTP header. The `--netrc-file` parameter is used to avoid exposing Bitbucket password. An even better solution would be to authenticate via public key infrastructure (PKI).
- ④ Be nice to other users and wait for 2 seconds before creating the next fork.

In addition to the `/forks` endpoint the `/branch-restrictions` endpoint was also used to set the access rights for the master branch. Groups for the instructors were created using the web interface as this was not possible using the APIv2 at that time.

Kantu Browser Automation tool accepts command description as JSON — this helps to avoid having to use the GUI to set the nearly 300 operations needed for this application. The JSON was generated using a small script and the list of student ID's. Every repository needed three operations every time a new manual was released and some needed two rounds as a couple of bugs were identified in the code. The following snippet of [Code Example 3](#) contains the

description for these three commands.

### Code Example 3

```
1 {
2   "CreationDate": "2018-9-12", ①
3   "Commands": [
4     {
5       "Command": "open", ②
6       "Target": "https://bitbucket.org/account_name/robotics-loti.05.010-18-19a-student-xxxxxx",
7       "Value": ""
8     },
9     {
10      "Command": "click", ③
11      "Target": "link=Sync now.",
12      "Value": ""
13    },
14    {
15      "Command": "click", ④
16      "Target": "//*[@id=\"bb-undefined-dialog\"]/div/div[2]/button",
17      "Value": ""
18    },
19  ]
20 }
```

- ① Date is used for revisions in the Kantu extension.
- ② First command to open the dashboard of a particular students repository.
- ③ Second command, parse the dashboard for a link named "Sync now." and click on it.
- ④ Third command, in the dialogue window that opens find the correct `<div>` with a button in it and click on it.

#### 5.4.2.5. Reflect and Plan (2018/2019S)

The automation done for the repositories paid for itself in less than one semester. When considering only the Kantu upstream merge over 3000 clicks were avoided. Further automation is planned to reach a milestone where after getting and cleansing account information from the students and assigning instructors to groups one could generate all of the student repositories with the corresponding access rights and settings. Some time is also planned to comply with API changes as the GDPR-related change announced in October 2018 and in effect by the end of April 2019

### **5.4.3. Course Projects' Repository**

It was agreed that each course project would have its own repository. The exact details were not agreed in the planning phase as the course project starts about mid semester.

#### **5.4.3.1. Plan (2017/2018A)**

Students would get their first introduction or reminder of Git from using their personal repositories. It was collectively decided that the course project would start after students have had their first exposure to image processing in general and OpenCV in particular. Image processing was considered to be one of the more evolved topics in the course and most of the projects would likely try to make use of these methods. As for the course projects it was decided that the students would get the opportunity to create and manage their own repository. One could argue that this is conceptually later than the natural learning process of first creating a repository and then working with it but better reflects the junior position many of the students would have after graduation or later in their studies. At the same time we also needed to maintain some sort of control over who can access the projects and make sure that the instructors would not have to look at students work on multiple platforms complicating the grading workflow and leaving less time for feedback.

The balance between students learning new concepts and the staff maintaining some sort of control over all of the repositories was achieved by using Bitbucket teams.

#### **5.4.3.2. Act and Observe (2017/2018S)**

A team was created for each project. Read/write access was granted to the students in that group and corresponding pair of instructors. This meant that students could freely add repositories under the team account.

#### **5.4.3.3. Reflect and Plan (Summer 2018)**

Students did use Git to manage some parts of the projects but with only two students in the group some were still tempted to use other more known communication platforms to share code. It was decided that the need to present their work via Git would be made compulsory for all stages of the project to encourage the students to learn more about code versioning. The increase in the number of students also meant more students per group increasing the benefits of using a VCS. Some students did not go through the suggested Git tutorial in the beginning of the semester as suggested in the lectures and labs. This in turn resulted in an extended learning period and missing out on the benefits of using VCS. One of the reasons

communicated by the instructors was that the activity wasn't graded. Working through a Git tutorial was added as an official step in the new Lab00 manual.

#### **5.4.3.4. Act and Observe (2018/2019A)**

Git was again explained in the lectures and the tutorial was completed by all students in their first weeks. A Google Forms questionnaire was created by the instructors responsible for the Lab00 to help with collecting user accounts that needed access to the repositories. Groups of mainly four students worked on the course projects.

#### **5.4.3.5. Reflect and Plan (2018/2019S)**

The learning curve of using Git for the purposes of the course seemed to be better judging by the feedback than on the previous semester. One of the supporting factors for this could have been the fact that now some of the instructors had more experience with using Git in the educational process and the instructions available were better worded but it is also plausible that going through the tutorial on the first weeks helped in this process. The course projects showed an increase in unsuccessful merge conflict resolutions with some of the conflict information getting pushed to the repository on Bitbucket servers.



# 6. Internal-Facing Solutions

In addition to the software infrastructure elements that were mostly student-facing there were also several other solutions developed for the staff and supporting members from the University. VCS plays a central role in making sure that students would not lose their work and in instructors being able to monitor their progress. It was also used for tracking large portions of the work done by the staff. In addition Google Docs is used to support use cases where rapid and lightweight versioning is key. Managing all of this software also introduced inherent need for standardisation to make sure that all of the environments were homogeneous and easily reproducible. In addition to all of the software the hardware also needed to be kept track of—solutions for this are covered in the final section of this paragraph.

## 6.1. Instructors' Repository

### 6.1.1. Plan (2017/2018A)

Following the path of the students, the instructors also started using Git. The author set-up a central repository for much of the course related information that the instructors needed. Each instructor had at the very least their personal branch in the repository. Use of more branches was encouraged but not compulsory. This repository was also hosted on Bitbucket.

### 6.1.2. Act and Observe (2017/2018S)

As with the students the instructors also had occasional struggle with using Git but no persisting issues emerged. Most common issues were related to confusion with the active branch and with resolving merge conflicts. The repository contained a `README.md` that outlined the lab review deadlines for the instructors and information on the structure of repository and how to compile the lab manuals with prerequisites to do so. It later also included the assignment table which listed SD-cards assigned to students.

### 6.1.3. Reflect and Plan (Summer 2018)

Access rights needed a review and updates as some of the instructors were unavailable for the Autumn semester and 9 new instructors were introduced to the course. A small introduction to the repository and how to commit new content was conducted by the author. The course repository did not contain any grades for the previous semester which was

beneficial as three of the instructors were students on the previous semester. Care was also taken in other areas not to expose results from the previous year.

#### **6.1.4. Act and Observe (2018/2019A)**

Introducing many new instructors to the course added new ideas and helped to pinpoint several grammar and wording inconsistencies in the course materials. It also helped to locate areas where there was an agreement between the older instructors but no information about the decision in writing. These included coding conventions, documentation explaining the manual release procedure, and the overall style guide used for writing the manuals.

#### **6.1.5. Reflect and Plan (2018/2019S)**

There is a general direction of migrating more of the information relevant for the instructors over to Git—such as various grading rubrics. One of the more influential changes in works for the instructors repository is the instructor’s manual. As the course has grown to having 12 instructors it is increasingly hard to communicate all of the information to all of the instructors. In addition it was communicated by the new instructors that it was hard to catch up with all of the details and pedagogical reasoning behind the tasks in the manuals.

All lab manuals, example solutions, grading rubrics are kept in that repository. By convention changes should be first added to a personal branch and then a PR is created to be approved by the author or a fellow instructor.

## **6.2. Lab Manual Template**

The course makes use of several templates to simplify the development and release process while helping to ensure a more uniform experience for everyone involved. Some templates are covered in other sections such as the student repository template under [Section 5.4.2](#). The author has established and developed a template for lab manuals that is maintained in a Git repository. There have been many suggestions for improvement by the instructors. The main idea of this template is to ensure consistency between lab manuals, introduce possibly useful AsciiDoc functionality for new instructors, and simplify the process of updating repetitive information (e.g. how to get updated code before starting with the lab). Template gets most of its major updates between the end of previous course and start of new one. This is to ensure that the instructor responsible for reviewing the lab manual can make appropriate structure updates to the lab manual. [Figure 18](#) shows the complete process

description.

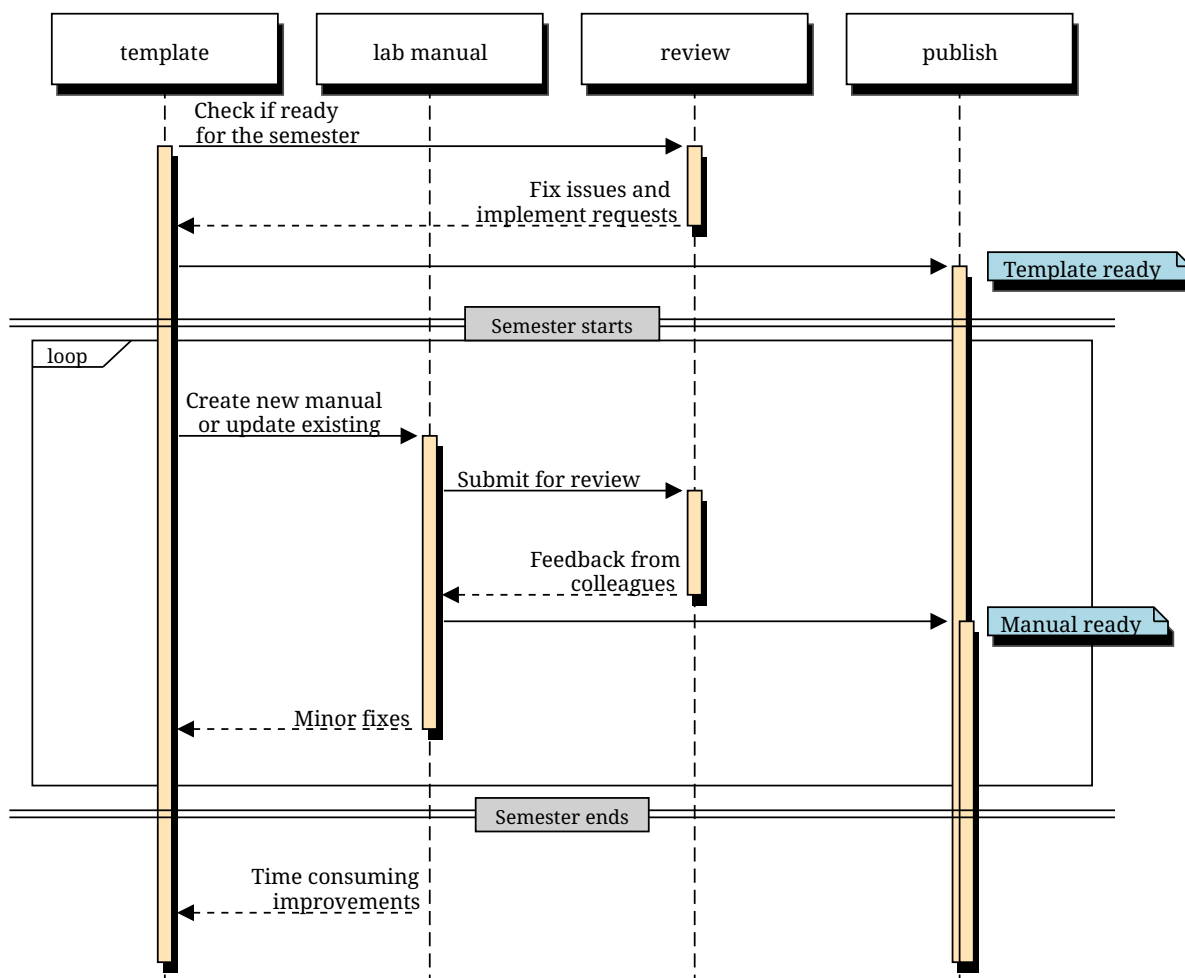


Figure 18. Lab manual review process

### 6.3. Google Docs






The course staff used a collection of documents based on the Google Docs service. The most heavily used document is the one for Student progress. This is the entry level solution if the document doesn't fit well under Git or needs some rapid collaborative development iterations before becoming stable and being transferred to Git. That said for some of them this is the current best option. These include the Google Forms based questionnaires. [Table 9](#) is used to give a more structured overview of all of the files and their use cases. Some files are exclusively for staff use and some (denoted Both) are fillable, visible or even editable for the students.  is used to denote a Google spreadsheet was used to contain the information and  to denote the use of Google Forms. Some of the files have overlapping topics or duplicate information (automatically imported). This is to make the usage more convenient and at times allows to achieve results that are otherwise not possible with the official access model of Google Docs. Like the "Open seats in labs" document.

Table 9. Files Managed in Google Docs

Document	Usage and Reasoning
<p><b>Student progress</b>              2017/2018S - ..            Staff only</p>	<p>This is the main file containing information about students progress in the course. This takes more than 20 sheets all in all. Each lab group has a sheet with all of their students and their progress in individual task level. Project progress and hardware lists are also maintained here for all of the groups including all of the points gained and lost. Bug bounty points for students, extensions, teaching assistants and a lot of exam related information. It also contains a sheet where instructors assign students to "Open seats in labs" spreadsheet (interfaced documents).</p>
<p><b>Topic registration and Availability for visits</b>              2017/2018S - ..            Both</p>	<p>Document where students can indicate their general availability for visits by two hour slots. This helps in planning company visits and other activities. Second sheet in the document is for students to register their "Week in the News" presentation topics.</p>
<p><b>Open seats in labs</b>              2018/2019A - ..            Both</p>	<p>A spreadsheet for students to see open seats in labs. Registration works via a request to someone in the staff— either directly or via e-mail. This process serves two purposes. Firstly it creates a small registration barrier helping to increase the likelihood that the student will actually show up otherwise students just sign-up and forget. Secondly it helps to mitigate the risk of resource starvation commonly known from concurrent programming. In our case this translates to a student being unable to register for extra lab time due to mismatch in slot release times and student checking the spreadsheet for an available seat.</p>
<p><b>Grades for the Poster Session</b>              2017/2018S - ..            Staff only</p>	<p>In addition to the grades this file contains individual points given by instructors with additional comments. The reason for this file to be separate is that the final presentations are graded by many instructors and it is therefore different from the overall lab progress.</p>

Document	Usage and Reasoning
<b>Instructor Preferences</b> <input type="checkbox"/> 2018/2019A - .. Staff only	This file is used by the author to divide instructors to lab groups and to assign lab manuals for review. It serves a very important role in making sure that on one hand instructors get the lab groups and manuals that suit them as well as possible. On the other hand every lab group gets at least one instructor that has previous professional experience.
<b>Bitbucket accounts</b> <input checked="" type="checkbox"/> 2018/2019A - .. Staff only	It's a form used for collecting information on Bitbucket user accounts belonging to the students to enable semi-automated repository creation and permission granting. The students are introduced to this form in Lab00. The results are only visible to the instructors.
Project preferences <input checked="" type="checkbox"/> 2018/2019A - .. Both	A Form for student pairs to express their preferences and ideas on the course project. Later this information is used to combine the pairs into project groups. On the previous semester e-mails detailing this information were sent to the lecturer but this did not scale reasonably.

## 6.4. Configuration Management

### 6.4.1. Plan (2017/2018A)

The classroom consisted of laptops that had a dual-boot set-up of Ubuntu 16.04 and Windows 10. The list of software to be installed was going to be discussed on the weekly meetings and the installation would happen after the meeting. It was checked that correct versions of Python and Git were installed before the semester started. Students could opt to use their own laptop but they would in most cases need to manage the software installation. The Raspbian Stretch that was going to be installed on the students SD-cards and would be managed by the students and guided by the lab manuals and instructors where necessary.

### **6.4.2. Act and Observe (2017/2018S)**

Many of the students opted to use Windows or their own laptops. This meant that they had to switch between operating systems and manage installations on two different platforms. On a couple of occasions it happened that one or two of the laptops had not been configured for the corresponding week or had been configured differently. Instructors tried to support students questions about installing software on their own laptops when there were no other questions.

### **6.4.3. Reflect and Plan (Summer 2018)**

Configuring 14 laptops for the labs needed considerable amount of time and was somewhat error prone. If something was planned incorrectly then it had to be done again in all 14 laptops. Documentation on what was the exact configuration stayed with the instructors involved at that time. This suggested that a more systematised approach on configuration management might be beneficial. The laptops used for this lab get a clean installation by the ITO every summer. The amount of software needed on the laptops is considerably large as there are several courses using these laptops. The fresh installation is based on a cloning approach where one laptop per hardware configuration is set-up manually considering all requests for software an configuration. After which the whole image is cloned to all of the other computers. The author decided to set-up some of the software stack right away under ITO supervision which included OpenSSH Server and an authorised key for later remote access. This meant that no other manual steps should be needed in the service life of this operating system.

### **6.4.4. Act and Observe (2018/2019A)**

The author created and developed an Ansible Playbook to ensure a homogeneous configuration in the classroom. This was used to install new software and to configure software already installed on the system. In the first couple of weeks of the semester it also became apparent that the computer would get littered by students repositories and other files as personal user accounts were not available for this semester. A group of clean-up tasks was added to the Playbook and executed on a weekly basis.

### **6.4.5. Reflect and Plan (2018/2019S)**

The ease of installing new software and making tweaks on request was well received by the instructors based on feedback from the meetings. Clean-up script helped to reduce the

problems from missing central accounts. This also encouraged the students to actually push all of their changes after they were done for the day as their local copy of the repository was likely to disappear before their next lab. That said there were still a couple of occasions where students got lost in some other students repository. There is a plan to configure the laptops to use central accounts for the students for the next semester.

## **6.5. Resource Management System**

### **6.5.1. Plan (2017/2018A)**

It was decided that as the course would use considerable amount of hardware the staff would need to organise it in a way which would make it easy to distribute and collect the items in each lab. Transparent plastic storage boxes would be used to organise most of the hardware. List of items included would be taped to the lid of the boxes for reference.

### **6.5.2. Act and Observe (2017/2018S)**

The boxes were a good idea in general as one of the lab groups had to use a different classroom further away from the storage and the instructors needed to transport all of the equipment each week. It also helped the students to keep a tidier workspace as they had a box to rest unused items in. The boxes were used for the full course of the semester. A need for a lending system became clear in the second half of the semester as students started work on their projects and some needed to catch-up with the others. This shortcoming raised at a weekly meeting initiated a over-the-weekend implementation of a very simple spreadsheet to log borrow and return events.

### **6.5.3. Reflect and Plan (Summer 2018)**

The lists of items in the box became somewhat outdated as the course progressed and new labs became available. It was decided during a planning meeting that these lists would not be used in the next semester. The box-system proved to be very beneficial and would continue to be used in the following semester. The spreadsheet proved to be enforced too lightly as when reviewing the list the author identified multiple items that had consecutive borrow events but no return events although a quick check showed that the item was present. This issue when raised at the end of the semester meeting was identified to relate to some instructors ignoring the use of the spreadsheet after it became too long for them to go through and identify the corresponding borrow event. This combined with the approach of many more students next semester ended in a decision that the staff would try to find and

deploy an existing borrowing/asset management system that would be more convenient to use. The selection of the software was pushed to the beginning of the semester as it would not be needed right away.

#### **6.5.4. Act and Observe (2018/2019A)**

The short-listing of possible resource management systems for the course was conducted by instructor Kristo Allaje. Leihis [35] and Snipe-IT [36] were proposed. The author considered both systems for the task. They both have links to live demonstrations on their front page—enabling easy testing. Both are far more sophisticated when compared to the spreadsheet approach taken on the previous semester. From sustainability perspective both were scheduled to have a major release in the next 6 months which coincidentally was to version #5.0. Both systems were identified to be open source and free as in "free beer" with Leihis also being free as in "freedom of speech". The staff did not have any experience using either of these platforms. Most of the practical considerations did not give strong distinction between the platforms. This left the author with empirical methods, such as knowledge acquired from previous work to choose one of the platforms that seemed better documented and which gave slightly better initial user experience (UX).

An overall need for a course server that could host some services was identified. [Section 4.3.4](#) explains another use case for a course server. It could be either a virtual machine on some internal or external hosting service or our own server. Options were discussed within the Institute of Technology and a collective decision was made that the fastest way to get a server is to use one of the machines that had reached its' end of useful life for the previous application. Two servers originally configured by Ordi was set-up by the author. The author decided to combine RAM modules from two machines for improved performance. The server was also cleaned from dust and given a fresh BIOS battery, the later was needed before the computer would start. The final hardware configuration is presented in [Table 10](#) and compiled using physical inspection, OS level tools, and manufacturer or distributor websites [37, 38, 39, 40]. The hardware was evaluated to strongly exceed the minimum needs of the selected software stack [Figure 19](#) on the expected user load (5 concurrent users at most). The evaluation was based on authors previous work experience.



Table 10. Server hardware configuration

Item	Model	Parameters (single item)
CPU	<ol style="list-style-type: none"> <li>Dual Core AMD Opteron(tm) Processor 265</li> <li>Dual Core AMD Opteron(tm) Processor 265</li> </ol>	<ul style="list-style-type: none"> <li>Data width: 64 bit</li> <li>L1 cache: 2 x 128 KB</li> <li>L2 cache: 2 x 1 MB</li> <li>Number of Cores: 2</li> <li>Thermal Design Power: 95 Watt</li> <li>Frequency: 1800 MHz</li> </ul>
RAM	<ol style="list-style-type: none"> <li>Apacer</li> <li>Apacer</li> <li>Apacer</li> <li>Apacer</li> </ol>	<ul style="list-style-type: none"> <li>PC 3200</li> <li>4 GB DRAM</li> <li>ECC</li> </ul>
Storage	<ol style="list-style-type: none"> <li>WDC WD2500JS-22NCB1</li> <li>WDC WD2500JS-22NCB1</li> </ol>	<ul style="list-style-type: none"> <li>Interface: Serial ATA-300</li> <li>Capacity: 250 GB</li> <li>Cache: 8 MB</li> <li>Software RAID: level 1</li> </ul>
Network	<ol style="list-style-type: none"> <li>Broadcom NetXtreme BCM5704 Gigabit Ethernet</li> <li>Intel EtherExpress 82557 PRO/100 S Server Adapter</li> </ol>	<ul style="list-style-type: none"> <li>Broadcom 1 Gbit/s socket</li> <li>Broadcom 1 Gbit/s socket</li> <li>Intel 100 Mbit/s socket</li> </ul>
Motherboard	<ol style="list-style-type: none"> <li>Tyan Thunder K8S/K8SD Pro</li> </ol>	<ul style="list-style-type: none"> <li>Dual CPU socket</li> <li>North bridge: AMD-8111</li> <li>South bridge: AMD-8131</li> </ul>

CentOS Linux [41] was selected as it has a 10-year security patches support for major versions and a long running support for older hardware [42]. Version 7.6 was the latest available at the time of installation. The BIOS in the server was unable to boot from a USB-device. Another reason supporting the decision was that the author already managed other servers based on the same OS. The author carried out the following sequence of operations after all of the previous steps:

1. created Live DVD for CentOS;
2. installed the OS;
3. moved the server to a rack;
4. connected the server to the network;
5. applied for a static IP from ITO
6. connected the server to an uninterruptible power supply (UPS) with sufficient capacity.

Configuration of the server was done and is managed using Ansible. The structure of the configuration file is presented on [Figure 19](#).

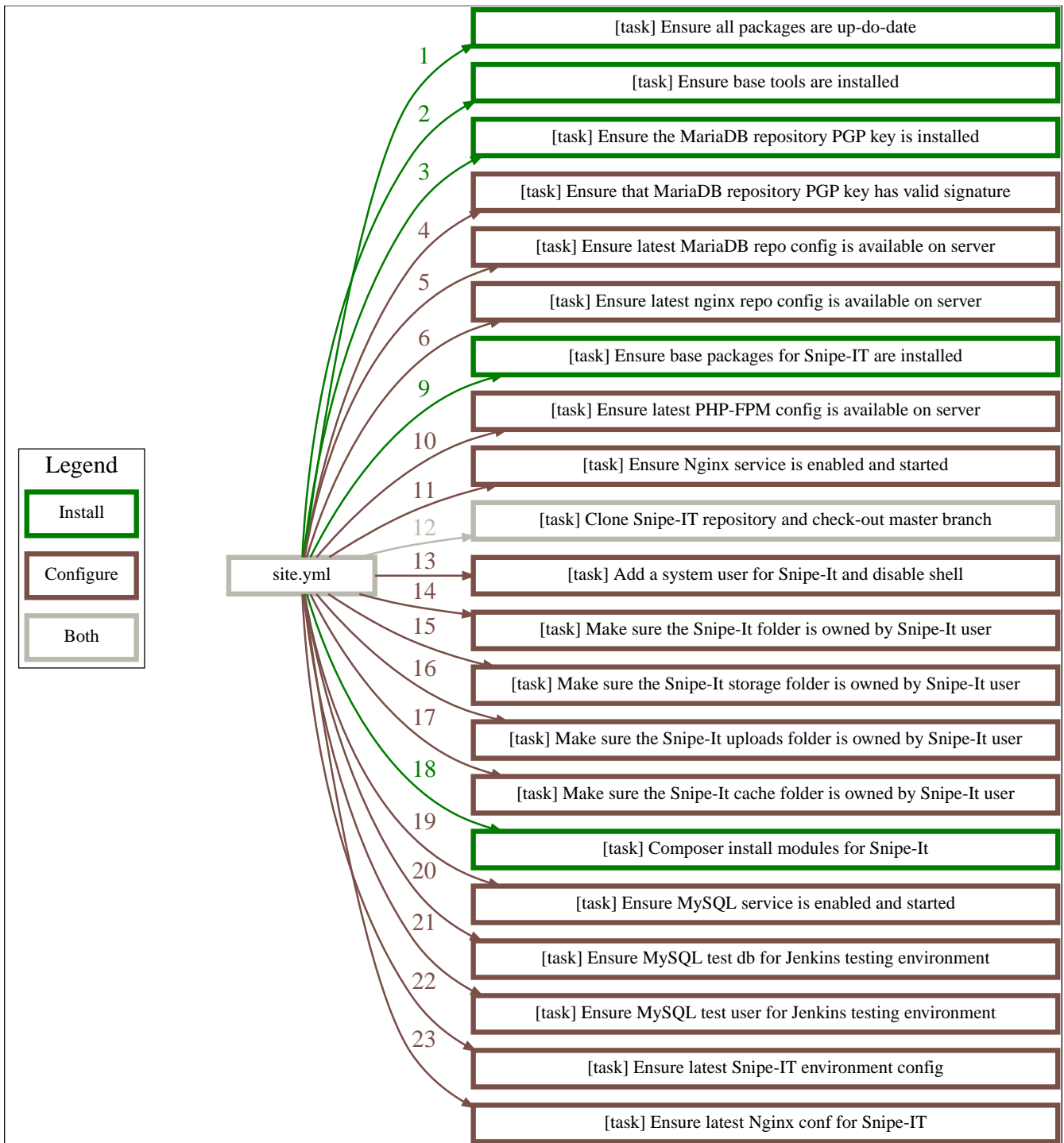


Figure 19. Server Ansible configuration structure

Snipe-It together with its prerequisites were installed and configured on the course server. On the Snipe-It application side the author:

1. created accounts and granted permissions to all of the staff members,
2. created accounts and withdrew access rights to all of the students to enable automatic notifications via e-mail,
3. created the initial structure and added example items for course hardware.

After all of the steps were completed the system was introduced at a staff meeting and put to use.

### **6.5.5. Reflect and Plan (2018/2019S)**

Snipe-It was considerably easier to use than the spreadsheet but a bottleneck was identified by the instructors—the system was missing the ability to check-out multiple items at the same time to the same student. The corresponding check-in feature was very convenient to use. Solving this problem is still a work in progress and is on the roadmap to be resolved before the next semester starts. It is also likely that v5.0 will be released before the next semester as there is a beta-2 release available since March 2019 [36].

## 7. Summary

In the thesis it was described in the form of educational action research cycles how an introductory robotics course was designed, developed, and conducted. Two full cycles, including two runs of the course were considered. The description focused on the author's work as the lecturer in charge leading 14 instructors over the two semesters. The work included introducing new tools, technology, and teaching methodology to the course and following through in their implementation and continued use in the educational process. The use of VCS and weekly staff meetings amongst others supported the rapid development process needed for the transition period from previous team. The nature of the field and the strain put on the robots and their components will continue to push the course towards new models and versions. Both semesters saw additional routine tasks automated, which helped scale the course from 30 to 80 students in one semester. The work done by others was acknowledged and appreciated.

In the two semesters more than a hundred students had the opportunity to advance their knowledge in the fields of robotics and programming. Two students who had successfully completed the course took advantage of the opportunity to join the staff for the 2017/2018S semester, bringing useful insight from the previous course organisation. Four students who successfully completed the 2017/2018S semester were granted the opportunity to gain additional knowledge whilst instructing the course in the 2018/2019A semester. First students from the fresh Science and Technology curriculum were introduced to robotics, their progress gave feedback to the programme manager on how to improve the curriculum. The change introduced to the Computer Engineering curriculum - moving the course from the fourth to the first semester - was successfully accommodated.

Many of the sections in chapters contained plans and possible improvements for the 2019/2020A semester. These included the plan to stop using Python 2.7, start using 4.0+ version of OpenCV library, further introduction of the newer GoPiGo3 robotic platform, and further implementation of the resource management system besides many other improvements. The course has been opened for registration for the 2019/2020A semester and the next staff meeting will take place in June 2019.

# Acknowledgement

The journey of completing my master's has been very educational. I would firstly like to thank Sven for being my initial supervisor who helped me in choosing some additional courses for my studies. He also taught me a thing or two about time management, which has proven invaluable both in and outside of academia. Secondly, I would like to thank my wife for constant support and encouragement, for teaching me the value of doing things the right way, and for making me much of what I am today. Finally, I would like to thank my friends and family for offering various invigorating perspectives and interpretations on education and teaching.

In terms of the thesis I would like to thank all of my colleagues for getting on board with somewhat time consuming and controversial ideas. I would also like to thank Pille, Uku, Sandra, Madis, Eva and my dad for the constructive criticism provided on my thesis—it was very helpful. And last but certainly not least, I would like to thank Eno Tõnisson for getting on board in the very end, helping to ensure that the work meets the academic standard.

# References

1. **O. Gomes, S. Pereira.** On the economic consequences of automation and robotics. *Journal of Economic and Administrative Sciences*. Epub ahead of print 2018. DOI: 10.1108/JEAS-04-2018-0049.
2. **M. DeHaan, Ansible Community.** Ansible configuration management system, <https://www.ansible.com> (accessed May 5, 2019).
3. **M. DeHaan, Ansible Community.** Ansible documentation - Playbooks, [https://docs.ansible.com/ansible/latest/user\\_guide/playbooks.html](https://docs.ansible.com/ansible/latest/user_guide/playbooks.html) (accessed May 5, 2019).
4. **S. Rackham, et al.** AsciiDoc markup language for document description, <http://asciidoc.org> (accessed May 5, 2019).
5. **D. Allen, S. White.** AsciiDoctor toolchain for AsciiDoc processing, <https://asciidoctor.org> (accessed May 5, 2019).
6. **TUIT.** LOTI.05.010 Robotics course description, <https://ois2.ut.ee/#/courses/LOTI.05.010/details> (accessed May 7, 2019).
7. **UT.** Study Regulations of the University of Tartu, <https://www.ut.ee/studreg> (accessed May 7, 2019).
8. **TUIT.** Computer Engineering Curriculum (83866), <https://ois2.ut.ee/#/curricula/83866/details> (accessed March 6, 2019).
9. **TUIT.** Science and Technology Curriculum (144918), <https://ois2.ut.ee/#/curricula/144918/details> (accessed May 6, 2019).
10. **J. Prior.** Integrating extra credit exercises into a university English-language course: how action research provided a framework to identify a practical problem. *Educational Action Research* 2018; 26: 770–786.
11. **S. Kemmis, R. McTaggart.** *The Action Research Planner*. Deakin University, <https://books.google.ee/books?id=EkhLQAAACAAJ> (1988).
12. **E. Lõfström.** *Tegevusuuringu käsiraamat [Estonian]*. Eduko, <https://www.digar.ee/arhiiv/et/download/107855> (2011).
13. **O. Jones, A. Gorra.** Assessment feedback only on demand: Supporting the few not supplying the many. *Active Learning in Higher Education* 2013; 14: 149–161.
14. **Canonical Ltd.** Ubuntu Linux operating system, <https://www.ubuntu.com> (accessed May 5, 2019).

15. **L. Torvalds, et al.** Git, distributed version control system, <https://git-scm.com> (accessed May 5, 2019).
16. **A. Chapyzhenka, J. Probell.** WaveDrom: Rendering Beautiful Waveforms from Plain Text, [https://wavedrom.com/images/SNUG2016\\_WaveDrom.pdf](https://wavedrom.com/images/SNUG2016_WaveDrom.pdf) (accessed May 7, 2019).
17. **ELEC Freaks.** Ultrasonic Ranging Module HC-SR04, <https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf> (accessed April 30, 2019).
18. **Public Domain.** Moodle learning management system, <https://moodle.org> (accessed May 9, 2019).
19. **Dexter Industries.** Raspbian for Robots operating system customisation, [https://github.com/DexterInd/Raspbian\\_For\\_Robots](https://github.com/DexterInd/Raspbian_For_Robots) (accessed May 9, 2019).
20. M. Thompson, P. Green, Raspberry Pi Foundation. Raspbian operating system, <https://www.raspberrypi.org/downloads/raspbian/> (accessed May 9, 2019).
21. **I. Murdock, Software in the Public Interest.** Debian Linux, <https://www.debian.org> (accessed May 9, 2019).
22. **Canonical Ltd.** Ubuntu Linux release cycles, <https://www.ubuntu.com/about/release-cycle> (accessed May 5, 2019).
23. **I. Murdock, Software in the Public Interest.** Debian Linux releases, <https://wiki.debian.org/DebianReleases> (accessed May 9, 2019).
24. **Public Domain.** PEP373 - Python 2.7 Release Schedule, <https://www.python.org/dev/peps/pep-0373/> (accessed May 9, 2019).
25. **T. DiCola, Adafruit Industries.** Adafruit Python CharLCD, [https://github.com/adafruit/Adafruit\\_Python\\_CharLCD](https://github.com/adafruit/Adafruit_Python_CharLCD) (accessed May 9, 2019).
26. **Public Domain.** CircuitPython, <https://github.com/adafruit/circuitpython> (accessed May 9, 2019).
27. **OpenCV team.** OpenCV (Open Source Computer Vision Library), <https://opencv.org/about/> (accessed May 5, 2019).
28. **OpenCV team.** OpenCV Releases, <https://opencv.org/releases/> (accessed May 5, 2019).
29. **Software in the Public Interest Inc.** Debian packages list, <https://www.debian.org/distrib/packages> (accessed March 5, 2019).
30. **M. Ulianytskyi.** Linux packages list, <https://pkgs.org> (accessed March 5, 2019).
31. **Raspberry Pi Foundation.** Raspberry Pi hardware configuration documentation,



<https://www.raspberrypi.org/documentation/hardware/raspberrypi/README.md>

(accessed March 5, 2019).

32. **Atlassian Corporation Plc.** Bitbucket APIv1, <https://confluence.atlassian.com/bitbucket/version-1-423626337.html> (accessed March 10, 2019).
33. **Atlassian Corporation Plc.** Bitbucket APIv2, <https://developer.atlassian.com/bitbucket/api/2/reference/> (accessed March 10, 2019).
34. **a9t9 software GmbH.** Kantu for Chrome, <https://a9t9.com/kantu> (accessed March 6, 2019).
35. **Zürich University of the Arts (ZHdK).** Leihs, <https://github.com/leihs/leihs/wiki> (accessed March 25, 2019).
36. **Grokability, Inc.** Snipe-IT Asset Management system, <https://snipeitapp.com> (accessed March 25, 2019).
37. **AMD.** Opteron™ Processor Power and Thermal Data Sheet, <https://www.amd.com/system/files/TechDocs/30417.pdf> (accessed April 30, 2019).
38. **C|net.** Western Digital Blue WD2500JS - Hard Drive, <https://www.cnet.com/products/wd-blue-wd2500js-hard-drive-250-gb-sata-300-series/> (accessed April 30, 2019).
39. **Intel.** 8255x 10/100 Mbps Ethernet Controller Family - Open Source Software Developer Manual, <https://www.intel.com/content/dam/doc/manual/8255x-10-100-mbps-ethernet-controller-software-dev-manual.pdf> (accessed April 30, 2019).
40. **Supermicro.** Specifications: Broadcom NetXtreme® 57XX User Guide, [ftp://ftp.supermicro.com/ISO\\_Extracted/CDR-INTC\\_1.31\\_for\\_Intel\\_platform/Broadcom/Build8.1.3/Manuals/English/specs.htm](ftp://ftp.supermicro.com/ISO_Extracted/CDR-INTC_1.31_for_Intel_platform/Broadcom/Build8.1.3/Manuals/English/specs.htm) (accessed April 30, 2019).
41. **The CentOS Project.** CentOS Linux, <https://www.centos.org/about/> (accessed May 9, 2019).
42. **The CentOS Project.** CentOS Linux releases, <https://wiki.centos.org/FrontPage> (accessed May 9, 2019).

# Licence

## Non-exclusive licence to reproduce thesis and make thesis public

I, Janno Jõgeva

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,  
**Software Infrastructure and Course Design of a Robotics Course,**  
supervised by Eno Tõnisson.
2. I grant the University of Tartu a permit to make the work specified in p.1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in p.1 and p.2.
4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Janno Jõgeva

**16/05/2019**