Aalto University

School of Electrical Engineering

Degree Programme in Automation and Systems Technology

Sakari A. Pesonen

An open and general numerical control and machine vision based architecture for payment terminal acceptance test automation

Master's Thesis

San Jose, May 6, 2016

DRAFT! — September 10, 2016 — DRAFT!

Supervisor:	D.Sc. Seppo Sierla, Aalto University
Advisor:	M.Sc. Tatu Kairi

Aalto University
School of Electrical Engineering
Degree Programme in Automation and Systems Technology

ABSTRACT OF MASTER'S THESIS

Author: Sakari A. Pesonen

Title:
An open and general numerical control and machine vision based architecture for payment terminal acceptance test automation

Date:	May 6, 2016	**Pages:**	vii + 37
Major:	Intelligent Products	**Code:**	T-110
Supervisor:	D.Sc. Seppo Sierla		
Advisor:	M.Sc. Tatu Kairi		

Software testing is a crucial part of modern software development and it is commonly accepted fact that the earlier defects and errors in the software are found, the lower the cost of correcting those will be. Early detection of errors also increases the possibility to correct them properly.

Acceptance testing is a process of comparing the developed program to to the initial requirements. Acceptance testing of a system should be executed in an environment as similar as possible to the production environment. This master's thesis will discuss how to address these guidelines in automated acceptance testing environment of payment terminal software.

This master's thesis will discuss the theories related to software testing, testing of embedded systems and the challenges stated above. Master's thesis will present a proposed architecture for automated acceptance testing of payment terminals including the needed hardware and software.

Keywords: automated acceptance testing, software testing, payment terminal, robot framework, computer vision, open source

Language: English

Aalto-yliopisto
Sähtkötekniikan korkeakoulu
Automaatio- ja systeemitekniikan koulutusohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Sakari A. Pesonen		
Työn nimi:	Avoin ja yleispätevä numeeriseen ohjaukseen ja konenäköteknologioihin pohjautuva maksupäätteiden automaattisen hyväksymistestausympäristön arkkitehtuuri		
Päiväys:	6. toukokuuta 2016	**Sivumäärä:**	vii + 37
Pääaine:	Älykkäät tuotteet	**Koodi:**	T-110
Valvoja:	D.Sc. Seppo Sierla		
Ohjaaja:	M.Sc. Tatu Kairi		

Ohjelmistotestaus on tärkeä osa modernissa ohjelmistotuotannossa ja on yleisesti hyväksytty, että mitä aiemmin virheet ohjelmistosta löytyvät, sitä edullisempaa niiden korjaus tulee olemaan. Aikainen virheiden havaitseminen myös nostaa mahdollisuutta korjata virheet oikein.

Hyväksymistestaus on ohjelmistotestauksen vaihe, jossa kehitettyä ohjelmistoa verrataan alkuperäisiin ohjelmistovaatimuksiin. Ohjelmiston hyväksymistestaus tulisi suorittaa lopullista tuotantoympäristöä mahdollissimman hyvin vastaavassa ympäristössä. Tämä diplomityö käsittelee näitä ohjeistuistuksia maksupäätteiden autommaattisen hyväksymistestauksen ympäristössä.

Tämä diplomityö käsittelee ohjelmistotestaukseen liittyvää teoriaa, sulautettujen järjestelmien testausta ja yllä mainittuja haasteita. Diplomityö esittelee ehdotetun ympäristön maksupäätteiden automaattiseen hyväksymistestaukseen ja käsittelee siihen tarvittuja ohjelmistoja ja fyysisiä komponentteja.

Asiasanat:	automaattinen hyväksymistestaus, ohjelmistotestaus, maksupääte, robot framework, konenäkö, avoin lähdekoodi
Kieli:	Englanti

Acknowledgements

I wish to thank my instructor Tatu Kairi and my supervisor Seppo Sierla for their great help and knowledge throughout the writing process of the master's thesis.

San Jose, May 6, 2016

[illegible]

Sakari A. Pesonen

Abbreviations and Acronyms

CNC	Computer Numeric Control
UI	User Interface
LCD	Liquid Crystal Display
BW	Black and White
PIN	Personal Identification Number
RF	Robot Framework
AAT	Automated Acceptance Test
PWM	Pulse Width Modulation
OCR	Optical Character Recognition
QA	Quality Assurance
SUT	System Under Test
BDD	Behavior-Driven Development

Contents

Chapter 1

Introduction

Software testing is a crucial part of modern software development and it is commonly accepted fact that the earlier defects and errors in the software are found, the lower the cost of correcting those will be. Early detection of errors also increases the possibility to correct them properly. (*Myers et al. [2011]*)

Acceptance testing is a process of comparing the developed program to to the initial requirements (*Myers et al. [2011]*). Therefore especially in agile software development, automated acceptance testing (AAT) plays important role as new versions of software are being developed constantly and AAT phase should be executed whenever new features are added. Automation can free valuable human resources from this process (*Haugset and Hanssen [2008]*) and therefore lover the overall cost of the software.

According to *Sommerville [2011]* acceptance testing of a system should be executed in an environment as similar as possible to the production environment of the final product. System should also be tested with real data rather than with simulated sample. When the software being developed is actually embedded software and the production environment is actually real embedded system, in this case a payment terminal, the acceptance testing should be executed on actual payment terminal with actually interacting through the user interface (UI) of the machine. This also leads to a situation where concerns pointed out above are actually being emphasized as late detection of defects in embedded software can considerably raise the overall cost of the system (*Ebert and Jones [2009]*).

Sommerville [2011] states that it is practically impossible to perfectly replicate the system's working environment and when considering an embedded system, this can can be even harder. Buttons of the device have to be actually pressed and visual changes on the screen of the device has to be observed. In order to automate this, some sort of test environment has to be implemented that can observe and manipulate the device through physical word, i.e. not simulating the keystrokes nor reading the LCD communication line. Some kind of joint hardware and software solution has to be created and it also has to mimic real human user as realistically as possible.

This master's thesis will discuss the theories related to software testing, testing of embedded systems and the problems stated above. Master's thesis will present a proposed architecture for automated acceptance testing of payment terminal software including the needed hardware and software.

Research presented in this master's thesis was carried in co-operation with Eficode Oy and one of the main payment terminal software provider in the Nordic countries.

1.1 Problem statements

In order to survey the topic of this work in adequate level, this master's thesis will discuss four different problem statements. problem statements are as follows:

1. What are the benefits of using open source software and how can the architecture be designed to maximally exploit these benefits?

2. What are the distinguishing characteristics between different payment terminals that have impact on automated acceptance testing? How can the architecture be designed to adapt the system to different payment terminals with minimal effort?

3. What kinds of test automation approaches exist and which approach is best suited for payment terminal acceptance test automation?

4. How should test keywords used in test suites be defined to make the test suites compact and understandable? How should keywords be defined to make the tests reusable for other types of payment terminals?

1.2 Structure of the master's thesis

This master's thesis will first discuss the theories and literature related to the topic and will then present a proposed architecture of automated test environment for payment terminal software acceptance testing. In the first chapter of this master's thesis the topic will be introduced, problem statements will be presented and structure of this work will be explained.

Second chapter will cover the literature review of the topic of the master's thesis. Each problem statements will have related subsections and individual problem statements will be discussed on those sections. Each subsection will first give introduction on problem statement's point of view and it will be followed by the most relevant references around the topic. Subsections will point out what has been done earlier and how the fundamental aspects of these previous works can be used as a basis for this work.

Third chapter of the master's thesis will present the proposed architecture for automated acceptance test environment for payment terminal software based on literature review done on previous chapter. Chapter will present the fundamental parts of hardware and software needed for this kind of environment. This chapter will have diagrams of proposed software architecture as well as fundamental design of needed hardware.

Fourth and the final chapter will conclude the research done on this master's thesis and will summarize the benefits obtained by this kind of environment.

Chapter 2

Payment terminal acceptance testing

When developing software with agile methodologies for payment terminals, i.e for embedded system, testing is a crucial part of the process. The earlier the defects and errors in the software are detected, the lower the cost and needed effort will be for correcting those (*Myers et al. [2011]*).

Motivation for this research came from payment terminal software provider as they needed cost efficient and simple as possible automated acceptance test environment in order to lower the costs and speed up the acceptance testing phase of their software development.

In order to automate the acceptance testing of the payment terminals, test environment that can manipulate and observe the device through physical world has to be created. In other words, environment has to have some sort of a robot for pressing the buttons, screen of the device has to be observed and all this must be controlled by some kind of combination of software.

Test environment that can be used in acceptance testing of payment terminals has several challenges to tackle and matters related to physical and technical aspects of the payment terminals have to be considered. This chapter will discuss the background of these challenges. Customer also had a desire for open source technologies and this chapter will discuss the benefits obtained by using open source software and hardware in acceptance testing environment for payment terminals. Chapter will also discuss the different approaches for acceptance testing as well as how should the test suites be defined in order to make them understandable and reusable.

2.1 Benefits of Open Source solutions

When designing automated acceptance testing environment from scratch, evaluation and availability of different possible components play significant role in terms of development speed and costs. Suitability of one individual software subsystem is hard to determine just based on manual or documentation of the product. Software has to be evaluated in terms of functionality, stability and performance and different software decisions have to be compatible with each other. Software components might also need some modification to suit the needs of intended environment. All this applies also in a sense to the hardware parts as well.

Open source software provides advantage on these matters over closed source products as the source code is easily available (*Morgan and Finnegan [2007]*). As open source software can be accessed free of charge, component can be easily evaluated by trying out whether they work for the purpose or not. Evaluation can also include an analysis about how easily the open source product can be modified to suit the need. This especially is hard to achieve with closed source products as the source code is not available.

According to *Paulson et al. [2004]* open source projects usually have fewer defects than closed source projects. Defects are found and fixed rapidly as they are reported openly to the open source community. If defect is found during evaluation of the product, it can be also corrected by the user and by doing this the user can contribute to the project. This on the other hand is hardly never possible with closed software.

Paulson et al. [2004] also states that open source projects foster more creativity than closed source counterparts. This means that number of functions added over time is higher on open source projects. When using the product in some new field of use, this can be great advantage as user can report desired feature to the community and it can be added relatively quickly if the feature is considered needed by the community.

"Open source" hardware on the other hand means that details and plans of the product and parts are commonly available. This allows that parts can be manufactured and modified by anyone with knowledge and skills to suit individual needs. When detailed part descriptions are available, multiple manufacturers can fabricate the actual parts. This creates competition and therefore usually lowers the price of individual hardware components.

As the overall security of the payment terminals is a high priority, use of open source technologies can also be seen as an effort to fulfill this requirement. Open source products provide transparency to the actual users and therefore supports growing thrust amongst customers.

2.2 Common characteristics between payment terminals

When designing automated test environment for different kinds of payment terminals, different physical and technical features have to be taken into account. Environment has to be able to manipulate different types of payment terminals and test structure has to be designed to adapt to needs of different software and software versions running on payment terminals.

Majority of payment terminals share some common characteristics as they are made for same purpose: handling card payments. Scope of this thesis is to view those payment terminals that share three main features: keyboard, screen and card slot. Different types of terminals can be observed in Figure 2.1 and Figure 2.2 bellow.

Screens of the payment terminals differ in terms of size, placement and type. Test environment has to take into account different screen placements and it has to support both black and white (BW) and colored displays.

Keyboards of payment terminals share majority of keys together as number keys are needed for entering the PIN code and accept and decline buttons are needed for accepting and canceling the payment. Keyboard layouts, however, differ between different manufacturers and even amongst different models of the same manufacturer.

Location of the chip card slot is usually on the lover edge of the payment terminal or on top of the screen of the payment terminal. Research done within this master's thesis is limited to those terminals that have the chip card slot at the lower edge of the payment terminal as this simplifies the hardware needed for test environment. This is described more in depth in Chapter 3.2. This study is also limited to only chip card readers and magnetic stripe readers or near field communication (NFC) payments are not addressed.

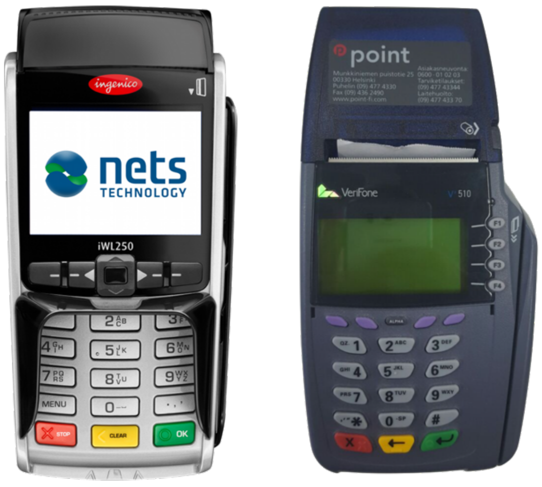


Figure 2.1: Two examples of payment terminals from different manufacturers. Left image from: `http://www.netskauppa.fi/images/t/24-85-PrimaryImage.image.ashx`

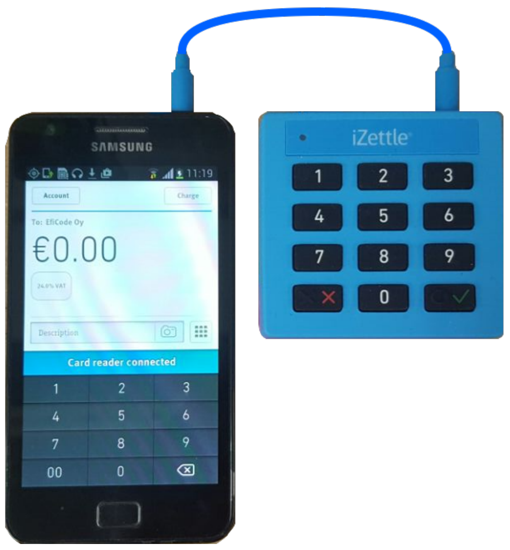


Figure 2.2: Example of a payment terminal which attaches to a smart phone.

2.3 Different approaches for test automation

Problem of testing the payment terminal software in automated way can be viewed at different levels. Most abstract division can be seen if the testing is divided into two levels: white box testing and black box testing. White box testing is a method where source code is investigated and test cases are written to test the internal logic of the program. Black box testing on the other hand concentrates only on the inputs and the outputs of the software. Everything between those is not in a field of interest and black box testing only focuses on whether the right input produces the wanted output. (*Myers et al. [2011]*)

Huizinga and Kolawa [2007] on the other hand presents that test automation can be divided into several other layers that are unit testing, integration testing, system testing and acceptance testing. Unit testing is defined to cover testing of single unit of the software source code e.g. individual methods and functions of the software. Integration testing is described as a testing phase to verify that different parts of the software work together as a group.

System testing is described being a testing phase where hardware and software is integrated and tested to meet the requirements of the system. This can however include simulated data. Acceptance testing is represented as highest abstraction level of this division and it is described to ensure that the final product meets its acceptance criteria within the customers.

Khan and Khan [2012] distinguishes these methods clearly from each other by stating that white box testing is a process where full knowledge of source code is needed in order to write the tests. Black box testing is described in a way that only fundamental aspects of the application has to be known and black box testing has no or only little relevance to internal works of the program (*Pressman [2005]*). Black box testing techniques can be thus seen to apply for testing working product against the initial requirements of the software. In this way white box testing can be distinguished to cover unit and integration testing part of the software testing and the black box testing can be seen covering the acceptance testing part of the testing.

As black box testing is based only to the external exceptions and behavior of the software (*Khan and Khan [2012]*), acceptance testing of the payment terminal software can be seen to follow this methodology. Intended automated testing of the payment terminals seems to also follow the acceptance testing phase of the division made by *Khan and Khan [2012]*.

Acceptance testing of a payment terminal software can be seen as a testing phase where the UI of the device and the use cases of the device are tested at the final production level, i.e. through using the real buttons of the device under test and observing that the expected messages can be seen through the screen of the device under test. This can be seen as an effort to automate real human user using the payment terminal.

2.4 Test suite syntax

Test suite syntax plays significant role in automated acceptance testing environment of payment terminals in terms of test readability, reusability and adaptivity. When building automated acceptance testing environment, the tests should be understandable enough that whole development team and people related to the project can easily adopt the test syntax.

According to the well recognized guidelines of test automation by *Bach [1996]*, test automation and the process that it automates should be kept carefully separated. Test automation should be built in a form that it is easy to review and distinct from the process that it automates. These guidelines should be taken into account also when determining suitable test framework and test suite syntax.

When evaluating suitable test automation framework, it should be recognized, that shared knowledge is a key factor of successful test automation. Software projects usually involves some sort of quality assurance (QA) or even individual a QA team. Projects tend involve also fair amount of people with no technical background or coding skills and still their responsibilities involve guaranteeing the quality of the software. *Mosley and Posey [2002]* recognizes that high level test languages help to share the knowledge amongst the people that are responsible of the product. When information and knowledge is shared it helps in achieving the objectives of test automation and builds up the morale amongst the people that are involved.

Lowell and Stell-Smith [2003] states that acceptance tests should be easy as possible to write or otherwise people working with the project will not write the tests as the task is seen unpleasant. Tests must be also easy to maintain and also people that have not written the tests should be able to modify the tests to suit updated program features. For this reason the test cases should be human readable and understandable also to non-technical people. Test steps should be self explanatory and unambiguous.

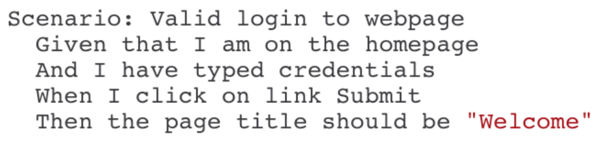
Test cases of a payment terminal software acceptance testing contain relatively high amount of repetition as for example test step of inserting PIN code is the same whether right or wrong PIN code is inserted or whether the test case would validate credit or debit payment. For this reason, test case syntax should be as modular as possible in order to allow reuse of keywords with different parameters. Easily reusable keywords also allows fast creation of new test cases.

Tests can be essentially written in some conventional coding language, for example Java or Python, or by using some higher level language. There are many widely used test frameworks available for conventional coding languages, for example jUnit for Java (*JUnit*). This, however, requires coding experience to some extend to be able to understand and write the tests or modify the existing ones. As it is stated above, combined with the guideline for writing tests understandable enough, usage of conventional coding languages can be seen opposing the best practices. On the other other hand, test case syntax must be versatile enough to accommodate different kinds of testing scenarios and needs. This leads to situation where the abstraction level of the test cases has to be considered thoroughly.

```java
@Test
public void validLogin() throws Exception {
    try {
        openBrowser();
        inputUsername("demo");
        inputPassword("mode");
        clickElement(By.name("Submit"));
        pageTitleShouldBe("Welcome")
    } catch (Exception e) {
        log.error(getClass().getName() + " failed. Exception:", e);
        takeScreenshot(getClass().getName() + "_failed");
        System.out.println(driver.getPageSource());
        throw e;
    }
}
```

Figure 2.3: Example of a jUnit test case that tries to login to website.

In addition to the test frameworks utilizing the use of some conventional coding language for test cases, there are also couple of well recognized tests frameworks available that uses more human-like language for writing the tests. These frameworks usually use the same libraries for interacting with the system under test (SUT) as more low level frameworks but they allow more higher level syntax of the actual test scripts. One fairly popular example of this kind of more high level test framework is Cucumber. Cucumber is an open source acceptance test framework that utilizes behavior-driven development (BDD) style (*Cucumber*). Cucumber uses Gherkin language that is designed to be human readable without previous knowledge of coding (*Gherkin*). This means that also business oriented people involved with the project can understand the test cases.

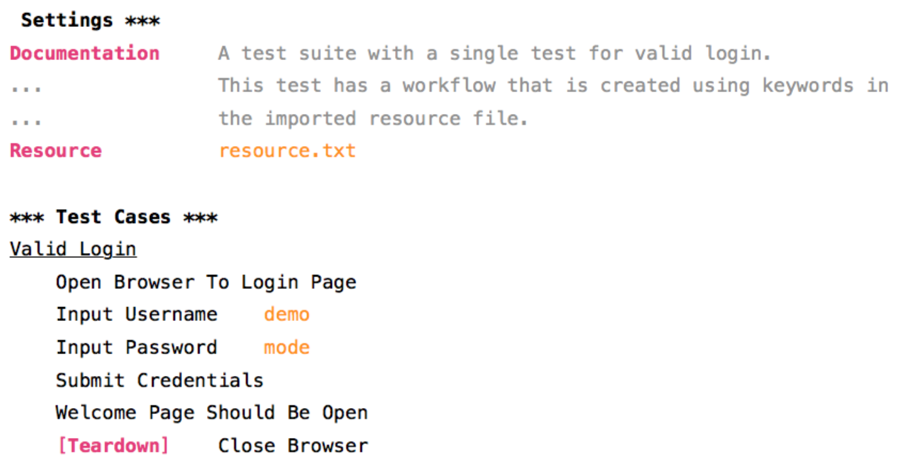


```
Scenario: Valid login to webpage
  Given that I am on the homepage
  And I have typed credentials
  When I click on link Submit
  Then the page title should be "Welcome"
```

Figure 2.4: Example of a simple Cucumber test scenario and use of Gherkin language.

Another good example of higher level test frameworks is Robot Framework (RF). RF is an example of generic keyword driven test automation framework (*Robot Framework*). RF allows creation of human readable test cases and reusability and extendability of high-level keywords is made relatively easy (*Stresnjak and Hocenski [2011]*). *Nokia Solutions and Networks [2015]* also outlines that RF has highly modular software architecture allowing it to be easily connected to any kind of SUT by using different test libraries.

Example of a Robot Framework test case can be seen in Figure 2.5 bellow. It is easy to see the intended test case execution by looking the test case and this will be the goal for the environment proposed later on in this master's thesis.



```
 Settings ***
Documentation     A test suite with a single test for valid login.
...               This test has a workflow that is created using keywords in
...               the imported resource file.
Resource          resource.txt

*** Test Cases ***
Valid Login
    Open Browser To Login Page
    Input Username    demo
    Input Password    mode
    Submit Credentials
    Welcome Page Should Be Open
    [Teardown]    Close Browser
```

Figure 2.5: Example of a simple test suite. Source: `http://robotframework.org`

Chapter 3

Proposed architecture

Based on the aspects pointed out on Chapter 2, this part of the master's thesis will present a proposed architecture for automated acceptance testing environment for payment terminal software. Components of the environment can be divided into hardware and software components and this chapter is divided accordingly.

Research done in this master's thesis was carried out in co-operation with Eficode Oy and one of its customers providing payment terminal software in the Nordic countries. Motivation for this project came from the customer and Eficode Oy took responsibility of implementing the system according to the best practices of the industry. This proposal was initial plan for the project and it will be presented in this chapter.

3.1 Overview

When planning an automated acceptance test (AAC) environment for payment terminal software, environment has to be highly adaptive for different types of hardware and software features of different payment terminal models. This proposal was done for one payment terminal software provider and they had several different models of payment terminals and altogether 51 different software configurations for those devices.

Security is a top priority of payment terminal electronics and software and it is not possible to access internals of the payment terminal hardware. This means that AAC environment has to be able to manipulate the physical interface of the device. This also creates requirement for supporting different types of keyboard layouts and screen locations. In other words, environment has to be non dependent on single manufacturer or payment terminal model.

One of the requirements for the AAC environment was also use of open source technologies. For the reasons pointed out in Section 2.1, customer wanted that the environment is as open as possible. This also creates reputation and visibility regarding the security matters.

Other requirements for the AAC environment was simplicity, low cost, low need for maintenance and ability to run the tests 24/7.

3.2 Hardware

Hardware for this proposal was intendedly kept simple and low-cost as possible. This proposal presents the use of just one Raspberry Pi 2[1] computer as a main computer for AAC environment. Raspberry Pi 2 in relatively inexpensive compared to its computing power and it can also run full Linux operating system. It is small sized and does not require any cooling equipment. Therefore it suites well to this project as it can be situated easily to the environment and can be run over the clock without concerns about wearing cooling fans for example.

AAC environment also requires computer vision as changes on the screen have to observed. Manufacturer of Raspberry Pi offers low-price solution for this as a form of Raspberry Pi Camera Module[2]. This module is proposed for use in computer vision tasks of the AAC environment.

[1]`https://www.raspberrypi.org/products/raspberry-pi-2-model-b/`
[2]`https://www.raspberrypi.org/products/camera-module/`

3.2.1 The Robot

To be able to manipulate the physical UI of the payment terminals, some sort of robot is needed. Robot should be therefore able to accommodate different types of payment terminals and be able to press all types of buttons in question. Low cost and low need for maintenance are also requirements for this robot. Robot should also be able to accomodate multiple payment terminals to the working area at the same time in order to allow parrallel execution of acceptance tests. This is intended to speed up the overall process as the same tests have to be run on different models of payment terminals. Other option would also be to make changing of the device under test easy and fast so that the manual work required can be minimized.

The master's thesis proposes the use of ShapeOko 2[3] Computer Numerical Control (CNC) milling machine to be used as a manipulator. Though the machine is intended for milling purposes, it can be turned into a portal robot when milling tool is removed.

ShapeOko 2 is an open-source project and plans of the machine are openly available on their GitHub[4]. This allows easy modifications to the hardware parts of the robot if needed.

ShapeOko 2 is controlled by an Arduino board running a program called GRBL[5]. GRBL is an open source, high performance G-code interpreter and it is used for controlling CNC milling machines in general[6]. G-code command are sent from Raspberry Pi 2 to the Arduino on the robot using serial communication.

Robot should be equipped with a pushing tool that can be manipulate the buttons. Pushing tool can be easily manufactured using for example 3D printing techniques.

[3]`http://www.shapeoko.com/wiki/index.php/ShapeOko_2`
[4]`https://github.com/shapeoko/Shapeoko_2`
[5]`https://github.com/grbl/grbl`
[6]`http://www.shapeoko.com/wiki/index.php/Software`

3.2.2 Card Feeder

Insertion and removal of the credit card might be hard to accomplish in a simple way using just the robot described in previous section. This work proposes the use of generally designed card feeders to accomplish this task. Proposed card feeders consist of 3D printed base plate that attaches to the payment terminal, servo motor and 3D printed tray that attaches to the servo and to the credit card.

Card feeders are designed in a way that they can be used with any types payment terminals that have the card slot at the bottom edge of the device. Standard hobby servos are used as servo motors in order to keep the cost of the setup low.

Arduino board will be used to drive the servos as it can easily provide the needed pulse width modulated (PWM) signal for the servos. Arduino is suggested in order to ensure quality and accuracy of the PWM signal compared to what can be produced easily with non real time operating system running on the Raspberry Pi. Raspberry Pi on the machine will communicate with Arduino through serial communication.

3.3 Software

As stated in the Chapter 4.2.4, automated acceptance tests should be simple and understandable enough to actually make the automated testing efficient and beneficial. Open source solutions should be favored as this was requested by the customer and to achieve benefits described in the Chapter 2.1 For these reasons software decisions of the AAT environment should be carefully considered in order to achieve good maintainability, compatibility and overall simplicity.

For software part of this AAC environment, Raspbian Wheezy is proposed for operating system. Raspbian is the official supported operating system for

Raspberry Pi by Raspberry Pi Foundation[7]. Raspbian is based on widely used Debian Unix-like operating system. This allows the use of components developed for Debian to be used with this AAC environment.

3.3.1 Test Framework

Robot Framework[8] is proposed for the test framework. RF is an open-source, generic, keyword-driven test automation framework that has human readable test case syntax (*Nokia Solutions and Networks [2015]*).

Robot Framework also has highly modular software architecture (*Nokia Solutions and Networks [2015]*) and this allows the framework to be used with any kinds of testing libraries to connect to the system under test. This feature can be seen as a great advantage when implementing test libraries for machine control and computer vision (Chapter 3.3.2). Illustration of this modular architecture can be seen in Figure 3.1 bellow.

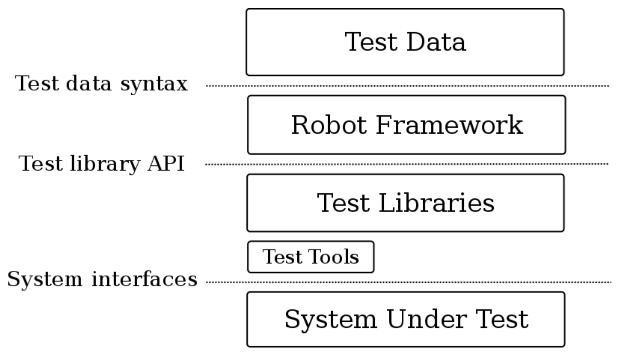
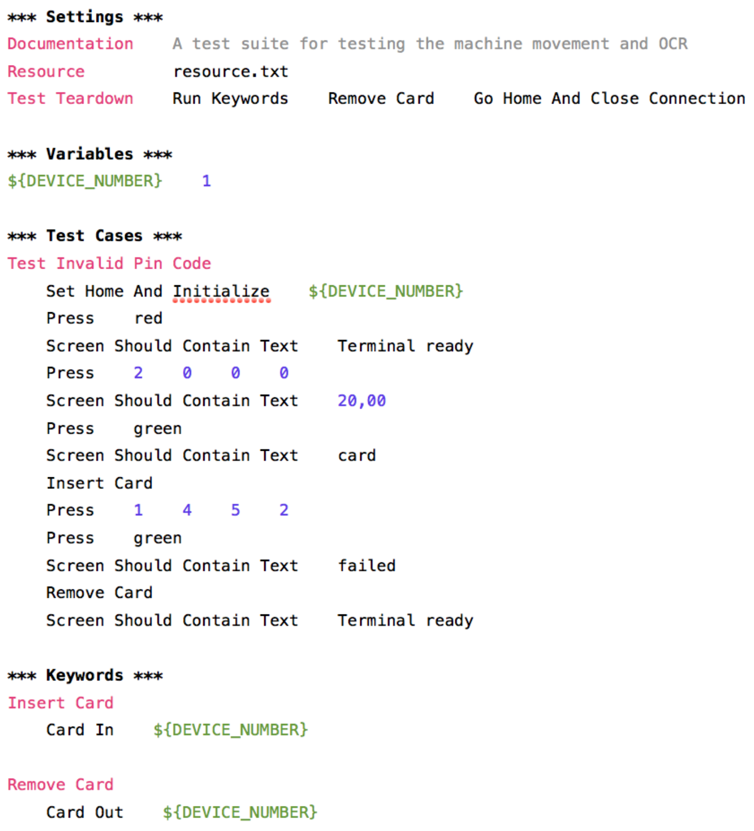


Figure 3.1: High level modular architecture. Source: `http://robotframework.org/img/architecture-big.png`

When RF tests are being run, it generates clear report and log files of the test case execution results (*Nokia Solutions and Networks [2015]*). These files offer high level view of all test cases and step-by-step descriptions of individual test cases in order to make the debugging more easy.

[7]`https://www.raspberrypi.org/downloads/raspbian/`
[8]`http://robotframework.org/`

Example of intended test case can be seen in Figure 3.2. This test case describes automated RF acceptance test for entering invalid PIN code when trying to execute card purchase.



```
*** Settings ***
Documentation     A test suite for testing the machine movement and OCR
Resource          resource.txt
Test Teardown     Run Keywords    Remove Card    Go Home And Close Connection

*** Variables ***
${DEVICE_NUMBER}    1

*** Test Cases ***
Test Invalid Pin Code
    Set Home And Initialize    ${DEVICE_NUMBER}
    Press    red
    Screen Should Contain Text    Terminal ready
    Press    2    0    0    0
    Screen Should Contain Text    20,00
    Press    green
    Screen Should Contain Text    card
    Insert Card
    Press    1    4    5    2
    Press    green
    Screen Should Contain Text    failed
    Remove Card
    Screen Should Contain Text    Terminal ready

*** Keywords ***
Insert Card
    Card In    ${DEVICE_NUMBER}

Remove Card
    Card Out    ${DEVICE_NUMBER}
```

Figure 3.2: Example test case for invalid PIN code test

3.3.2 Test Libraries

As can be seen on Figure 3.1, architecture requires external libraries to connect to the system under test. In this case those libraries would be a library for machine control, a library for computer vision and a library for card feeder manipulation. All these libraries can be written using Python programming language that is supported out of the box by Robot Framework (*Robot Framework*).

For machine control library the environment has to be able to send G-code command through USB serial communication to Arduino on the robot. For this pySerial[9] library is proposed.

For the computer vision task of the environment, messages on the display are usually those that need to be verified. For this character recognition is needed. Open source optical character recognition (OCR) engine called Tesseract OCR[10] is proposed. It was initially developed by HP but since 2006 it has been developed by Google. In order to use Tesseract OCR with Python, pytesseract[11] wrapper is needed.

Library for controlling the card feeders is the most simplest one of these three libraries. For this pySerial[12] library is proposed to send the serial communication command to the Arduino controlling the card feeders.

[9] `http://pythonhosted.org/pyserial/`
[10] `https://github.com/tesseract-ocr/tesseract`
[11] `https://pypi.python.org/pypi/pytesseract`
[12] `http://pythonhosted.org/pyserial/`

Chapter 4

Results and Evaluation

This chapter will cover the subsystems and steps taken that were needed to achieve the testing environment described in Chapter 3. This chapter will first discuss the arrangements related to the hardware of the framework and then software related arrangements will be presented and described. After presenting the built framework, achieved results will be discussed and finally the test environment presented in this Thesis will be evaluated.

4.1 Hardware arrangements

AAT environment presented in this Master's Thesis consists of several different hardware components. Environment had to be a smooth combination of manipulation and computing hardware and hardware architecture is thought to be modular in a sense that every component has a specific functionality. This allows easy maintenance and upgrade of each subsystem.

As stated in Chapter 3, one of the requirements for this AAT environment was affordable price. For this reason hardware decisions have been made taking quality-price ratio into consideration and hobby-grade electronics were used widely through out the environment. 3D printing was also utilized as a manufacturing technique of custom made components for its relatively low manufacturing price and relatively acceptable quality of outputted plastic parts.

Main components of the AAT environment are the robot that handles the manipulation of the payment terminals, Raspberry Pi 2 Model B single-board computer which was used as a main computer of the environment, two Arduino Uno boards for more specific control needs of certain components, camera for machine vision and 3D printed payment card feeders for the payment terminals. These subsystems and components are described in following subsections.

4.1.1 The Robot

As suggested in Section 3.2.1, ShapeOko 2 open-source 3-axis CNC milling machine was used as robot manipulating the payment terminals. ShapeOko was built according to the instructions found from the homepage of the project (*ShapeOko 2*). Construction was altered only regarding to the tool that was used as the spindle motor was substituted by 3D printed pushing tool.

ShapeOko 2 has a working area of about 300 mm x 300 mm x 60 mm and this means that it can accommodate up to three payment terminals at same time to the working area. This allows parallel test case execution i.e. test cases can be run at the same time with different terminals. Arrangement of the devices was implemented by dividing the work area into three sections. Each payment terminal was attached to a standard sized MDF-plate and each section of the working are can accommodate one of these MDF-plates. Holes were drilled in to the working area and nuts were inserted in to these holes at the back of the work bench. MDF-plates attach to these holes with screws enabling easy installation and removal of plates with different models of payment terminals. MDF-plates can be observed in Figure 4.6.

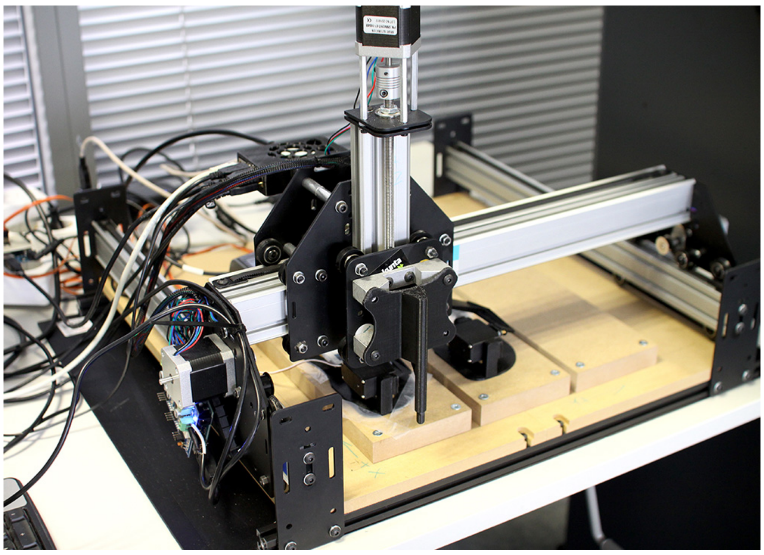


Figure 4.1: Robot in its production state.

Each axis of the robot is controlled by stepper motors. Use of stepper motors instead of servo motors offers affordable way of controlling each axis in a relatively fast and reliable manner. X- and Z- axises are both manipulated using one stepper motor on each axle and bigger Y-axis is manipulated using two parallel stepper motors. Manipulation of payment terminal buttons stresses the machine much less than actual milling of materials that the machine is designed for and this allows faster movement of the machine that would be possible when executing actual milling job.

The robot was controlled using G-code that was sent from the main computer to an Arduino Uno attached to the robot. Arduino Uno and the main computer were connected via USB connection. More detailed description of the electronics can be found from Section 4.1.2.

Section 3.2.1 suggested equipping the robot with a pushing tool and this was implemented to the final solution by 3D printing the tool from PLA plastic. Tool consisted of two parts: cylindrical beam and a stem inside of it. Stem slides inside the beam and the two parts are segregated with a spring. Spring provides the needed attenuation in order to forgive slight misalignments and too long trajectories when pushing the buttons of the payment terminals. Pushing tool can be observed in Figure 4.2 and Figure 4.1.

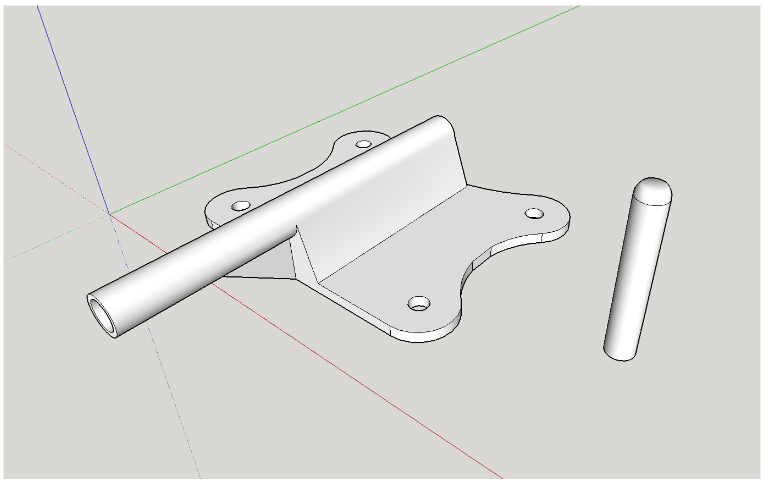


Figure 4.2: CAD design of the pushing tool. Metal spring is inserted inside to the cylinder and the stem on the right side of the image slides to the cylinder.

4.1.2 Computing hardware

Raspberry Pi 2 Model B single-board computer is used as a main computer of the AAT environment. Raspberry Pi provides optimal computing power compared to it's price and has big community of users and developers world wide. 3D printed enclosure was manufactured to protect the computer board and it was attached to the moving Z-axis assembly of the robot.

In addition to the Raspberry Pi 2, the robot also has two Arduino Uno boards for handling some specific functionalities of the AAT environment. One Arduino Uno is interpreting the G-code commands sent from the Raspberry Pi and it is connected to the stepper motors of the robot through a stepper motor driver shield[1].

[1]`http://www.shapeoko.com/wiki/index.php/GrblShield/`

Second Arduino Uno is handling the servo motor control of the card feeders. It is connected to the Raspberry Pi via USB connection and control commands to Arduino Uno are sent using serial communication. Arduino Uno board provides PWM signal to the servo motors and can accommodate three card feeders at the same time. Self-made circuit board was fabricated and attached on top of the Arduino Uno board in order to make connecting the servo motor cables easy.

Connection diagram and main electronic components are visualized in Figure 4.3.

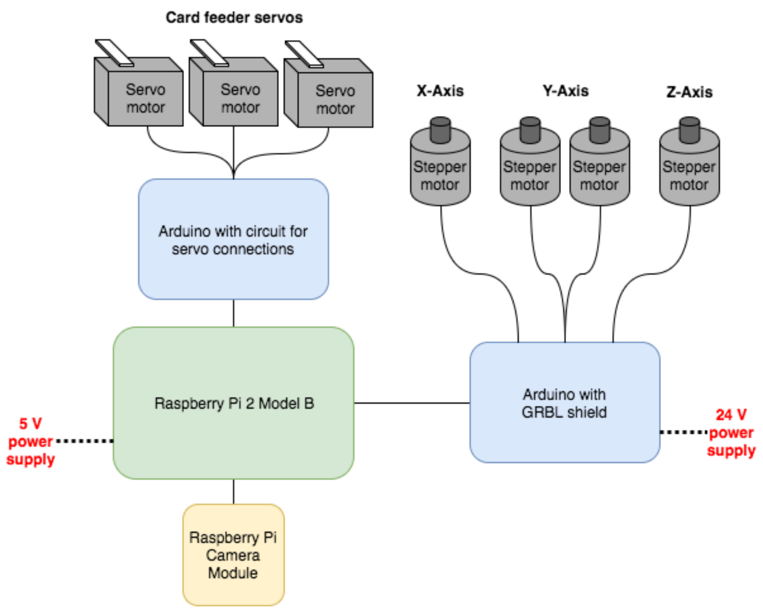


Figure 4.3: Main electronic components and connection diagram of the robot. Note that Y-axis is manipulated using two stepper motors.

4.1.3 Camera arrangements

As suggested in Section 3.2, Raspberry Pi's own camera module was used for machine vision hardware. Camera was attached to the bottom of the Raspberry Pi's enclosure and the enclosure was attached to the Z-axis assembly of the robot to the opposite side where the pushing tool is. Camera can be moved within the X- and Y-axis. Z-axis movement of the camera isn't possible. Depth of focus of the camera provides clear image of the screen even when the distance between the lens and the screen differs slightly between different payment terminal models. Camera attachment can be seen in Figure 4.4.

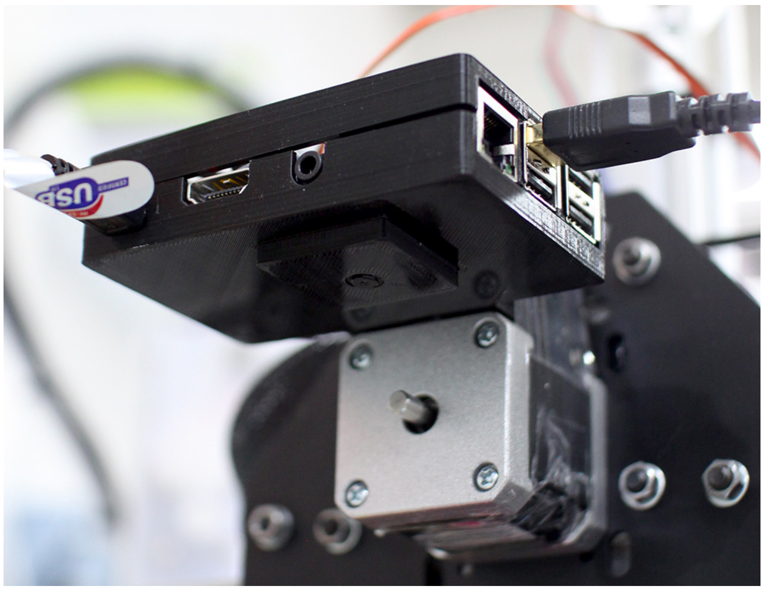


Figure 4.4: Camera is attached to the bottom of the Raspberry Pi's enclosure. Image also shows the attachment of the Raspberry Pi enclosure to the Z-axis assembly of the robot.

4.1.4 Card feeder arrangements

As suggested in Section 3.2.2, card feeder structures were 3D printed using PLA plastic. Finalized card feeders consisted of bottom plate, payment card holder and servo motor. Servo motor attaches directly to the bottom plate and card holder attaches to the arm of the servo motor.

Simplistic design can be used with different kinds of payment terminals which have the card slot at the bottom edge of the device. Flexibility provided by the plastic structure and the payment card itself allows the solution to be compatible with most of the payment terminals of this type. Design of the card feeders is presented in Figure 4.5. Figure 4.6 shows ready part installed to the environment presenting the servo installation and attachment of the card holder to the servo arm.

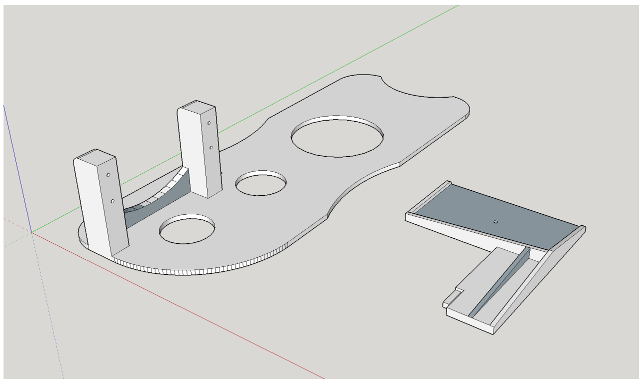


Figure 4.5: CAD design of the card feeder. Servo motor attaches to the bigger plate on the left and card holder on the right attaches to the arm of the servo motor. Card holder is designed to fit standard sized payment card.

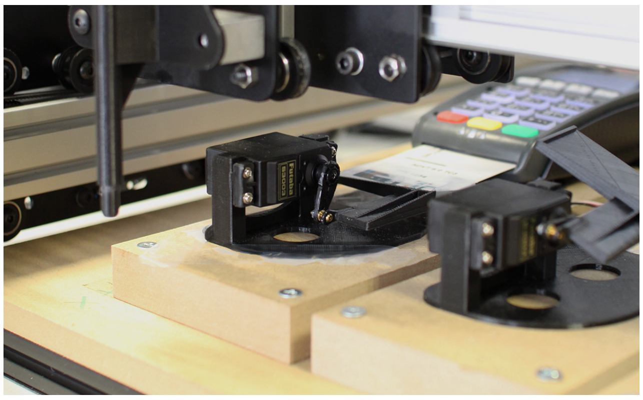


Figure 4.6: Card feeder installed to the environment. Image also presents the idea of MDF-plates described in Section 4.1.1.

4.2 Software arrangements

As proposed in Section 3.3, this chapter will describe decisions and arrangements regarding to the software point of view of the AAT environment. Proposal was followed rather loyally though some additional arrangements had to be implemented to the environment in order to increase usability and effectiveness.

Modular architecture was implemented also to the software level similar to the hardware level. Implementation only included open source or self made software components from the operating system to individual software libraries used in the AAT environment.

This section describes the individual software components of the AAT environment and their usage and function in the whole system. System configuration, test framework and libraries and the final test suite syntax will be presented.

4.2.1 Software architecture

Suggested already in the Section 3.3, Raspbian Wheesy Debian-based operating system was used with the Rasbperry Pi 2 Model B single-board computer. Operating system was used to run the test framework, test libraries and other software components and to handle the communication with different subsystems of the AAT environment.

Robot Framework was used as a test framework for its modularity, simplicity and versatility. RF was run on top of Python runtime environment and all test libraries were written using Python programming language[2]. Python test libraries were implemented to handle the needed serial communication to the Arduino board on ShapeOko 2 and to the other Arduino board used for controlling the card feeder servo motors. Picamera[3] Python library was used for providing the needed Python interface for communication with the

[2]https://www.python.org/

[3]http://picamera.readthedocs.io/en/release-1.12/

Raspberry Pi camera module.

As different keyboard layouts have to be supported, configuration files for keyboard layouts were implemented. There are two types of configuration files: one for device locations in the working area of the robot and one for each keyboard layout. In this way configuration of devices under test can be easily modified at the test case level.

Overall visualization of the software architecture can be observed in Figure 4.7.

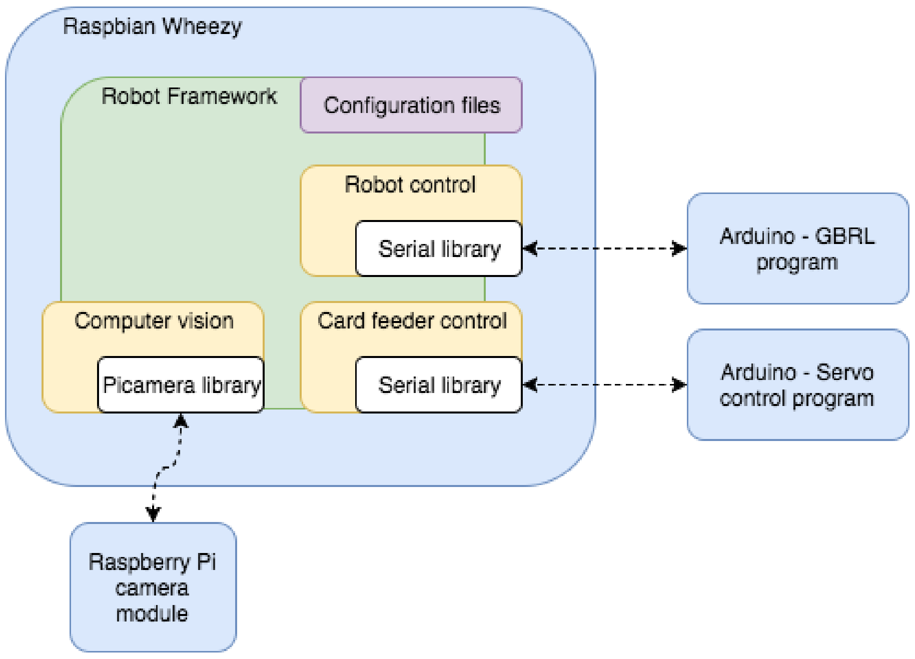


Figure 4.7: Simplified visualization of the software architecture of the AAT environment.

4.2.2 Robot Framework and libraries

4.2.3 Computer vision arrangements

4.2.4 Test syntax

4.2.5 Test result syntax

4.3 Results

4.4 Evaluation

Chapter 5

Discussion

At this point, you will have some insightful thoughts on your implementation and you may have ideas on what could be done in the future. This chapter is a good place to discuss your thesis as a whole and to show your professor that you have really understood some non-trivial aspects of the methods you used…

Chapter 6

Conclusions

This seminar report presented a proposal for automated acceptance testing environment for payment terminal software and addressed the theories and problems related to the topic. Presented AAC environment was joint combination of open source hardware and software and was formed by the requirements of a Eficode Oy's customer.

Literature review of this seminar report addressed the four problem statements introduced in the beginning of this seminar report.

This seminar report also lays a promise of how commonly and inexpensively available components can be used in relatively demanding applications. By combining different open source products, highly adaptive AAC environment could be possibly created for the needs of automated acceptance testing of payment terminal software.

Bibliography

James Bach. Test automation snake oil. *Windows Tech Journal*, 10, 1996.

Cucumber. URL `https://cucumber.io/` Accessed 7.9.2016.

Christof Ebert and Capers Jones. Embedded software: Facts, figures, and future. *Computer*, (4):42–52, 2009.

Gherkin. URL `https://github.com/cucumber/cucumber/wiki/Gherkin` Accessed 7.9.2016.

Borge Haugset and Geir Kjetil Hanssen. Automated acceptance testing: A literature review and an industrial case study. In *Agile, 2008. AGILE'08. Conference*, pages 27–38. IEEE, 2008.

Dorota Huizinga and Adam Kolawa. *Automated defect prevention: best practices in software management.* John Wiley & Sons, 2007.

JUnit. URL `http://junit.org/` Accessed 7.9.2016.

Mohd Ehmer Khan and Farmeena Khan. A comparative study of white box, black box and grey box testing techniques. *Int. J. Adv. Comput. Sci. Appl*, 3(6), 2012.

Charles Lowell and Jeremy Stell-Smith. Successful automation of gui driven acceptance testing. In *Extreme programming and agile processes in software engineering*, pages 331–333. Springer, 2003.

Lorraine Morgan and Patrick Finnegan. Benefits and drawbacks of open source software: an exploratory study of secondary software firms. In *Open Source Development, Adoption and Innovation*, pages 307–312. Springer, 2007.

D.J. Mosley and B.A. Posey. *Just Enough Software Test Automation*. Just enough series. Prentice Hall PTR, 2002. ISBN 9780130084682. URL `https://books.google.com/books?id=PEBvfWESIt4C`.

G.J. Myers, C. Sandler, and T. Badgett. *The Art of Software Testing*. IT-Pro collection. Wiley, 2011. ISBN 9781118133156. URL `https://books.google.fi/books?id=GjyEFPkMCwcC`.

Nokia Solutions and Networks. Robot Framework User Guide, 2015. `http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html/`. Accessed 6.5.2016.

James W Paulson, Giancarlo Succi, and Armin Eberlein. An empirical study of open-source and closed-source software products. *Software Engineering, IEEE Transactions on*, 30(4):246–256, 2004.

R.S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill higher education. Boston, 2005. ISBN 9780073019338. URL `https://books.google.com/books?id=bL7QZHtWvaUC`.

Robot Framework. URL `http://robotframework.org/` Accessed 1.6.2016.

ShapeOko 2. URL `http://www.shapeoko.com/wiki/index.php/ShapeOko_2` Accessed 1.6.2016.

I. Sommerville. *Software Engineering*. International Computer Science Series. Pearson, 2011. ISBN 9780137053469. URL `https://books.google.fi/books?id=l0egcQAACAAJ`.

Stanislav Stresnjak and Zeljko Hocenski. Usage of robot framework in automation of functional test regression. In *ICSEA 2011 The Sixth International Conference on Software Engineering Advances*, 2011.

Appendix A

First appendix

This is the first appendix. You could put some test images or verbose data in an appendix, if there is too much data to fit in the actual text nicely.

For now, the Aalto logo variants are shown in Figure A.1.

A?
Aalto University
School of Electrical
Engineering

(a) In English

A!
Aalto-yliopisto
Sähkötekniikan
korkeakoulu

(b) Suomeksi

A”
Aalto-universitetet
Högskolan för
teknikvetenskaper

(c) På svenska

Figure A.1: Aalto logo variants