MISK-Moves

Bo Elovson Grey

CERTEC | DEPARTMENT OF DESIGN SCIENCES
FACULTY OF ENGINEERING LTH | LUND UNIVERSITY
2023

ENGINEERING THESIS

MISK-Moves

Using your body as a musical instrument

Bo Elovson Grey

LUND
UNIVERSITY

MISK-Moves

Using your body as a musical instrument

Published by
Department of Design Sciences
Faculty of Enginnering LTH, Lund University
Box 118
SE-221 00 LUND
Sweden

Subject: Interaction Design (MAML05)
Division: Certec
Supervisor(s): Charlotte Magnusson, charlotte.magnusson@certec.lth.se and
Héctor Caltenco, hector.caltenco@certec.lth.se
Examiner: Kirsten Rassmus-Gröhn, kirsten.rassmus-grohn@certec.lth.se

Abstract

Move-to-play is a musical instrument for persons with both cognitive and physical impairments, who would have trouble playing traditional instruments. Everyone, no matter their abilities, are given the chance to play and control music by moving their own body. This project is part of the MISK project, which is a collaboration between Certec, Furuboda folkhögskola and Eldorado resurscenter. To make this possible, a machine learning model for the detection of people and their limbs, pose estimation, is used. This enables the translation of body movement into data used for controlling music software, which is made as another part of the MISK project. It runs under Max 8 by Cycling '74. To make it easy and affordable for other creators to recreate the project, relatively cheap hardware has been used, a NVIDIA Jetson Nano and a Raspberry Pi V2 camera. It is a system-on-module, which compact size and low power consumption makes placement even in small compartments easy. The machine learning model used is ResNet-18 pose, which proved to have adequately sufficient precision and performance to track people and their limbs in real time. It has not mattered whether a person has been standing or sitting, playing the instrument has worked equally well in both cases. Move-to-play has some latency however, up to about 150 ms. Users will have to be selective about what or how the instrument should control the music, since different people have different sensitivity to the connection between their own actions, latency and the effect of their actions on the music.

Keywords: MISK, machine learning, ResNet-18, NVIDIA Jetson Nano, system-on-module, Certec, Furuboda KompetensCenter, Eldorado resurcenter, pose estimation, object detection, object tracking, Raspberry Pi V2 camera, Max 8

Sammanfattning

Move-to-play är ett musikinstrument för personer som genom såväl kognitiva som fysiska funktionsnedsättningar skulle ha svårigheter att spela traditionella instrument. Alla, ovsett förmågor, ges chansen att spela, eller påverka musik genom att röra sin egen kropp. Det här arbetet är en del av MISK-projektet som är ett samarbete mellan Certec, Furuboda folkhögskola och Eldorado resurscenter. För att möjliggöra det här används en maskininlärningsmodell för detektering av personer och individuella lemmar, så kallad pose estimation. På så vis översätts deras rörelser till data som kan användas för att styra ett musikprogram, utvecklat i en annan del av MISK-projektet, som körs under Max 8 från Cycling '74. För att göra projektet enkelt och överkomligt att återskapa för andra intresserade används relativt billig hårdvara för att köra maskininlärningen på. nämligen en NVIDIA Jetson Nano med en Raspberry Pi V2-kamera. Hårdvaran är ett modulsystem med och dess kompakta storlek och låga effektförbrukning medför att den är enkel att placera även i relativt trånga utrymmen. Den maskininlärningsmodell som används är ResNet-18 pose, vilken visade sig ha tillräckligt god precision och prestanda för att kunna spåra personer och deras lemmar i realtid. Det har inte spelat någon roll ifall personen som spelar har varit sittande eller stående, musicerandet har fungerat lika väl i båda fallen. Move-to-play har dock en del fördröjning, upp till cirka 150 ms, vilket gör att en användare får välja hur och vad instrumentet ska styra med omsorg, eftersom olika personer är olika känsliga för kopplingen mellan deras egna handlingar, fördröjningen och effekten handlingarna har på musiken.

Nyckelord: MISK, maskininlärning, ResNet-18, NVIDIA Jetson Nano, modulsystem, Certec, Furuboda KompetensCenter, Eldorado resurscenter, pose estimation, objektdetektering, objektspårning, Raspberry Pi V2 kamera, Max 8

Acknowledgements

First I would like to thank my supervisors Charlotte Magnusson and Héctor Caltenco for providing valuable ideas and comments, and for pointing things out that needed to be pointed out. Without you two, nothing would have worked at all. Huge thanks go out to John Säbom, the developer of the MISK patcher. You were able to make Move-to-play sound really good, and the workshops together have given me great insight into what the application needed to do and what I needed to implement or change for things to work as well as possible. Every session has made Move-to-play take a large step forward. Thanks go out to David Wirzén and Sara Rundström at Eldorado for showing me how the music sessions work. I would also like to thank my examiner Kirsten Rassmus-Gröhn, without whom this thesis work would not exist. I was originally enquiring about another thesis project, but after some discussion the idea of joining the MISK project was floated. I do not think that other project would have been as interesting as this one. Last, but not least, I would like to thank my family and friends for putting up with me, and for always being there.

Lund, June2023
Bo Elovson Grey

Table of contents

List of acronyms and abbreviations 9

1. Introduction 10
   1.1 Background 10
      1.1.1 Furuboda folkhögskola 10
      1.1.2 Eldorado resurscenter 11
      1.1.3 Certec 11
      1.1.4 An outsider's view of an Eldorado musical session 12
   1.2 Purpose 12
   1.3 Goal statement 13
   1.4 Problem statement 14
   1.5 Motivations behind the thesis work 15
   1.6 Boundaries and limitations 16

2. Technical background 17
   2.1 The hardware 17
      2.1.1 Jetson Nano developer kit 18
      2.1.2 MIPI CSI-2 camera 21
   2.2 Max 8 – the music software 21
      2.2.1 OSC protocol 22
      2.2.2 MISK patcher 22
   2.3 Machine Learning and Computer Vision 26
      2.3.1 Neural Network 29
      2.3.2 Convolutional and Residual Convolutional Neural Networks 31
      2.3.3 Framework and library 34
      2.3.4 What happens during pose estimation? 35

3. Method 38
   3.1 Doing research 38
      3.1.1 Finding the right computer vision and machine learning tools 39
      3.1.2 Workshops 40
   3.2 Python 42
   3.3 Developing MISK-Moves 43

3.3.1 Software design and development 44
3.3.2 Testing 45
3.3.3 Figuring out PoseNet and ObjectPose 46
3.4 Max 8 test patcher 46
3.5 Source critique 47
4. Analysis 49
4.1 Takeaways from the workshops 49
4.1.1 The Eldorado workshop 49
4.1.2 The first session in the Certec lab 50
4.1.3 The second session in the Certec lab 50
4.2 Hardware selection 51
4.3 Installing things on the Jetson Nano 52
4.3.1 Docker containers 53
4.4 Settling for Jetson Inference 53
4.4.1 A quick pose estimation test using YOLO 55
4.4.2 A reason why MediaPipe pose is not used 55
5. Results 57
5.1 A quick overview of MISK-Moves's features 57
5.2 Selection in the MISK patcher 57
5.3 Output from MISK-Moves. 59
5.3.1 Person, limb and keypoint message contents 60
5.3.2 Distance measurement messages contents 62
5.4 Available settings. 63
5.5 Test patcher 64
5.6 I/O classes 64
5.7 Things provided by the Jetson Inference classes 65
6. Discussion 67
6.1 User testing 67
6.2 Latency and precision 68
6.3 Possible software performance improvements 69
6.4 Possible hardware improvements 70
6.5 Recommended usage for MISK-Moves 71
6.6 Ethical aspects 72
7. Conclusions 73
7.1 Conclusions and summaries 73
References 76
A.Appendix 78
A.1 UML diagram 78

List of acronyms and abbreviations

ANN	artificial neural network
COCO	Common Objects in Context – a dataset used to train and benchmark computer vision network models
CPU	central processing unit
CSI	Camera Serial Interface
DNN	deep neural network
fps	frames per second
GPU	graphics processing unit
MIDI	Musical Instrument Digital Interface – the original communication protocol for communication between music devices such as synthesizers, drum machines, effect modules, computers, tablets, etc.
MIPI	Mobile Industry Processor Interface
MISK	Musik, Interaktiv design, Sinnesstimulering och Kvalitet, or translated into English; Music, Interactive design, Stimulating the senses and Quality
ML	machine learning
ms	milliseconds
ONNX	Open Neural Network Exchange
OSC	OpenSoundControl – a communication protocol for sound and music devices, and can be seen as a complement or replacement to MIDI.
ReLU	rectified linear units
RGB	red green blue
SOM	system-on-module
synth	synthesizer
UML	Universal Modelling Language. A commonly used method for modelling software.
USB	Universal Serial Bus
TPU	tensor processing unit
YOLO	You Only Look Once – a family of network models for object detection and pose estimation

1 Introduction

The introduction chapter contains some background information on the project. The background, goal, motivation for, problem statements and scope are presented, as well as some information about the stakeholders in the project.

1.1 Background

This thesis work is part of the MISK project. It is short for Musik, Interaktiv design, Sinnesstimulering och Kvalitet, or translated into English; Music, Interactive design, Stimulating the senses and Quality [1]. The project is a collaboration between Eldorado resource centre in Gothenburg, Furuboda folkhögskola and Certec, the rehabilitation engineering and design branch of the Department of Design Sciences at Lund University's Faculty of Engineering. Allmäna Arvsfonden is financing the project, which has a time frame of three years. The main reason and motivation for this project is to make a new musical instrument for the MISK project.

The goal of the MISK project is to make multi-sensory musical instruments for people with severe communicative, cognitive, and physical disabilities. These instruments should help enable ways for musical expression, interactivity, engagement, and communication, even for those who are not normally able to play traditional instruments or sing. The instruments are meant to be used in musical therapy sessions.

1.1.1 Furuboda folkhögskola

Furuboda folkhögskola is part of Föreningen Furuboda, a non-profit organisation. In the beginning the organisation provided activities, and shortly thereafter education, for people with disabilities. Since the official start in 1959 they have expanded their operation to include the boarding school Furuboda folkhögskola, and education centres in Kristianstad and Malmö. Föreningen Furuboda provides language education and help to get established in the Swedish society for newly arrived immigrants, education for people that never got or failed to get a Swedish high school diploma, practical trade craft educations, several musical educations, personal assistant education, and academic research. To get the full width of what they do, a visit to their homepage is recommended.

1.1.2 Eldorado resurscenter

Eldorado resurcenter is part of the services that the city of Gothenburg provides. The centre provides cultural, musical, and other leisure activities for people with cognitive impairments, and their assistants and family, where the importance of social community is an important part. Among the activities provided are music cafés, musical sessions and activities in their sensory stimulation rooms, each room with a different theme. The themes vary from relaxing to engaging. There are both open activity days and the possibility to book individual sessions with specialised instructors, for people with severe impairments. The centre also provides education and lectures for people that work with, or in other parts of their daily life regularly deal with people with disabilities [12].

1.1.3 Certec

Certec, Rehabilitation Engineering and Design, is part of the Department of Design Sciences at Lund University's Faculty of Engineering. In addition to research, Certec provides a wide variety of different courses within all areas of. The courses range from theoretical to practical hand-on projects where students get to build working prototypes and devices.

In most, if not all projects, Certec works with the principles of universal design. Here is a short version of what they are:

- The design should be useful to people with a diverse set of abilities.
- Usage of the design should be flexible, so that it accommodates a wide range of individual abilities and preferences.
- It should be easy to understand how to use the design, regardless of each user's levels of experience, knowledge, language skills or ability to concentrate on the task at hand.
- Regardless of the user's sensory abilities, the design should communicate necessary information in an effective manner.
- There should be a minimal risk of any hazardous and adverse consequences if a user makes any accidental or unintended actions.
- A user should not have to use much physical effort to use the design, and its usage should be effective and comfortable and not introduce unnecessary fatigue.
- The design itself, and its functions should be appropriately sized. A user should be able to approach and reach all necessary controls and functions,

and at the same time have enough space to be able to effectively manipulate and use the design's functionality regardless of body size, posture, or mobility.

This fits well into what this project tries to accomplish. Although, in this case principles 4 and 6 might not as pronounced. In some cases, a Certec project sets out to give a practical solution for a problem of an individual, or a specific group of people. Then this solution can hopefully be generalised, transferred, and applied to a wider variety of problems, or to fit a larger group of people. Adapting a design for an individual to become a more universal design. This has been Certec's approach since the start in 1988 [18].

It was Certec that provided this thesis work. The department provided the initial idea for what went on to become the MISK-Moves application and the resources needed for this project. Everything from hardware and support to the facilities to be able to work on and do some preliminary testing on the MISK-Moves application.

1.1.4 An outsider's view of an Eldorado musical session

To give some context for when and how the musical instruments created under the MISK project are supposed to be used, a description of a music session is in order. From observing a couple of sessions on Eldorado, this is an observer's interpretation of how it works: A specialist music pedagog sits down together with one or more people with fairly to very severe cognitive impairments, and their personal assistants. Usually, the pedagog plays the guitar and sings, while the participants are given different instruments that they are encouraged to use during the songs. During the songs, the pedagog often calls out for one of the participants to respond, and then waits for the response before continuing the song. The instruments given to the participants are usually different rhythm instruments, large buttons that can play different sampled sounds, or tactile noise making squeeze toys. Examples of other instruments used in Eldorado's daily activities [12] are push or squeezable inflatable cones, a trampoline like area and a pushable cloth wall.

One of the driving ideas behind the MISK project is to have good sounding instruments that feel responsive. Someone playing an instrument should feel in control and be able to get that they are the one actually making the sounds, i.e., contributing to the music.

1.2 Purpose

The initial purpose of this thesis work is to make an application where a user can use their own body as a musical instrument, by moving their limbs, or moving around the room. The application should interpret the movements and use them to send

control signals to the music program Max 8. At the time of writing this, examples of things that can be controlled in Max 8 by the currently existing version of the MISK patcher are note pitch, volume, length, different sound filters, among several other things. This works for both a built-in sample player and a software synthesizer in the MISK patcher. There is also the possibility to change the quality of the sounds coming out of the patcher, such as changing different effects like reverb, gate, chorus effects etc., and changing filters and oscillator parameters in the software synth. The application should be able to send a variety of values, making it possible to control several things at the same time with the body. The application should also allow for the interaction between two people to control the MISK patcher.

At the start of the project there were already several existing MISK instruments. There is a pressure sensitive pillow, which lights up when pushed or squeezed. Another instrument that gives visual feedback is a pressure sensitive projector screen from which the MISK patcher both receives control messages from and sends visualizations to. The visualizations can be multi-coloured stars, clouds, patterns, among other things. There is also a tactile and pressure sensitive floor mat. An instrument similar to MISK-Moves is the FaceAR phone or tablet app, which is played by grimaces and face movements. MISK-Moves should provide new and different ways of musical interaction than the other MISK instruments. This is to give the potential users as many options as possible, increasing the chances to find at least one instrument that will work for each individual that visits them.

During the music sessions, the specialist music pedagogue should be able to pick and choose which limbs should control the music, or interactions that suit the impaired person the best. How much a person can move their limbs, how mobile they are, and how much motor control they have varies between each individual. That is why it is important to have several options for how to control the Max patch.

After the first workshop at Eldorado, it became clear that a good feature to have was the ability to enable two people to play the instrument together. Otherwise, it would just do about the same things as another already existing MISK instrument, the FaceAR app, but in a less convenient way. The purpose then shifted from just allowing one person to play, to enabling collaboration between several people.

1.3 Goal statement

The goal of this thesis work is to have a fully working application, which is able to send control messages to Max, and preferably receive messages as well. It would be good if the application is easy to start and manage, since the main users are music pedagogues, not engineers. The application shall allow a person to control things in the Max 8 MISK patcher by moving their left and/or right arm, their head, or their

entire body. Moving around in the room, or interaction between two people should also be possible ways, or input modes, to control the MISK patcher. As mentioned earlier, the ability to have two or more people to play the instrument in collaboration was added later in the project. The bodies of the users should be the instrument.

Making the source and the bill of materials available to others is important, as this might be interesting for other habitation resource centres to recreate this project. The hardware used should be inexpensive, or at least affordable, given that (re)habilitation centres and institutions like that often have strict budgets and limited resources.

To be able to use the body as an instrument, another important goal of the project is to use computer vision and machine learning (ML) to detect a person and their movements. To be able to run machine learning algorithms and models there are some options: Either run the application on a personal computer (either stationary or laptop; The operating system does not matter), a smartphone or tablet, or use some kind of system-on-module (SOM). For this project, the use of a system-on-module is explored.

1.4 Problem statement

To be able to solve the body as an instrument problem some questions have to be answered, and some problems need to be solved.

- Which system-on-module is the best fit for this project?
- What kind of machine learning models and algorithms are needed to be able to detect the movement of a person and/or their limbs? Also, can the same model and algorithm be used to handle more than one person?
- Will pre-trained models work or is there a need for re-training an existing model?
- Most potential users will be sitting down. Will there be a problem detecting someone sitting down, or if there is some part of the body that is obscured in some way?
- Will the application be able to track people and/or limbs in a reliable way, or will it lose track, make false detections or give large value fluctuations in some other way. Will this make it unplayable, or is there some way to smooth out the values if there are unreliable results?

- Will the instrument, i.e. the application, be responsive enough? For professional musicians, using professional music equipment, any amount of lag or latency is unacceptable. Does that apply to this application as well?

- How will the finished system be deployed?

- Will the application be able to detect and track more than one person, and will the performance be good enough when several people are tracked at the same time?

1.5 Motivations behind the thesis work

Allowing people with mild to severe cognitive, and sometimes physical impairments to be able to express themselves musically seems like a very worthwhile task. Making a musical instrument that is played by the free body or limb movements of one or several persons in a room, without having to hold or use an item is an interesting problem to solve. Making a usable prototype application seems like a good fit for an engineering thesis work.

A benefit to the MISK project is that there is a new instrument to try out and evaluate. Provided that the camera and system-on-module are well placed, this instrument allows fairly free movement, and no wires should be in the way or risk being pulled out or stumbled upon when playing it.

Certec gets some exploration of the system-on-module and its particular quirks and benefits out of the problem. Another benefit is that this thesis work allows for making an instrument there would not necessarily be time and resources to make otherwise, given that everyone involved is busy making their own instruments for the project.

The MISK-Moves application allows for collaboration in a way the other instruments do not really do. The collaboration can be between either the specialist music pedagog, and the main user, or between the personal assistant and the user. Some of the other instruments allow for some collaboration, but not in the same way, and not explicitly selectable in the Max 8 MISK patch in the same way.

The ultimate motivation behind this project is to provide a way, or means, to have fun with music. It should also enable someone who in most situations is unable or not allowed to have control over what happens to them or around them at least some control, by playing, triggering, or modifying sounds or music by themselves through the MISK-Moves application.

1.6 Boundaries and limitations

The scope of this thesis work is limited to the MISK-Moves application itself. This means that capturing a video feed, analysing it, and finally packaging and sending the results to a host computer running generating the sound are the limits for this project.

There are several existing, widely used ML-model implementations, and there is no need to develop a custom one for this project. It is also not really feasible to do any meaningful training or reinforcement training of an existing model. A large, good quality dataset would have to be prepared. Getting access to enough potential users, and then getting their consent to be part of a training set would be too time consuming, given that preferably the number of different people used should be in the hundreds, and that each person needs to be photographed from several different angles in different environments under different lighting conditions. As an example, the COCO (Common Objects in Context) dataset that the model in this project is trained on, contains more than 200000 images with 250000 person instances in them, according to the COCO Consortium themselves [4].

In the MISK project, the software that makes the different devices and instruments sound is Max 8 by Cycling '74. Developing the patcher that takes the input from all the different instruments and turns it into sound is another part of the larger project, and outside this thesis work.

Although this project is labelled as an interaction design project, the interaction design itself is very simple. All the user interface properties, behaviour, design, and interaction in the MISK patcher belongs to another part of the MISK project.

- The user plays the instrument by moving their arms, their head or by moving around in the room.
- Multiple users can play at the same time.
- Try to keep the input latency as low as possible, to make the instrument as responsive as possible.
- Allow for as many methods of control as possible. Not just absolute positioning, but movement speed (intensity), angle, direction, etc.

Emphasis during development and testing has been on ensuring that the application delivers expected and consistent behaviour that matches what is specified in the documentation and keeping the input latency as low as possible.

2 Technical background

In this chapter the technology and concepts used in this thesis work will be described, both the hardware and the machine learning models and concepts. There will also be an explanation of what computer vision is and what some of the differences between the different Machine learning models are.

2.1 The hardware

It was decided even before the thesis work started that the software should run on a system-on-module. There are a couple of good reasons for choosing a SOM over an ordinary computer. For projects like this one they are:

1. The device itself is small, which should allow for plenty of placement and housing options. It does not need much space, and can be put in a small box, the shelf of a bookcase, on top of a cupboard or up in the ceiling, just to name a few possible placements. It does not take up much storage space either.

2. A system-on-module does not need much power. Typically, these kinds of systems draw somewhere between 5 Watts and 60 Watts (the Jetson Orin AGX). A Jetson Nano, which is the SOM used in this project uses 5–10 W (low–high power mode) [15]. The power draw from ordinary computers is usually larger. A MacBook Pro 14 2023 M2 Pro draws 50–60 W during heavy load, when connected to an external screen [27]. Power consumption including screen should be higher. A desktop computer can use over 1 kW of power. A top-of-the-line consumer GPU, NVIDIA RTX 4090 [23], alone can pull 450–600 W (600 W is the maximum allowed power draw when overclocked), and it is possible to have several installed in a desktop computer. That is in a completely different performance class, though.

3. A system-on-module can be fairly cheap. At the time of writing the NVIDIA Jetson Nano 4GB development kit is $149, and a Raspberry Pi 4 is $149 – $300, depending on memory size and whether it is a complete kit with SD cards, casing, power supplies and other extras, or just the module itself. A Google Coral USB Accelerator that can be used with a Raspberry Pi is about

$145 on Amazon. A complete standalone Google Coral development board is $175 at the same store.

4. Given the price of the modules, especially the cheaper ones, the ratio value–performance is good for a system-on-module. All the systems mentioned can use some kind of hardware acceleration when running machine learning workloads. The Jetson Nano can utilize its GPU, and the Coral devices, either standalone or connected to a Raspberry Pi, has a highly specialized TPU (tensor processing unit) which only purpose is to run machine learning tasks.

If other makers, researchers, and activity centres, among others, want to recreate this project, the total cost of the device and its accessories should be as low as possible, to make it affordable for everyone.

2.1.1 Jetson Nano developer kit

NVIDIA introduced the Jetson Nano developer kit system-on-module in 2019. It is marketed towards embedded designers, researchers, and DIY makers. The price at the introduction was $99, but in March 2023 the typical retail price is $149 at retailers such as Amazon, Seeed, Arrow and Sparkfun Electronics. The system was powerful enough to run the latest ML-models at the time [15], and it can still run some of the very latest models at the time of writing this thesis in March 2023. As an example, it can run YOLOv8 models if the user is willing and able to customise the software environment on the Jetson Nano [26]. YOLOv8 was introduced on January 10th 2023.

The system itself is a complete tiny computer. Its system specifications are found in table 2.1. It comes with four USB ports, an SD card slot for the operating system and applications, a CSI connector for connecting a MIPI CSI-2 camera, a gigabit Ethernet port for networking and both HDMI and DisplayPort for display output. Connect a screen, mouse and keyboard and the system works like any desktop computer or laptop, running Ubuntu 18.04. The user gets a complete graphical desktop environment, and the possibility to install applications just like on any other personal computer. This means that software development can be done without any compromises on the device itself, since it provides a complete Ubuntu desktop environment. As an example, in this project Microsoft Visual Studio Code with a full set of Git(hub) and Python plugins was installed and used. The fact is that this entire thesis work could have been done on the Jetson Nano, research, development, writing the thesis, drawing figures, editing photos and all. The combined operating system and base development package is called Jetpack.

The system has other expansion ports, but they are not useful in this project. As a combined CPU/GPU it has an NVIDIA Tegra X1 chip, it is related to the system

Table 2.1 Jetson Nano Developer Kit 4GB revision B technical specifications.

Processing	
CPU	ARM A57 @ 1.43GHz, four 64-bit cores
GPU	128-core @ 921MHz, NVIDIA Maxwell architecture
Memory	4GB 64-bit LPDDR4 @ 1600MHz, 25.6 GB/s transfer rate
Video Encoder	3840x2160p30, up to 2x 1920x1080p60, and up to 4x 1920x1080p30 streams
Video Decoder	3840x2160p60, up to 2x 3840x2160p30, up to 4x 1920x1080p60, and up to 8x 1920x1080p30 streams.
Interfaces	
USB	4x USB 3.0 A (Host), and USB 2.0 Micro B (Device)
Camera	MIPI CSI-2 (15-position Flex Connector
Display	HDMI and DisplayPort
Networking	Gigabit Ethernet (RJ45)
Wireless	M.2 Key-E with PCIe x1
Storage	MicroSD card (16GB UHS-1 recommended minimum size)
Other I/O	3x I2C, 2x SPI, UART, I2s, and GPIOs

chip that the popular games console Nintendo Switch has, but with different clock speeds and half the number of GPU cores [19].

One of the advantages of the Jetson Nano is the large amount of supported software, and the wide variety of tasks that can be performed on a Jetson device, as hinted by the introduction announcement [15]. Several popular machine learning networks such as PyTorch, TensorFlow, Caffe/Caffe2, Keras and MXNet are available with full native hardware acceleration on the Jetson Nano. Frameworks like the open ONNX can be converted into NVIDIA's own TensorRT framework, to take full advantage of the device. This means that image recognition, object detection and localization/tracking, pose estimation, semantic segmentation, video enhancement and ML-based analytics can be done on a small low power device. Jetpack comes with several libraries that are modified and enhanced by NVIDIA to work optimally on the Jetson:

- TensorRT and cuDNN for hardware accelerated deep learning applications.
- CUDA, NVIDIA's GPU accelerated general compute package.
- NVIDIA Container Runtime for running Docker containers. This allows for the Jetson to run fully hardware accelerated Docker containers.

Figure 2.1 The Jetson Nano developer kit connected to a Raspberry Pi V2 camera mounted to a tripod. A pen and a micro SD card are included for size comparison.

- Multimedia API, which allows the use of several different cameras, and several different kinds of input stream encoding and decoding.

The libraries come with a set of samples and example code. Then there are several tools and application for specifically debugging software written for the NVIDIA libraries, and some profiling and optimization tools, to help developers run their code as efficiently as possible on Jetson devices.

Being able to run Docker containers with full access to the hardware is a useful feature. A Docker container is a complete runtime environment. This means that instead of having to install all the libraries and dependencies needed for a particular piece of software, a complete Docker image can be downloaded and run as a container instead [11]. Especially when distributing the same software to several computers or SOMs, this saves a lot of time and effort. It works similar to a virtual machine, but uses less resources.

Competing products can sometimes have better performance in specific tasks, as

seen in the material [15], but none of the competing products at the time provided the same versatility, software, and tool support. Jetson devices can be found in robots, security systems, cars and more.

2.1.2 MIPI CSI-2 camera

MIPI (Mobile Industry Processor Interface) CSI-2 (Camera Serial Interface) is a specification for processor – mobile camera communication, a high-speed camera and imaging interface. Devices complying to the protocol feature high performance, low power consumption, and have low electromagnetic interference. The interface is typically used in high-performance applications such as high-resolution photography, and 1080p, 4k and 8k video. MIPI CSI-2 cameras are used for virtual reality, drones, the Internet of Things, medical devices, tablets, notebook, industrial systems and more [20].

The CSI-2 interface on the Jetson Nano has driver support for the Sony IMX219 sensor [15]. A camera module that uses this sensor is the Raspberry Pi Camera Module 2, which is the module used in this project. The sensor is backlit with a resolution of 8 megapixels [28]. At the time of writing, price listings on amazon for the camera module varied between $28 and $35. The camera module is the small circuit board with a lens on it mounted to the tripod in figure 2.1.

In this project the raw image feed from the camera is used. There is the possibility to fine-tune the camera output, such as setting white balance, using different methods of noise reduction, lens distortion correction, changing brightness and saturation among other things, to get an image quality that suits the user's subjective preferences. However, the raw image feed worked well enough for the purpose of pose estimation.

2.2 Max 8 – the music software

Max is a visual programming language for music and multimedia applications, and Max 8 is the name of the current software package, i.e. the development and runtime tool for applications using the Max language. It is made by Cycling '74 [5]. The software is available for Windows and MacOS.

Programming in Max is done by placing different objects. Objects can be sound generators, filters, arithmetic or logical functions, network communication objects, visualizers, video decoders, to name only a few examples. User interface elements such as buttons, sliders, knobs, and displays are also objects. These objects are then connected graphically with virtual wires. Most objects have several inputs and outputs that let them process things in several different ways, depending on how the

programmer connects the different objects to each other. A Max program is called a patcher, and a patcher can also become an object. If the set of objects that Max 8 provides is not enough, there is the possibility to use plugins, either pre-made made by other developers or by the user themselves. Examples of what the MISK patcher looks like are found in figure 2.2, 2.4 and 3.1, among others.

2.2.1 OSC protocol

OSC is short for Open Sound Control. It is meant to be a more flexible alternative MIDI, a data transport specification for real-time communication among applications and hardware. The protocol allows full user customization of the address space and the parameters that get sent. Unlike MIDI, OSC does not need a specialized interface, but uses standard networking hardware (e.g., Wi-Fi, Bluetooth or Ethernet). Since it is meant for real-time music performances, it is designed to be lightweight, low latency and highly accurate [32]. MIDI is the Musical Instrument Digital Interface standard.

If something does not support MIDI, Max 8 uses the OSC protocol to communicate with external devices and computers, or with other applications running on the same computer as Max 8. This is what all the instruments in the MISK project use to send and receive control messages to and from the MISK patcher. The OSC messages themselves are technically UDP packets, and Max 8 has pre-made objects for sending and receiving such packets. They are called `udpsend` and `udpreceive` respectively.

The MISK-Moves application uses the Python-osc library for its OSC communication. The library provides simple client and server python classes for easy information packaging, sending, and receiving. The library follows the OpenSoundControl Specification 1.0, and is unlicensed and can be used freely without any restrictions [3].

2.2.2 MISK patcher

The MISK patcher for Max 8 is developed as another part of the MISK project. This is the musical hub of the project. All the different instruments connect to the patcher and can be used to control different parts of it. This is what makes the actual sounds and music, except for the specialist music pedagog singing and playing their own instrument. The main screen that greets the user at start-up is shown in figure 2.2. There is a variety of functionality built into the patcher. To generate sound, it has a sample player, a synth, and a regular music player. There is also a sequencer that can be used to write and/or record melodies and other musical sequences, so that they can be replayed. In addition to sound and music, the patcher can also generate

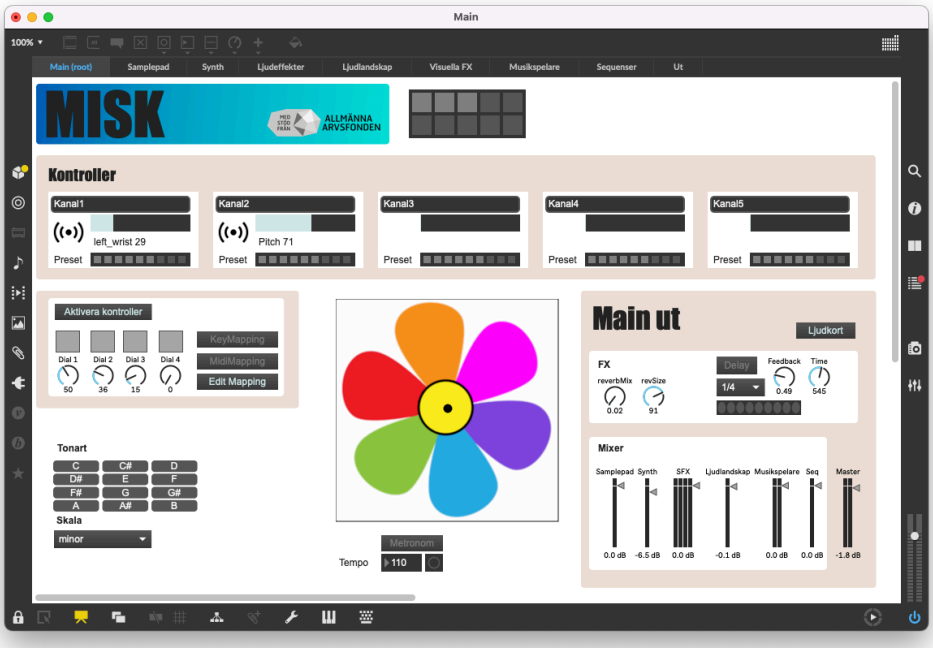

Figure 2.2 **The main tab of the MISK patcher. Two channels are active. Channel 1 is controlled by the left wrist and channel 2 is controlled by the distance between the playing person's arms.**

various visual effects on the host computer's screen, or on a projector connected to the computer.

The sample player is used for a sample pad with 12 pads, where one sample can be connected to each pad. This allows for either quick sample selection, or for playing a sequence of samples. Another possibility is to just trigger the pads randomly. The MISK instruments can either trigger a sample in general, or have it trigger a specific sample pad. Other things that can be controlled by the instruments are the pitch and volume of the sample currently playing. The sample pad is shown in figure 2.3.

The synth, see figure 2.4 consists of three different selectable synthesizers: one with only the basic waveforms, seen in figure 2.5, one with more advanced sound generation, as seen in figure 2.6, and the last is a sampler synth, in figure 2.7.

Here the MISK instruments can control the note played, the chord played, or control some of the synthesizer parameters such as filters and waveforms.

The sequencer can be seen in figure2.8. Here the MISK instruments can control the volume of the drums and the different melodies, switch pre-set melodies, change the

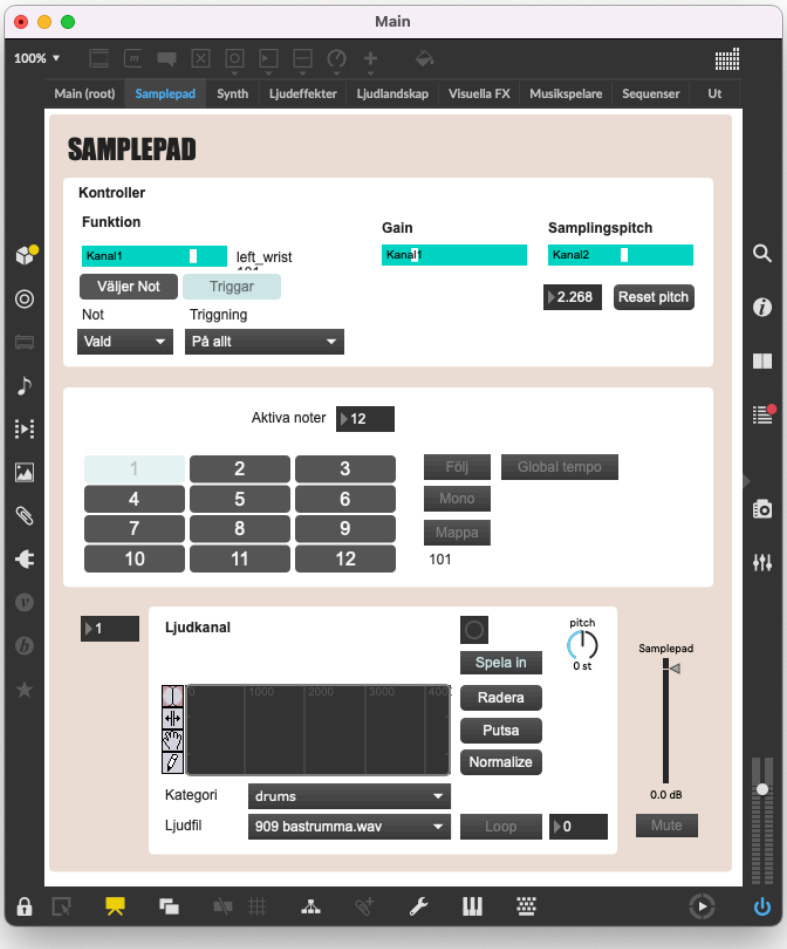

Figure 2.3 The sample pad. Here channel 1 is used to control stepping through the samples and control the volume, while channel 2 controls the sample pitch.

complexity of the generated melodies and the dynamic range of the notes played. Even though the melodies are seemingly randomly generated, they still follow harmony theory. This means that melody and base line are in the same key, and that both follow the same chord progression.

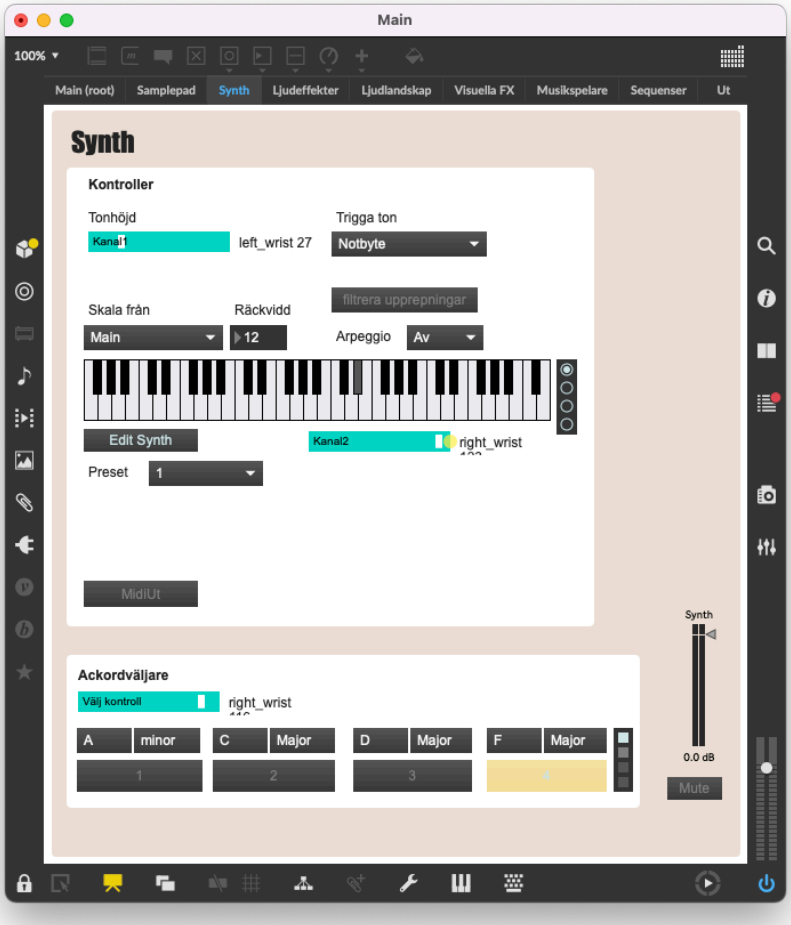

Figure 2.4 The main synth screen, using the left wrist to play the synth and the right wrist to control selected parameters.

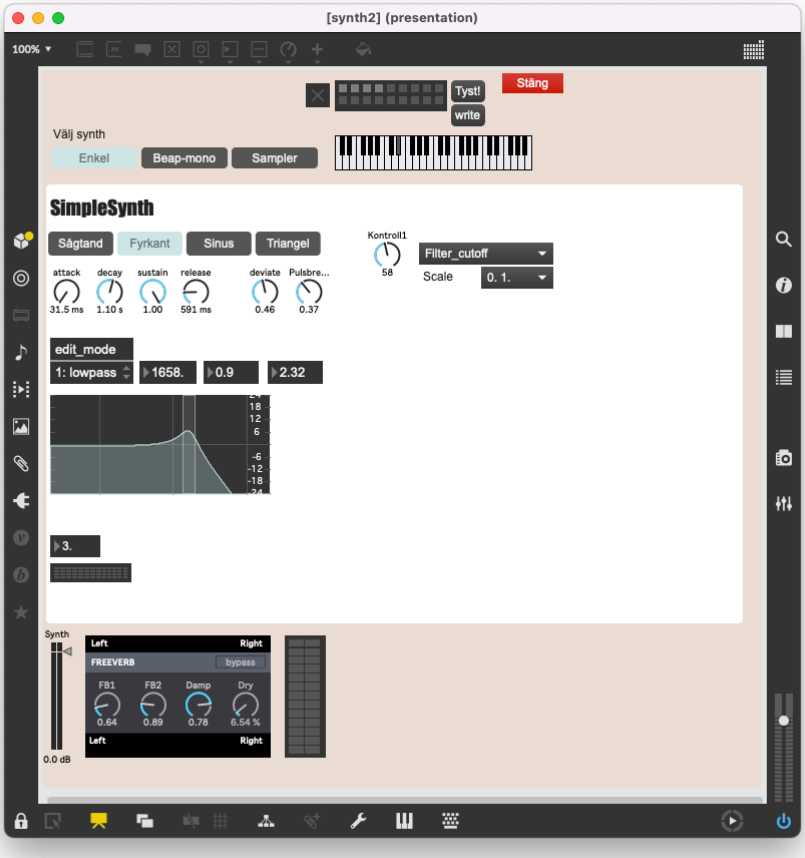

Figure 2.5 The simple synth. Here the filter cut-off is controlled by MISK-Moves.

2.3 Machine Learning and Computer Vision

Machine learning, ML, is a form of artificial intelligence, AI, where a computer model learns to solve a specific problem (or specific set of problems) without the solution being specifically programmed [6]. The model is fed with data, which is used in several training steps, often called epochs, to make the model improve the accuracy of its inferences, getting it as close as possible to reality. The problem, or task, needs to be well defined within clearly defined boundaries. These criteria make machine learning fall into the narrow AI category, as opposed to broad AI, which is a theoretical model for solving *any* type of problems. Proper broad AI does not exist yet, but machine learning models grow ever more popular due to their ability to quickly analyse large amounts of data.

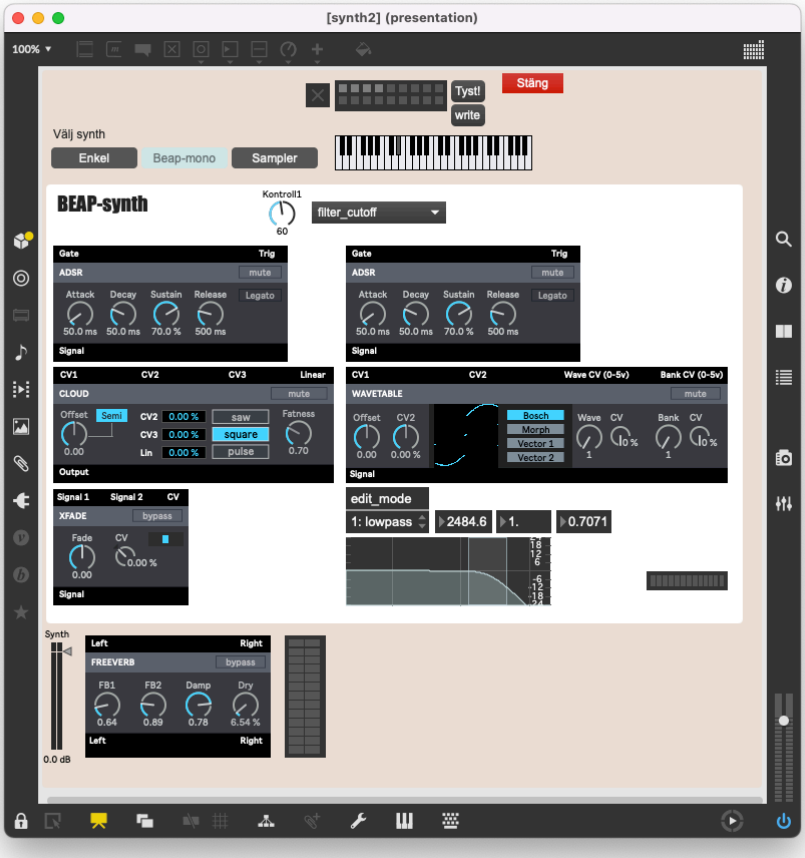

Figure 2.6 The BEAP synth. Again, the filter cut-off is controlled by MISK-Moves.

Deep learning is a sub-category of machine learning [10]. The idea is to have the deep learning algorithms and networks learn without any greater human involvement. This means that there should be less need for marking up or annotating the training data. Deep learning algorithms are well suited for unstructured data. The algorithms find similarities and reoccurring patterns during training and usually categorizes what they find in differently numbered categories. This means that with high quality training data a deep learning algorithm can teach itself to find objects in pictures and videos, learn to distinguish the different words spoken in audio input, or learn to detect suspicious or fraudulent behaviour in bank transactions, just to give a few examples. If there is a need for it, the numbered categories that the algorithms output can be translated into proper noun categories, if a human wants to easily understand what an algorithm has concluded. The deep in deep learning

Figure 2.7 The sampler synth. Here VCA bypass is controlled by MISK-Moves.

refers to the layer structure. Usually, deep neural networks have a far larger number of layers than ordinary machine learning neural networks, thus being deeper. Deep learning models are usually very hard to interpret, given the usually large number of layers and nodes and the autonomous unsupervised training. This makes it hard to understand why and how a network manages to draw accurate conclusions.

Most of these models and engines are open source. The advantage of that is that if so inclined, everyone can see *exactly* how a model works by reading the source code. Several, if not most of the models have their own GitHub page with a complete set of resources. This means that anyone can download, test them out and modify them to fit their own use cases. Usually there are instructions on how to train, retrain and/or do reinforcement training on the model, so it can be trained to suit a specific need. As an example, YOLOv7 and YOLOv8, two different YOLO models from different

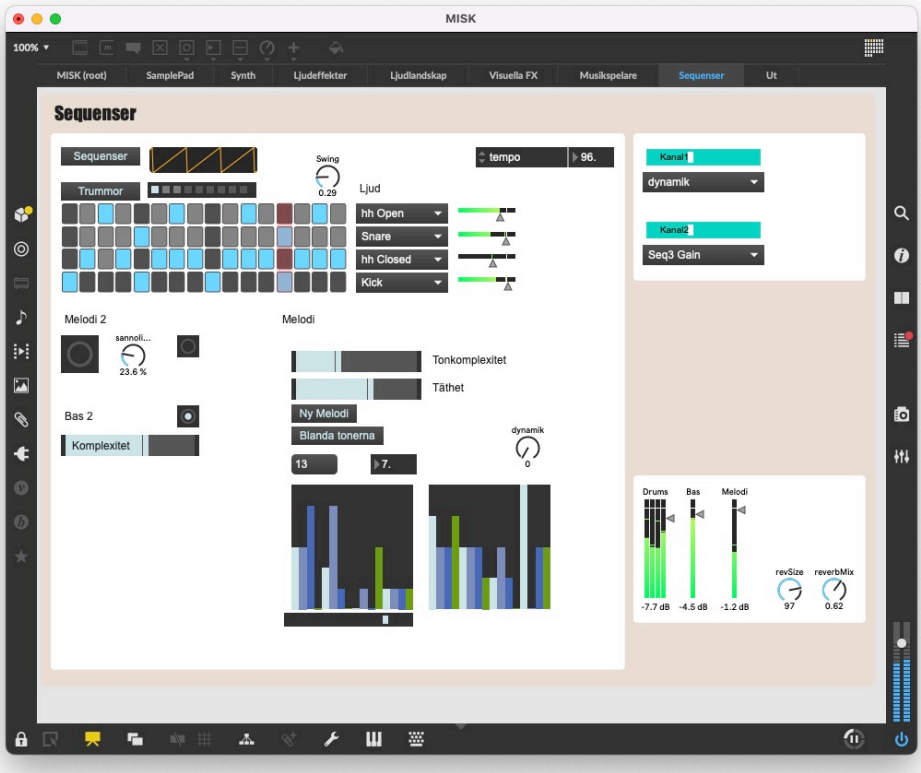

Figure 2.8 The dynamics of the notes played is controlled by channel 1 and channel 2 controls the volume of the third melody part.

authors and organizations can be found at [31] and [29] respectively.

2.3.1 Neural Network

Most machine learning is based on neural networks, or to use the correct term, Artificial Neural Networks, ANN. The inspiration for this way of processing data is the neurons in the human brain. A neural network has multiple layers of neurons. There is always an input layer, one or several hidden layers, and an output layer. The input layer receives and prepares the input data before it passes it along to the hidden layer(s). The output layer is responsible for the final decision based on the results from the previous hidden layers and delivers the result of what it has concluded. It is within the hidden layers that all the processing of the data happens. How many neural nodes each layer consists of and how many layers there are varies from model to model. When describing the size of a neural network the terms breadth, or width,

and depth are used. The width refers to how many nodes there are for each layer and the depth refers to how many layers the network consists of. A simple illustration of an artificial neural network is found in figure 2.9.

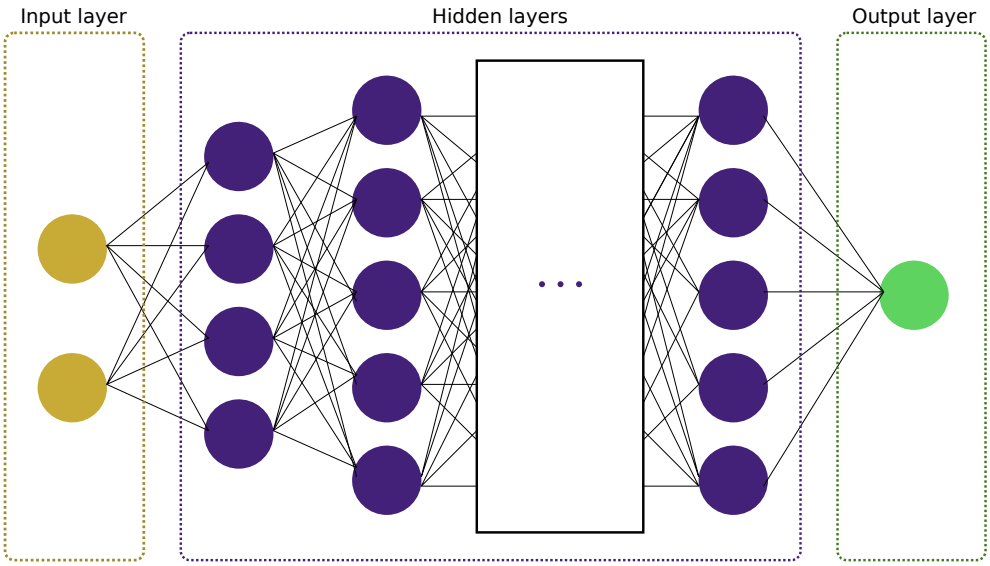

Figure 2.9 A simple illustration of an artificial neural network. The white box in the hidden layers represents several more hidden layers.

During training data is sent through the network and the result is then compared to some actual result from the data set. The comparison shows how accurate the conclusions from the network are, and the difference between the actual and concluded results are used as feedback to the network for the next training round, or epoch. This feedback allows for adjusting the importance of the input weights for all the inputs of each node in the network. This process is repeated until the accuracy of the conclusions from the network are accurate enough, and then the model is ready to use.

Deep neural networks, or DNN, are used in deep learning. DNN is just a subcategory of ANN. Deep refers to the fact that a DNN has multiple layers, and the number in for example ResNet-18 or ResNet-50 refers to the number of layers in those particular networks. Usually, a deep network takes longer to train, but the advantage is that inferencing using a model based on a deep network is both faster and has more accurate conclusions. This makes it possible to train the model on a powerful server, and then run the application on a far less powerful device, such as a mobile phone or a small embedded system-on-module.

What happens during learning, such as what weights are applied and if and how any

weights are adjusted during, or as a result of the processing, is usually not observable from the outside, especially when it comes to DNNs.

A network model describes how the neurons are connected, i.e. what their input and output is, how the input and output is weighted and sometimes added, what each layer of neurons consists of, how the layers are grouped, structured and interconnected, etc.

Data sent through the neural networks almost always comes in the form of tensors. A short and simplified explanations of what tensors are is that they can be seen as multi-dimensional matrices. As a relevant example to this project: In the case of computer vision and pose estimation, the image can be seen as a matrix with image height (in pixels) rows and image width columns. For colour images each pixel in the RGB (red, green, blue) format has one value for each colour channel. An input tensor for an RGB image would have one dimension for each, channel, i.e. three dimension. In turn, each dimension on their own would be an (*image height in pixels*)x(*image width in pixels*) matrix. Linear algebra is used to perform mathematical operations, such as multiplication, transposing, calculating the dot product, convolution, etc. [8]. The reason why this is an important property in this project is that the GPU in the Jetson Nano SOM is very well suited for performing the calculations needed. All 3D computer graphics are based on linear algebra, and GPUs are especially made and optimized for doing such calculations. GPUs are mainly optimized for matrix operations, but that still makes them better suited for the task than CPUs, which are more suited for general computing tasks. In the later generations of the Jetson SOMs, the hardware has a TPU, i.e. tensor processing unit optimized for tensor operations, as well as a GPU. This makes the new Jetsons even better at ML tasks [24].

2.3.2 Convolutional and Residual Convolutional Neural Networks

This project uses ResNet-18, which is a residual convolutional neural network with 18 layers. Hence ResNet-18. To give some insight how the pose estimation works this and the following sub-sections will describe ResNet, why such network models are used, and the steps of the pose estimation process.

The invention of Convolutional Neural Networks took inspiration from and are modelled after the human visual system. The input is split into several small sub-parts which are analysed individually, and in the end are put together to complete the inferencing on the whole input image. The breaking things up into smaller parts helps reduce the dimensions and complexity of each calculation [7].

The calculations of the smallest parts of the input are done in the convolution layers. The total performance and complexity of all the calculations depend on the size of each part, and if there is any space between the parts, and the size of that space.

After the convolution layers comes the pooling layer where neighbouring smaller parts get put into pools. Now depending on application, either the average of the parts in the pool or the maximum value of the parts is the result. This has the effect that the model can preserve small details of characteristics from the smaller parts in one larger part, or "pixel" if it is an image recognition application.

Finally, there is a fully connected layer that uses all the subparts and connect everything together for the final classification. Since it can use the subparts instead of each individual original pixel, the dimensions of the input are greatly reduced which means that it is possible to use tightly meshed layers (remember figure 2.9) without the complexity growing out of hand.

Here is where tensors come in. The data is described using tensors on which all the mathematical operations, such as convolution, are performed. This is the reason why the hardware acceleration in the Jetson family of som:s make them so well-suited for ML-tasks, especially those models that have dedicated tensor cores.

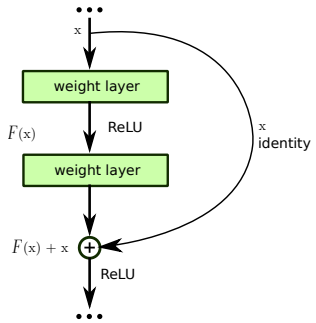

Figure 2.10 A building block for ResNet showing the building block layer structure of the network.

In theory, the deeper the network, it should be possible to achieve better results. The problem is that the training error grows the deeper the network is. Most training is dependent on backpropagation, but this causes any errors to travel through the entire network from the back to the front. During training calculations are made to find out how much each neuron contributes to the total error, and the method used is to calculate the gradient. What happens is that when traversing back to front, the neurons in the first layers will contribute very little to nothing to the final output when calculating the gradient, and that means that neurons that could actually be valuable risk being removed during training. This causes training errors.

To solve this problem residual networks were invented. The very short explanation is that that such networks approximate a residual function where shortcuts are used as identity mapping. Figure 2.10 illustrates how this works. For a more detailed explanation, see [17]. If a layer does not contribute anything to improving the final

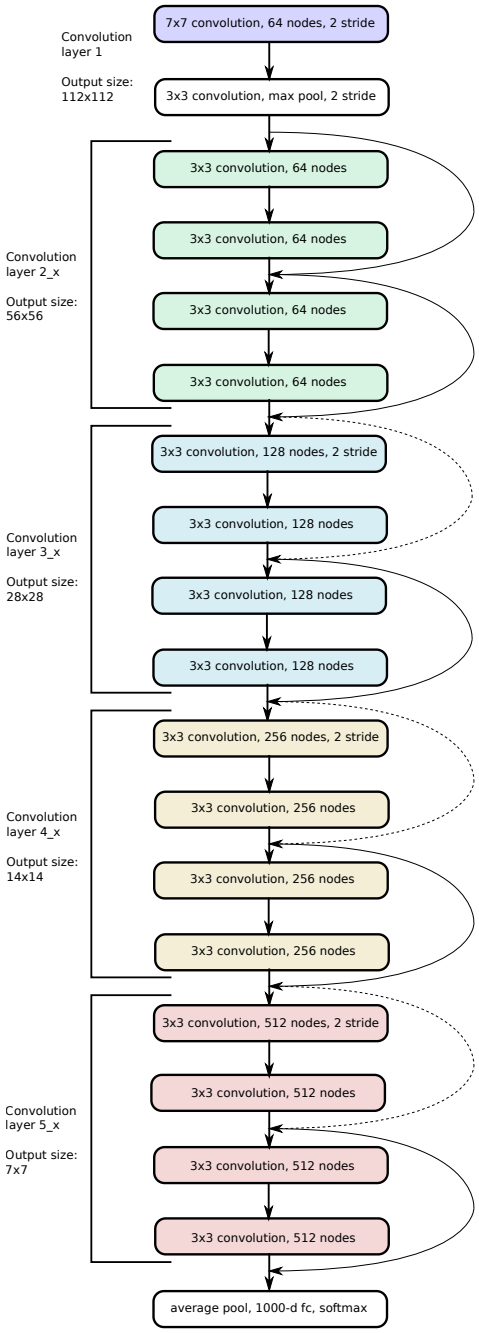

Figure 2.11 The complete structure of ResNet-18.

result it can be skipped. This means that the ResNet can have a large layer depth without having training errors from gradient degradation. Skipping layers that do not contribute anything also gives the ResNets the advantage of generally being less compute intensive, and therefore being able to be trained faster, and have shorter inference time on when running trained models.

A building block is shown in figure 2.10. ResNet-18 [17], which is used in this project, consists of several of these building blocks. The ReLU functions, rectified linear units [21], are the activation functions between each layer. They are used in the network to determine whether a neuron or a layer should activate or not based on the input. It seems like ResNet-18 uses a leaky ReLU which would look something like this mathematically:

$$f = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases}$$

What the entire network layer structure should look like is shown in figure 2.11

2.3.3 Framework and library

When implementing a neural network model there are several frameworks, or libraries, to choose from. These libraries provide tools for describing the neurons and how the neurons are interconnected. Other provided functionality includes mathematical operations (linear algebra, convolution, statistical functions, etc.), especially tensor operations, and often various graph traversal and building functionality. The libraries also provide tools for preparing, training, and processing data.

Some of commonly used frameworks are PyTorch, TensorFlow, Keras, and Caffe. All these frameworks are open source. Each framework has its own way to describe a neural network model, which means that they each have their own file formats, and their own way to classify and specify data and output. This for example means that a PyTorch model cannot directly be converted into a TensorFlow model. It is however possible to convert models between different frameworks taking an intermediary step, and something like Open Neural Network Exchange, ONNX for short, is often used as the intermediary model conversion. This middle step makes it possible to convert models between two otherwise incompatible frameworks.

For this project the original model was converted via ONNX to TensorRT, which is a proprietary framework by NVIDIA, and it is optimized for NVIDIA GPUs, just like the one in the Jetson Nano. When using the Jetson Inference package, this conversion happens automatically the first time when a model is used in one of the example programs.

2.3.4 What happens during pose estimation?

Now that all of the technology behind object detection and pose estimation have been explained, it is time to put it all together and put it to practical use.

What happens in MISK-Moves can be described using NVIDIA's description of how their DeepStream framework works is a really good description of how a machine learning computer vision pipeline works [25]. The entire process is illustrated in figure 2.12. Pose estimation is just a special case of object detection. The DeepStream

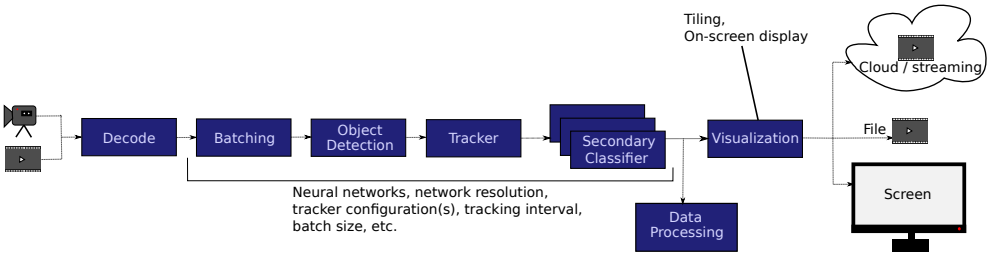

Figure 2.12 A typical DeepStream pipeline showing all the process stages from input to output.

workflow works like this: First there is some input. This can be an image, a video stream from either a camera or from a file, an audio stream, text, or some other data that needs to be analysed. A sample from the stream is grabbed. In the case of a video stream, a frame is grabbed. This sample can be modified in some way. In the video frame example, the image could need scaling, denoising, sharpening, have the colour format changed, or need some other kind of image manipulation. Then the sample is sent as input into an ml network. In DeepStream it is possible to link several networks together. To use the well-used car example, an example application tracks cars. The first Network is an object detector, which is set to detect cars in general. The output from that network is cropped so that only the part of the image that contains cars is sent to the next network. This network is trained to be able to tell of which make a car is. The cars also need to be tracked, so the output is also sent into an object tracker. After the inferencing chain, the results are processed, and any calculations and analysis are done at this stage. After the processing, the next stage is to present the results. With a video stream, the results can be combined with the original input and sent out for viewing. This can mean that the information is overlayed on top of the original frame, or that several frames are put side by side. All that happens in a continuous stream, which is then encoded and sent out as a new video stream for direct viewing, or saved to a file.

MISK-Moves does not use DeepStream, but the process is similar. It starts with capturing a frame from the MIPI CSI-2 camera. The image quality from the raw feed is enough, but on the Jetson Nano the image is scaled down to get some more per-

formance. The frame is then sent into poseNet. This is a pre-made class from the Jetson Inference library that NVIDIA provides. In this project, the class puts the data through the ResNet-18 pose model, which does the inferencing. PoseNet needs the model to be in either Caffe, UFF, or ONNX network model format, otherwise it will not be able to use it. If the model is run for the first time, PoseNet automatically converts the network model into the TensorRT format, to make sure it runs at optimum performance on the Jetson. The result from the model then gets analysed and post processed by PoseNet. The results from ResNet-18 comes in the form of keypoints, as defined by a topography specified by NVIDIA when the class was written. The complete topography, with all the keypoints named and numbered is shown in figure 2.13. Each keypoint comes with a confidence value, which shows how certain ResNet-18 is that it in fact is the actual limb (keypoint) that it has detected. If the confidence value from the model is below a pre-specified threshold, the post processing will ignore that keypoint, and not use it in the results. The same goes for links between keypoints. In the topography there are some links between certain pairs of keypoints defined. The links are shown as lines between keypoints in figure 2.13. If all the keypoints and links are found, the results will look like a complete skeleton when drawn. When the inferencing and post processing is done PoseNet returns the results as one ObjectPose struct per person it has found, containing all the data from the inferencing and processing. It also outputs an image where all the keypoints and links found are drawn as an overlay on top of the original input image. Each object contains the x and y coordinates of where in the frame a person is, and coordinates for all the keypoints, or limbs, it has found on each person. Each object also contains a list of any links between keypoints found. PoseNet can detect more than one person per frame. The processing part then consists of calculating how much each person and each keypoint has moved, and this in turn gets sent over to the MISK patcher running in Max 8 as OSC messages. The presentation consists of the original video input with an overlay of a skeleton of each person found. The skeletons consist of the keypoints and links drawn between the keypoints, as seen in figure 5.3. The output video can either be viewed directly on the Jetson's screen or remotely as a video stream. It can also save the video as a file.

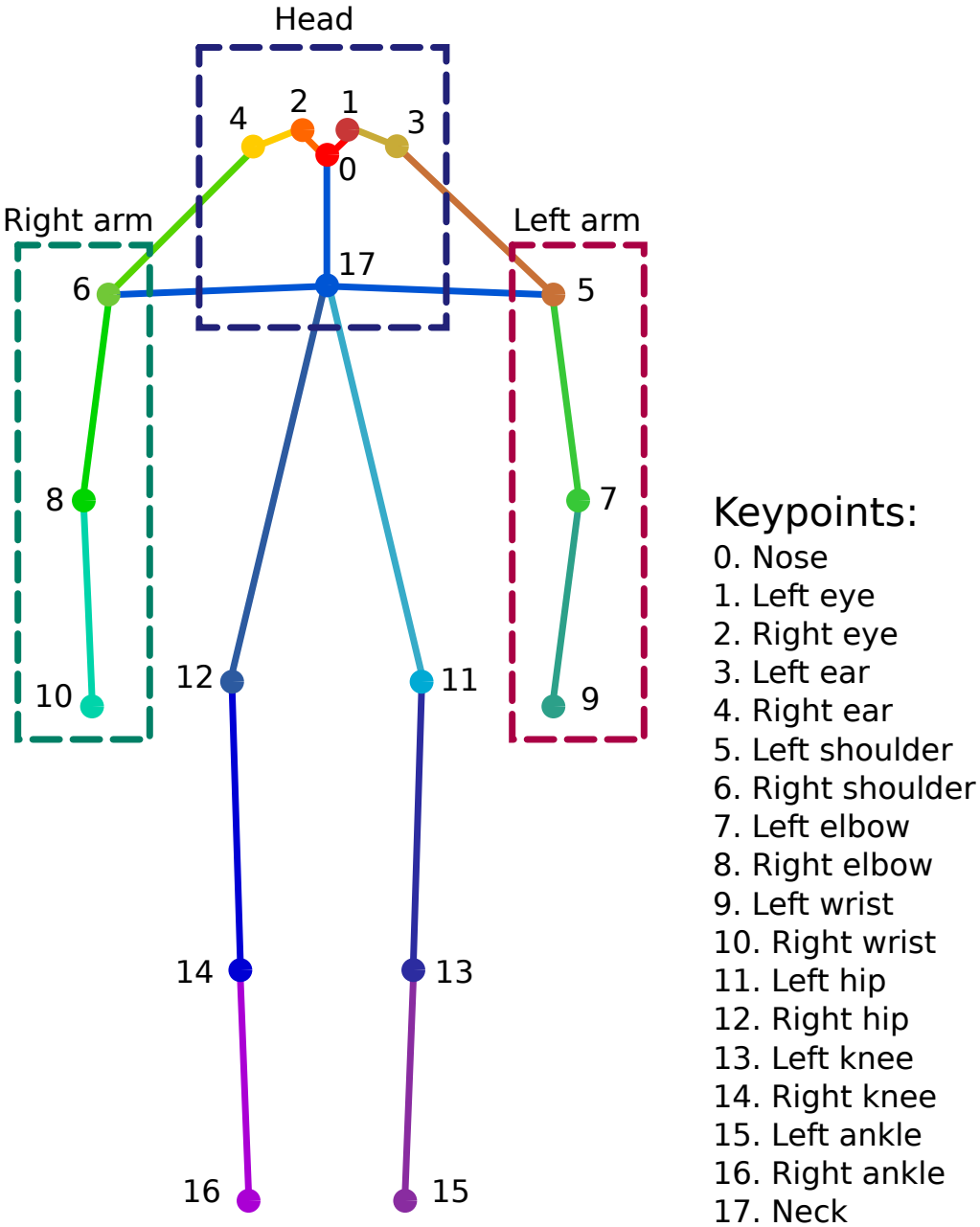

Figure 2.13 **The complete topography as defined in poseNet, showing all the keypoints and their names. Links are shown as lines between keypoints. Additionally, the groupings of keypoints that make up the head, left arm and right arm are shown.**

3 Method

In this chapter, the methods and development process for reaching the finished MISK-Moves application are discussed.

3.1 Doing research

The initial instructions for this project was to make an instrument which was played by moving around in a room. To achieve this, it was decided to use computer vision and machine learning.

The main source for finding information about anything in this project has been Google searches. It has been used for

- Finding information about similar musical instruments.
- Finding out what hardware to use.
- Finding suitable libraries, packages and other software development tools, ML-models/networks/engines.
- Finding documentation for the different packages and libraries used.
- Finding solutions to the different problems that arose during development.

Another valuable source of information has been various GitHub pages. As an example, NVIDIA's "Hello AI world" is hosted entirely on GitHub [14]. This including the entire Jetson Inference package, with example code, build scripts and pretrained models. Another example is that YOLOv7 is also hosted on GitHub. Documentation, example code, several pretrained models, benchmarks, benchmark results, academic paper, and all. Some GitHub pages also have useful links to other GitHub pages, or links to other useful resources.

NVIDIA's own home pages have been an important source of information. Their developer pages, which contain documentation about their software tools and hardware. Here anything from hardware and software installation instructions, to user manuals, to API specification and documentation, and other useful tips can be found. NVIDIA also has their own support forum where it is possible to find solutions to a variety of problems or ask for help if a solution does not currently exist.

3.1.1 Finding the right computer vision and machine learning tools

When searching around for computer vision tutorials, OpenCV tutorials are among the most common results. OpenCV, or Open Source Computer Vision Library is a popular open source computer vision and machine learning software library under the Apache 2 license. It contains algorithms for image enhancement and manipulation, producing overlays, handle video, and do some machine learning tasks. It is used by everything from hobbyists to the some of the largest corporations in the world such as Google, Microsoft, Intel, IBM, and Toyota. YOLOv7 [30] uses OpenCV in their implementation, to give another example. OpenCV has C++, Python, Java and Matlab interfaces and it can be installed on Windows, Linux, Android, and Mac OS. There is also support for NVIDIA's CUDA, which means that it is possible to have hardware accelerated OpenCV applications on the Jetson Nano. Doing some OpenCV tutorials seemed like a good starting point to get familiarized with computer vision, and to understand the concepts. YOLO is short for You Only Look Once.

Since the initial description was that a person should be able to play/control the application by moving around in the room, some object detection and tracking tutorials were researched and used. When searching for tutorials around the web and example code on GitHub, most of the source code is written in Python. To give examples from the bibliography: The Hello AI World video tutorials and YOLOv7 use Python. This led to some quick Python documentation studies and to doing some simple Python tutorials.

When searching for suitable object detection models and trackers, the models showing the most promise were later versions of YOLO, either YOLOv4 or YOLOv7. Looking at benchmark numbers, they provide a good combination of performance and accuracy.

A lot of time, at least a month and a half, was spent on trying to get various object detection models and machine learning frameworks working. There are hardware accelerated versions of PyTorch and TensorFlow available for the Jetson Nano, but at first, they would not install properly, or rather some of the dependencies for those frameworks would not install properly. Without the proper dependencies and libraries no example code, or implementations for any models or frameworks would run. In the best case scenarios, there is an error message, instead of a complete lockup. The messages can be hundreds of lines long, leading to that it can take tens of minutes to hours, to sort out what the root of the problem is.

A large part of the project, perhaps too much, went into searching around different support forums for solutions. Other projects with similar problems seemed to be helped by the solutions offered in the different discussion threads, but nothing would work for this project. The situation got so dire that even the Jetson Inference exam-

ples stopped working. Not only the machine learning examples, but simple video players and camera capture applications would stop working.

However, after the Eldorado workshop it became clear that the Room Camera application needed to use pose estimation. There are YOLO based pose estimators, but a lot of time had been spent on trying to get any ML-model and framework to work at all. In the end, all ideas of using the latest models were discarded. Instead the Jetson Inference class PoseNet using a ResNet-18 pose estimation model became the basis for the application, since that model and framework was more or less guaranteed to work.

When the basis for MISK-Moves was decided on, the Jetson Nano received a fresh Jetpack Install, and a Jetson Inference package install built from scratch locally on the module, and only the necessary development tools and libraries were installed. After that the development could progress without any problems.

After the application was mostly finished, the Jetson Nano received installations of PyTorch, TensorFlow and YOLOv7. A CUDA accelerated version of OpenCV 4.6 was also built and installed on the module. Everything works. The original Jetpack installation probably broke early during the exploration and research. Not fully understanding how Jetpack worked and some of the inner workings of Python in the beginning of the project is probably the root cause. Especially how particular it is regarding requirement, library, and dependency version compatibility. The various solutions when trying to fix the problems probably broke things even more. Another cause might have been a standard software update to Jetpack. Such updates also need to be handled with caution. Like Windows and MacOS Ubuntu, which Jetpack is based on, has the option to automatically install updates to the operating system and any software packages installed. Not all updates support the Jetson Nano hardware and are meant for another type of Jetson device. Automatically installing security updates and bug fixes is generally a good idea, but not when they risk breaking the system, so that it stops working properly.

3.1.2 Workshops

During the development of the MISK-Moves application three workshops were attended: one at Eldorado in Gothenburg, and two in Lund in the Certec lab. The Eldorado workshop was about three weeks into the project, while the workshops in the lab were at the end of the software development process.

At Eldorado there was the opportunity to observe the environment and the context in which the Room Camera application is to be used. The music sessions took part in two different rooms. In the largest room group sessions were held. In these sessions some of the sound effect patcher functionality was tested. One-on-one sessions were held in the smaller rooms, allowing for a more intimate setting with only the special-

ist music pedagog, the attendee and the attendee's personal assistant were present. In one of these rooms the MISK pillow and pressure sensitive mat were tested, and in the other the FaceAR app was tested. Since playing the pillow and mat involves physical activity and movement, the sessions involving those seemed the most interesting to observe. To not have an extra stranger, and risk disturbing the intimate mood or make the attendee uneasy, a camera was set up on top of a cupboard, and the sessions were filmed. Since the lighting was turned down, the film was very dark, blurry and grainy. It was good enough to be able to observe what was going on in the sessions, but not good enough for much else. The group sessions were watched live. The functionality of MISK-Moves was heavily influenced by these observations. Before the workshop it seemed like object detection and tracking was the way to go, but after the observations it became clear that several of the attendees either had trouble moving around in the room or could not move around at all by themselves. Most could move an arm and/or their head. This made it clear that pose estimation was the way to go, moving forward.

During the first Certec lab session the main goal was for one of the specialist music pedagogues to try out and give feedback on some new functionality in the MISK patcher, and to try out and get a feel for what possibilities are offered by the Room Camera application. Initially there were some issues connecting the app to various functionality in the MISK patcher. The way the OSC message decoding works in the patcher a message to the patcher needs to be formatted in a very specific way. This did not become clear, until the messaging problems occurred. When the inner workings of the message decoding were shown, the solution to the problem became clear and a fix could be implemented there, and then. After the fix connecting the patcher and the application worked as intended, but the values from the application seemed strange and erratic. The output from the pose estimation itself looked correct. It was the calculated values that seemed strange. This made it hard to test out the possibilities available. The next day during bug fixing, it turned out that it was an error introduced when doing the quick messaging fix the day before. A basic copy-and-paste-error made the values for the right arm mostly become 0. It the testing had been done using any other limb, that bug might not have been found. At least everyone got to see the pose estimation in action.

The second session in the Certec lab was a pure developer session. Before that session, the Room Camera application was mostly finished, and the MISK patcher had just received some new functionality. During the session, the messaging between application and patcher were further improved. For the first time, the functionality of MISK-Moves could be fully tested. Both solo playing and collaboration between two people were tested out. Connecting the different control methods offered by the application to several different kinds of functionality in the patcher was also tested out. This workshop also gave the opportunity to test out starting the application remotely from another computer than the one in the lab and streaming the video output

from the pose estimation to that computer.

After the last workshop, the software was basically finished. Only a few adjustments to the messages sent and received were made after that by requests from the developer of the MISK patcher.

3.2 Python

Most of the time spent on the project was on Python related activities. Not only on learning Python, but on installing different packages and libraries to fill the requirements for different network engines. Trying out different models, like YOLO, meant installing PyTorch, TensorFlow, Keras, etc. Other portions of the time spent went into understanding the workings of Python, such as how it handles threading.

Since most of the machine learning research program code is written in Python, as well as most of the example code on GitHub, it seemed appropriate to use that programming language to write the MISK-Moves. Python is an interpreting language, which in theory should make it easy to make the same code run on different platforms, without having to rebuild the entire project each time someone wants to run it on a new platform. A drawback of using an interpreting language is that programs written in such a language usually has worse performance than programs written in a language that requires the program to be compiled before it is run, such as C and C++.

The latest version of Jetpack that can be installed on the Jetson Nano developer kit is based on Ubuntu 18.04. This means that the latest version of Python that is officially supported is version 3.6.9. There are examples of Jetson Nanos running later versions of Python and/or Ubuntu but doing so is not supported by NVIDIA. To do that, a user will have to be willing and able to make any necessary modifications themselves at the risk of having an unstable system. For this project, this is not a viable option, since the main users are music pedagogues and not experienced developers and/or system administrators.

To expand the basic functionality of Python there are plenty of ready to use packages that can be downloaded with either Python's own package manager pip, or by using a third-party manager called Anaconda. Pip only installs python packages, while Anaconda can install packages that may contain software written in other languages than Python. Pip requires the system it runs on to have all the correct versions of the Python compiler and other build tools installed, e.g., C compilers and necessary libraries, since it has to build the packages. Anaconda on the other hand installs prebuilt binaries. It also has built in support for virtual environments, which means that a user can have one custom environment for each different project that they are working on the same development system [2]. Each virtual environment can have its

own Python interpreter, which means that it is possible to have one environment use Python 3.10, and a second one use version 3.8, provided that the system supports both versions of Python. The Anaconda repository contains several of the available Python packages. If something is missing from the repository, there is always the possibility install the missing package using pip instead within an Anaconda virtual environment. Another advantage Anaconda has over pip is that it does more thorough check to ensure that the package dependencies are met and that there are no version incompatibilities between installed packages. The checks do take some time, though.

The Jetson Nano developer kit has to run a special lightweight version of Anaconda called Archiconda, but the functionality and repositories are mostly the same. Even though Archiconda is explicitly made to be optimized for CPUs like the Cortex A57 in the Jetson Nano, installing packages can take time, especially if a package install leads to some extra packages the original one depends on also have to be installed. On two occasions package installations have taken more than 12 hours. In both cases the installations were left running overnight. In this project, one to two-hour installation times have not been uncommon, and times below 10 minutes have been unusual. Note that Archiconda is only supposed to install pre-built binaries and should not have to build anything locally. The long installation times mean that if some Python software requires several packages to be installed, it can take a while before anything can be done with the software. It also means that fixing problems that occur can be a very time-consuming process. Either incompatibility issues, issues caused by the wrong version of a package is installed, or packages being incompatible with each other are problems that have occurred during the MISK-Moves project. This is one explanation of all the time lost in the beginning. Using Archiconda was supposed to be the safe way to do things and should prevent the Jetpack installation from breaking. The unfamiliarity with both Jetson/Jetpack and Python/Archiconda was probably in the way of the realization that Jetpack was probably broken before the Archiconda installation.

3.3 Developing MISK-Moves

The allotted time for the project was 15 weeks, where it was expected that at least 40 hours of work would be put in for each week. Even though this has been a chaotic project time planning wise, time budgeting for each individual week of work has worked well, and the budget of 40 hours/week has been kept. This without resorting to any extra crunch-time before workshops.

Timeline wise, the workshop at Eldorado can be considered to be the unofficial proper start of the project, as that was the first time when it became clear what functionality MISK-Moves needed to provide to become a working and usable tool for the specialist music pedagogues at Eldorado.

The first workshop at Certec can be seen as an unofficial deadline for a fully working feature complete prototype of the application, and the second Certec workshop became the unofficial deadline for the final production release of MISK-Moves. There were 15 working weeks between the Eldorado workshop and the second Certec workshop, which means that the project has in fact gone over budget. At least if the writing of the thesis should fit within the budget.

This project has been more related to a hobbyist maker project than anything else. A lot of the publicly available software and libraries for the Jetson Nano are experimental and are made by enthusiasts. In many cases these are one-person projects. This means that there are no guarantees that anything will work well for any specific use case, or that things are well documented. The software is "as is" and anyone using it is doing so at their own risk, the MISK patcher included. This means that it is difficult to plan for how long certain things will take to implement. Either the libraries and software tools work immediately as expected, or there will have to be testing and exploring of what their actual behaviour and properties are, or something that should be useful will not work at all, and an alternative will have to be found. It has at times been like playing the lottery.

Another thing that makes any structured planning difficult is that the MISK patcher is constantly changing. It did not have the same feature set and user interface at the beginning of the project as it has now. Suddenly, the need for changes in MISK-Moves could arise. Either because of changed functionality in the patcher or to make it easier for future improvements to the patcher.

Initially there was a rough estimate of a time plan, but when about 40% of the allotted time had been spent and there still was no working pose estimation model in place, any initial plans had to be abandoned. From then on, all the activities got interwoven into each other. Software design, documentation, development, and testing could sometimes all be performed during the same hour.

There has been some stalling involved, so that a workshop could be attended at about the same time as this thesis was completed. The reason for doing so is that the workshops were the only chances to test out whether MISK-moves works as intended or not, and if it is possible to use the application for anything useful. That led to some small improvements to make it more convenient for the developer of the MISK patcher to handle messages sent from MISK-Moves. That became the final version.

3.3.1 Software design and development

Since the MISK-Moves application is a fairly small project, the need for an extensive overview of the classes involved is not that great. At the beginning of the software development process, there was an attempt at creating a UML (Universal

Modelling Language) diagram in the beginning. Such a diagram is usually a good tool for overview and to quickly see how things go together and relate to each other. At that stage there were not really any classes, except for the pre-made NVIDIA Jetson Inference libraries and classes, and drawing a UML diagram would not really make any sense.

An early decision was to try to implement the functionality in short iterations. This to make it easier to find bugs and larger logical programming errors. The reasoning was that if only a few lines of code are implemented for each iteration, it should be easier and quicker to find bugs caused by the new code, instead of having to search through hundreds of lines of code.

Another early decision was to use configuration and version management. Since the MISK project already has its own private GitHub repository, Git was chosen as the configuration management tool. During the development, the local commits were synchronized with the GitHub repository at least once per day. The possibility to use different branches for developing and trying out different functionality was used. There were four development branches, before the final version got merged into the main branch.

In the beginning there was only a `main()`-function, but that got refactored into being split into more functions and classes to make the code more readable. To give an overview of the classes involved hand how they fit together a UML class diagram was drawn. The class diagram can also hint at the functionality of each class by showing what methods and attributes are available. It is found in appendix A.1.

Microsoft Visual Studio Code was used as the main development tool. It is a fully integrated development environment with plugins for quick code completion, keeping track of the indentation for Python code, and extensive debugging. It also has several plugins for GitHub integration, which means that everything during the development of the application was tracked, and version controlled. Even though this was a one-person project, the possibility to track all changes and to be able to roll back to any of the previous versions was good to have as a safety.

The name of the functions, variables and classes are quite long. The reasoning behind that is to make the code and the names as self-explanatory as possible. Someone reading the code for the first time should hopefully be able to find out what the various functions and classes do fairly easily.

3.3.2 Testing

Testing has mostly been unit testing. There is no test automation and no existing test scripts, or well specified test cases. In most cases the tests that have been run were to make sure that the program sent reasonable looking values, in that the values seemed

to represent the output from the pose estimation. Another thing that was frequently tested was to run the program for longer periods of time than just a couple of minutes. This to ensure that it would not crash during long sessions, and that long run times would not cause the program to run out of memory, or the hardware overheating. The longest continuous run time has been about two hours.

3.3.3 Figuring out PoseNet and ObjectPose

Here is an attempt at explaining how to use PoseNet: Capture an image from a video stream and tell PoseNet to process it. PoseNet does all the inferencing and post processing, and when it is done it returns the results as one ObjectPose struct per person found in the form of an array. The ObjectPose struct is defined within the PoseNet class. This means, to access the data inferred for individual keypoints, it is found in another struct within the ObjectPose, called Keypoint. To clarify, the x and y coordinates for each individual limb, i.e., keypoint, resides in a struct (Keypoint) found in an array, which resides within another struct (ObjectPose) that is found in an array. That array is the results from the inferencing and post processing that PoseNet returns for each time the `Process()` method is called. The way to do this is not explicitly documented, but it can be found out by reading the various pieces of source code on the Jeton Inference GitHub [14]. An attempt at describing this is found in the UML diagram in appendix A.1, where Keypoint is marked as an inner class in ObjectPose, and ObjectPose in turn is marked as an inner class in PoseNet.

The results from PoseNet are then used to calculate angles, directions, and movement speeds. Basic trigonometry is used for the calculations. Then the results of the calculations are sent to the MISK patcher as OSC messages.

3.4 Max 8 test patcher

To be able to see whether the application sends reasonable values, a test patcher was built in Max. The advantage of a simple test patcher is that nothing needs to be connected in the MISK patcher, and it is possible to quickly try out different things without having to modify the MISK patcher. Otherwise, the message routing would have to be changed, since there is a specific way that messages to the MISK patcher have to be built. This means that any introduction of new functionality will lead to changes in the patcher, which might introduce undesired behaviour. Just testing things out in the test patcher eliminates that risk. More on how the messages have to be constructed and the functionality of it is described in the results chapter.

The test patcher itself has grown and shrunk during the development process, depending on what was needed during the different parts of the project. It has become a patchwork of different ideas and naming schemes. The relevant part of the patcher

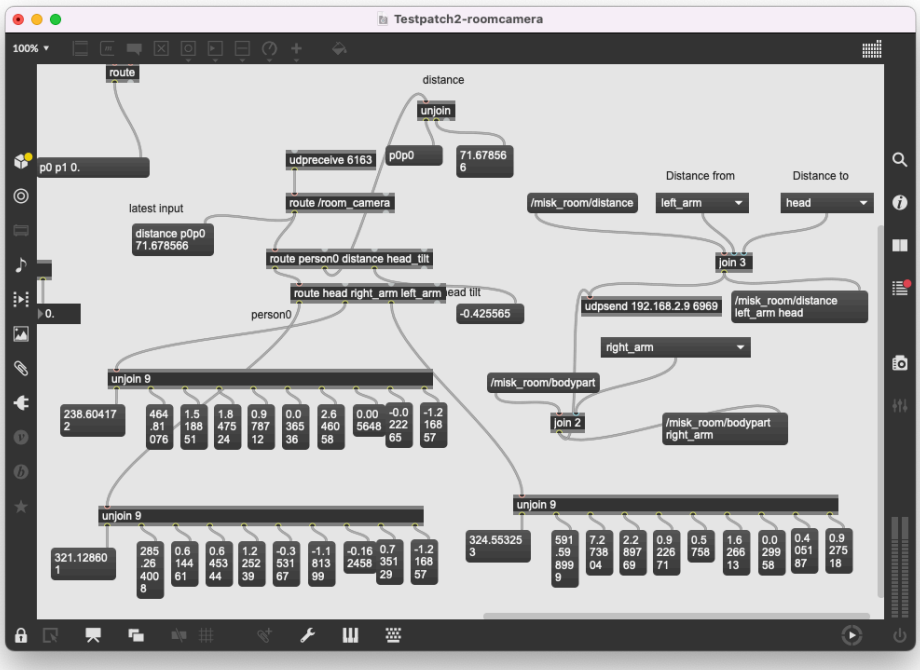

Figure 3.1 The relevant part of the test patcher. What is not shown in the example window is basically more of the same.

can be seen in figure 3.1. What is shown has functionality for both sending and receiving OSC messages.

3.5 Source critique

The information about all the stakeholders in the introduction chapter about Eldorado resurscenter, Certec and Furuboda [1, 12, 13, 18] come from the organizations themselves. In chapter 1 there is also a citation from the COCO Consortium [4], and this information also comes directly from the organization itself.

Most information about hardware products [15, 23, 24, 28], software libraries and protocols [3, 11, 14, 20, 25, 32], and software applications [2, 5], comes directly from the manufacturers, developers and creators themselves. The article about the Nintendo Switch hardware, [19], comes from Digital Foundry, which is one of the most renowned and respected publications for both computer and console games hardware and software technology. The power consumption numbers come from

Notebookcheck, a review site that has been around for over a decade, and they give the impression of being a thorough and serious review publication [27].

There is a lot of explanations of how all the different aspects of artificial intelligence and machine learning works. In this thesis most of the chosen citations come from Data Base Camp. During the research phase of the project several different explanations from several different sources have been read. Data Base Camp seems no better or worse than any other site, and in some of their explanations they have direct links to the original academic papers. Their explanations are often easier to understand than the original papers, and the site happens to be a convenient source since it has a large collection of explanations for most things artificial intelligence. The citations from Data Base Camp are: [6, 7, 8, 9, 10].

A paper still under review is the paper describing YOLOv7 [30]. The claims in this paper are partially corroborated by the OpenCV blog post comparing YOLOv7 pose estimation with MediPipe pose [16]. As mentioned in the background chapter, OpenCV is widely used by several multi-billion companies, which should make the organisation trustworthy enough.

Under the machine learning section there are also citations from two academic papers. One is from the original ReLU paper [21], and one is from the original paper about residual convolutional neural networks [17].

There are also citations from the GitHub pages for the official implementations of YOLOv7 and YOLOv8 [29, 31]. They are as official as something on GitHub can be, and anyone is free to read through, download and run the source code themselves.

[26] are just some install instructions. They are easy for anyone to validate; just follow the instructions to install YOLOv8 on Jetpack 4.6.1 or later.

4 Analysis

In this chapter some of the findings from the pre-studies and some of the choices made will be discussed. It contains the reasoning behind why the hardware and the software tools used were chosen.

4.1 Takeaways from the workshops

Each of the workshops shaped the project in some way. From showing what kind of computer vision machine learning should be used and what kind of features would be useful, to revealing programming errors and at last confirming things working as intended.

4.1.1 The Eldorado workshop

After attending and analysing the material from the workshop, it became clear that pose estimation was the most useful way when translating the movements of a person into a musical instrument. It captures the movements of the entire person, as well as the individual movement of each of the limbs as keypoints. This means that even if a person only has control over the movements of one of the arms or the head, they can still play the instrument.

The values that are calculated for each keypoint, such as positions, velocities, and movement angles, were also based on the observations from the workshop. A complete list of what values are available is discussed in the results chapter.

The camera placement for when setting up the hardware also became clear during this workshop. Before the workshop, it was assumed that the camera would be placed in the ceiling, right above the person playing MISK-Moves. The takeaway from the workshop is that:

1. The ceiling is fairly low. This means that if the camera is mounted in the ceiling the area captured by the camera is quite small. Something like 2x2 m at best.

2. Pose estimation should work better when doing inference on images taken from an angle than on images taken from straight above, as more of the person is usually visible when viewed from an angle.

4.1.2 The first session in the Certec lab

The most important takeaway from this session was the way the OSC messages sent to the MISK patcher should be structured. Once the messages were formatted the exact right way, it became easy to extract and use all the different values sent from MISK-Moves.

There was a severe programming error found more or less by chance. The error made the values fluctuate and jitter heavily, which lead to some strange behaviour when trying to control the different parts of the MISK patcher. Apart from fixing the bug, the wildly changing errors led to changing the behaviour of the application for when a keypoint was no longer detected. Instead of setting a value to zero, which in practice means that limb has moved to the top left corner of the image. The behaviour now is that if a keypoint is lost from one frame to the next, it is assumed that it has not moved and is still in the same position. This keeps the worst value spikes from occurring.

Even so, the values can be a bit jittery. If the values jitter too much, they can be evened out in the MISK patcher by a smoothing function.

A feature request that was brought up during the workshop and implemented was that a person's left and right wrist should be available specifically as a control method choice, and not just the left and right arm. In the final version both wrists and arms can now be used.

4.1.3 The second session in the Certec lab

The results of this session mostly led to changes in the MISK patcher. Some of the incoming message decoding was changed, and the selection of methods (e.g., limbs, velocities, etc.) of controlling the different functions in the patcher was expanded. As another example, the distance between two people, how active a person is, and the distance between a person's wrists became valid methods of control.

During the workshop it was stressed that the program code for the Room Camera application should be extensively commented, so other people can understand what each section of the code does. This to be able to maintain the application and possibly add or change functionality in the future.

4.2 Hardware selection

As one of the motivations behind the MISK-Moves application was to implement it on a system-on-module, what kind of hardware to use was one of the earliest decisions that had to be made. By looking at the sheer number of hits a Google search generates, and the number of GitHub projects, either a Raspberry Pi or an NVIDIA Jetson seemed to be a good fit for the project. There are two reasons for choosing a Jetson. The first is that by looking at benchmarking numbers, and given the wide software support from NVIDIA, and the fact that the Jetson family of systems-on-module is made for ML tasks, a Jetson is better than a Raspberry Pi for the project. The second reason is that Certec has access to both a Jetson Nano developer kit and a Jetson Xavier NX developer kit.

Of the two Jetson modules, the Jetson Nano was chosen. The reasons were that it was less than 1/3 of the price of a Jetson Xavier NX. At the time of purchase, the Jetson Nano was about $100 and the Xavier NX was about $350. Since this is more or less a maker project, keeping the hardware cost down is a good idea, so that other makers can afford to recreate the project.

The camera chosen for the project is the Raspberry Pi Camera Module V2. The V1 module was supposed to be used at first, but it turns out that the sensor in that module is not supported by any hardware drivers for the Jetson. Instead, the V2 module was chosen. The reasons to choose a Raspberry Pi Camera Module V2 over a USB camera is that the camera module small and cheap, while still being able to deliver good quality video with its 8-megapixel sensor. The sensor has multiple output resolutions and framerates available. The purchase cost of the camera module for this project was about 350 SEK, and the recommended price is $25. The size of module itself is about 25x24x9 mm and the weight is 3 g. This makes it easy to place the camera almost anywhere [28]. It there is a need for it, casing and mounts can be 3D-printed for it.

Why not use a smartphone? There is already another MISK instrument that runs on a smartphone, the FaceAR app. So far, the different instruments have been implemented in different ways, and one of the purposes of the Room Camera project is to explore the use of a system-on-module. Another reason is that while a smartphone running MediaPipe will allow fast and accurate pose estimation, as is the case with FaceAR, there did not seem to exist any network models that could handle multiple pose detection with performance equal or better than that of the Jetson Nano. Another argument is that high performance smartphones are usually more expensive than a Jetson Nano developer kit including camera, sd card and power supply. This last argument could be considered invalid though, since most people already own a personal smartphone, and either a pedagog or a personal assistant could use their own phone.

4.3 Installing things on the Jetson Nano

Despite the initial problems installing anything in the way of machine learning models and frameworks, it is fully possible to install and use some of the latest models such as YOLOv7 on the Jetson Nano developer kit. The conclusion that can be drawn from the troubled early parts of the project is that the installation of a Python package must be done carefully and cautiously. It might be good to use a virtual environment for the package installation, and Pip in combination with Virtualenv probably gives shorter package installation times than Archiconda.

When installing packages, it is important to read through the lists of required packages for each of the pieces of Python software that is to be used. The requirements often have a minimum, or a range of version numbers for the packages that the software needs. The latest version of a package is not always compatible with some of the Python software available. This means that when Pip installing packages, the safe thing to do is to always specify a specific version number. Problems might arise during package installation when Pip tries to automatically install packages that the original package depends on. These dependencies are hidden when just reading through a requirements list. The automatic installs can cause that either a too new version of a package is installed, or even worse, it can cause a working package being overwritten by a newer non-working one. If an automatic installation has broken something it can be usually be solved by installing packages in the right order, or by forcing the installation of an older working version of a package after a too new version has broken something. When not having full prior knowledge of all the relevant python packages and their dependencies, the only way to go about it is trial and error. This has been a very time-consuming part of the project.

The same applies not only to Python, but to any libraries and packages in the operating system. In Ubuntu, the Linux distribution Jetpack is based on, software is managed by a packet manager called Apt. When installing packages with Apt, the same level of caution must be observed as when installing Python packages. Just like with the Python packages, it is possible, and sometimes necessary, to install a specific version of a package or library. Otherwise, there is a risk of some piece of software no longer working due to incompatibilities.

In the Room Camera project, no virtual environment was used, and everything was directly Pip and Apt installed directly into the operating system's folders. Even though some packages were built locally on the Jetson Nano, the installation process went much quicker this way than when Archiconda was used. Observing caution, and being careful when choosing the packages installed, and the version for each package, everything installed now works.

4.3.1 Docker containers

In the early stages when things failed to install, using a Docker container could have been an option. Since a Docker image should have all the necessary libraries and dependencies needed to run some specific software, everything should be pre-installed when the image was built. The existing images lacked the Python-OSC library, which means that a specific image would have had to be built for the project. There was a failed attempt to build an image, but since the project was starting to get short on time, the decision was not to spend any time on learning how to properly build Docker containers. The failure was probably once again down to the fact that the Jetpack installation was broken at the time of the build attempt.

The fact that the Jetpack install was broken meant that even running a docker container did not work at times, as the container failed to access the camera or show video output on screen. Normally a properly built and run Docker container has access to all the hardware in, or connected to the Jetson. Now, on a working Jetpack installation Docker containers run just fine.

4.4 Settling for Jetson Inference

After a whole lot of frustration and wasted time, the decision was to give the example classes and libraries from NVIDIA's tutorials Hello AI a second chance. Everything is packaged into the Jetson Inference package. A user can either build the entire package locally on their Jetson, or download and run it as a Docker container, to not have to go through the installation and building process. The MISK-Moves project uses the Python-OSC library/package, which is not included in the docker container. Since previous attempts at building docker containers had failed, the decision was to build the Jetson Inference package locally. This has worked well for both Jetpack 4.5 and 4.6.1-4.6.3. The earlier container build failures might have been down to the broken Jetpack installation, but it felt safer not to tempt fate.

The package contains several code examples written in both C++ and Python, and the latter is used in a series of video tutorials linked from the Hello AI GitHub pages. The examples cover things like object categorization, object detection, semantic segmentation, pose estimation and more. The reasoning behind not using the package from the beginning was that all the video streaming, camera handling and image processing used proprietary classes, made especially for the hardware that is in the different Jetson modules. If someone wants to run it on a Mac or PC instead, the source code for the package needs to at best be modified, and at worst be completely rewritten. Something using OpenCV would support a wider variety of computers and devices. Another potential drawback using Jetson Inference is that the models used are not the latest and most cutting edge. To make their tutorials easy to maintain, and

to make sure that a new user's potentially first ML-experience works and is stable, NVIDIA has chosen something that is tried and tested, and guaranteed to be stable.

The models that come with Jetson Inference work well enough in the Certec lab and its bright lighting conditions, where the Jetson Nano setup has been during the development of the application. Pose estimation does have a high-performance cost, at least when compared to object categorization and detection. The performance from the `poseestimation.py` example is around 15 fps, which means that the latency is at least 67 ms, before any calculations and communication. This is not ideal since a musical instrument ideally should be lag free. During testing, 67 ms was noticeable. Note that this latency comes from the Jetson Inference `poseNet` class alone. Then latency from the position, angle, distance, and velocity calculations, plus latency for inter-device communication will be added. There does not seem to be any other pose estimation models, or implementations that seems to be any faster on a Jetson Nano though. Doing Google searches, it is more common to find object detection and object tracking performance numbers than for pose estimation.

Running the Python version of the package might seem to be a potential problem. However, when looking in the build directories after the Jetson Inference package has been built, all the important libraries are built using C++, which should guarantee that the performance is about as good as it can get. The ResNet-18 pose model used is converted to a TensorRT network. It is NVIDIA's proprietary network engine, and it is optimized for NVIDIA hardware.

If there is a need for it, the model can be trained further to be better suited for a specific need. In this case, it had no problems detecting both standing and sitting persons during testing. It could also estimate where a limb would be in some cases, even when a limb was partly or wholly obscured. There is the possibility to set the confidence value for how high the probability needs to be, for it to report that the model has found a limb (or rather, a keypoint). If it is below the set value, the limb will count as not found. By default, it only needs to be 20% sure that it has found a limb, which gives a bit too many false detections. This leads to it thinking that chairs, bags, and jackets are limbs belonging to a person. Raising the value to 30% gives less false detections, while still not losing track of actual limbs too easily. With this in mind, the model should not have any difficulties detecting disabled people, without having to retrain it. Hopefully it will work as well in future Eldorado settings as it has done in the Certec lab during testing. It would have been a hard and time-consuming task to gather enough high-quality training data, if there would have been a need to do any training on the model. Ideally thousands of training images would have to be gathered and prepared. Getting access to enough people with a sufficiently diverse set of impairments and disabilities for the training to be relevant and of high quality could have been problematic.

After wasting a month and a half on trying to find the best model and network im-

plementation and trying to get different cutting-edge models to work on a Jetpack installation that probably got broken in several ways early on in the research process, the poseNet example from NVIDIA's Jetson Inference was the best alternative. It worked, it was optimized for the hardware, and it was stable.

4.4.1 A quick pose estimation test using YOLO

After MISK-Moves application was mostly finished, there was another quick look into what DeepStream could do. This was just to be able to prepare for a discussion on if it had been better to develop MISK-Moves as a DeepStream app instead. Among the tutorial lessons were implementing a YOLOv3 object detector. Ordinary object detection and tracking has a performance of around 13 fps. That would have made pose estimation performance even lower, which would have made it worse than the Jetson Inference poseNet class. Another thing that made the YOLOv3 model worse was that the accuracy was not good, since visibly the bounding boxes for the detected classes constantly kept dropping in and out.

YOLOv7 is now successfully installed on the Jetson Nano developer kit in the lab, and out of curiosity the yolov7-w6-pose model was tested. The accuracy might have been slightly better, than the ResNet-18 model used, but the frame rate was worse. MISK-Moves will have to be modified if someone wants to change models to YOLOv7 pose, since the numbering of the keypoints is different.

4.4.2 A reason why MediaPipe pose is not used

One of, if not the most accurate and high-performance pose estimators is MediaPipe pose [16]. It is one of the most commonly used pose estimators, and it is used for real-time pose estimation on systems with limited resources such as CPUs and edge devices like mobile phones and Raspberry Pis. As shown in VIDEO it usually does a better job tracking the limbs on one person than, as the tracking in many cases seem more accurate and less jittery.

It is possible to install MediaPipe on the Jetpack version the Jetson Nano currently has, and the current install should have all the libraries and packages needed in their respective required versions. The version of each library and package is important, and anyone installing MediaPipe has to make sure that the correct versions for each and every package and library are installed, and that no other updates or package installations overwrite the required installed versions. Otherwise, the MediaPipe installation will either not work to start with, or it will break after an update. At the time when Jetson Inference PoseNet was chosen, the project was running out of time, and it was much safer to choose something that was known to work.

Another drawback that MediaPipe has is that the pose estimation only works for one

person at a time. PoseNet on the other hand handles multiple person pose estimation. This means that using the distance between two people as a control method in the MISK patcher would not have been possible if MISK-Moves was based on MediaPipe. Another quirk that MISK-Moves would have had is that if MediaPipe would have started tracking the wrong person, that person would have to go out of the video frame, to give the model the chance to start tracking the right person. As it works now, it is possible to select the person that is in control of the patcher if there are more than one in the video frame. A feature like that is impossible with single person tracking.

5 Results

Here the results are presented. Both what the MISK-Moves application does, and its performance can be found in this section. There will also be some reasoning regarding why the poseNet class using a ResNet-18 model is the basis for the pose estimation in the application.

5.1 A quick overview of MISK-Moves's features

The most important features of MISK-Moves are the OSC messages sent containing information about the movement of one or more persons, i.e., the player(s), and their limbs. This can then be used to control various functions in the MISK patcher. It is also possible to change some of the MISK-Moves settings from the MISK patcher. The message structure and the contents are described in the sections that follows in this chapter.

It is possible to run and shut down MISK-Moves remotely, eliminating the need for a dedicated mouse, keyboard, and screen for the Jetson module.

5.2 Selection in the MISK patcher

To be able play music, i.e., make the MISK patcher produce sound, the user needs to select what values sent from MISK-Moves should control which functionality in the patcher. From the start screen in figure 2.2, the input for each channel can be set directly. When a user clicks on the channel button, a channel selection window pops up where all the different MISK instruments can be chosen. The output from MISK-Moves is labelled "RoomCam" in the drop-down menu. After that the user selects "person0" to "person4" to get the limb movement values from that person, and "distance" to get various distance values. The pop-up window with the different selections are shown in figure 5.1. The different input values are all remapped to values between 0 and 127. This is an old legacy remnant from the MIDI protocol. When the channels the user wants to use have the desired input, it is then possible to tell the different functions in the MISK patcher what channels to use to control the different sound generators and sequencers. The user selects the tab of the sound generator or sequencer that is to be played, and on each tab, there are one or more

Figure 5.1 The different input selections available.

channel selection buttons for the different functions of each module. These can be seen in figures 2.4, 2.6, 2.5 and 2.7. Pushing the channel select button opens the channel selection pop-up window in figure 5.2. In the channel selection pop-up win-

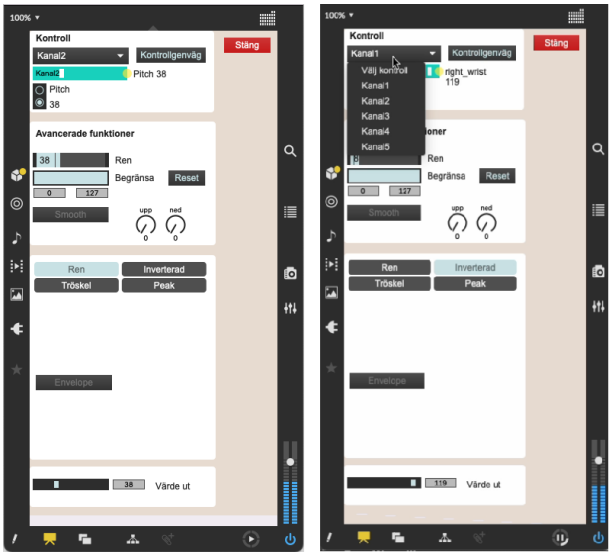

Figure 5.2 The channel selector in the MISK patcher. The right image shows an open drop-down menu with all the selectable elements.

dow, there are functions for smoothing out jittery and noisy input. It is also possible to invert the input values if that is more convenient for playing the sound or music. As an example: Moving your hand towards the ceiling will yield lower and lower values from MISK-Moves, while moving it towards the floor will increase the value. If a player wants to increase the volume or pitch by raising their hands towards the

ceiling, the invert value function will do just that.

MISK-Moves sends enough data to be able to work as a controller for all the five available channels in the MISK patcher at the same time. This means that a person can control several aspects of the music by moving around and doing different things with their limbs, or that two or more persons can collaborate and control one or more things each in the patcher.

5.3 Output from MISK-Moves.

To be able to control things in the MISK patcher MISK-Moves sends data in the form of OSC messages. Instead of sending one message per limb found, and one for each distance calculation, etc., all individual messages are put into an OSC bundle. The application sends one message bundle per captured video frame. This is to keep the number of messages sent from the application down, but the total size of the data that gets sent is still the same as if one package per limb and distance calculation would have been sent.

Message bundles are all sent to the OSC address `/room_camera`. This is the address that the MISK patcher expects to receive messages from MISK-Moves on. The length of the messages varies depending on what kind of message it is. Messages containing the distance between two people, or between two limbs consist of three arguments. The longest messages are the ones for the centre position, the limbs and various keypoints. They consist of eleven arguments.

When the MISK patcher in Max 8 receives a message from the application it always sorts the messages by the first argument, and then the user can choose which values to use for controlling the different instruments and effects in the patcher.

Instead of putting one message per keypoint found, only the values from five limbs get sent. This saves a bit of processing time compared to when data from all the 18 keypoints for each person found were sent. However, calculations for all the keypoints found in each frame are still being performed allowing for instant switching between what limb to track. Otherwise, there would be a small delay before accurate average speed and angle values could be shown.

A peculiarity of the poseNet class is that there is no user control over how it assigns identities to any persons found in an image. Theoretically in a worst-case scenario, the identities can get swapped around from frame to frame. That behaviour is extremely rare, since it has not occurred yet during hours of testing, but it is a possibility. In any case, it is impossible to know which person gets assigned which identity number. All persons found will show up in the selection menus in the MISK patcher. The user will have to figure out which person is which. This can be done by watch-

ing the video output, which will show all the persons detected, and by seeing which person's movements correspond to which input values in the MISK patcher. If there is a need to remove a person identity from the selection, make sure that person is out of the video frame for at least five seconds. This timeout can be set to a longer or shorter period of time at startup.

The results of the video output are shown in figure 5.3 and 5.4. In both images any keypoints and links are drawn forming a kind of skeleton. Where the network model finds a pair of keypoints questionable, it does not draw a link. That is why there are keypoints without links between them in some of the examples. In some cases, the network model guesses where the link is, even if the limb is partially or completely obscured, and even though a person is sitting behind a table both keypoints and links are still being inferred. As shown in figure5.4, the model can detect a sitting person.

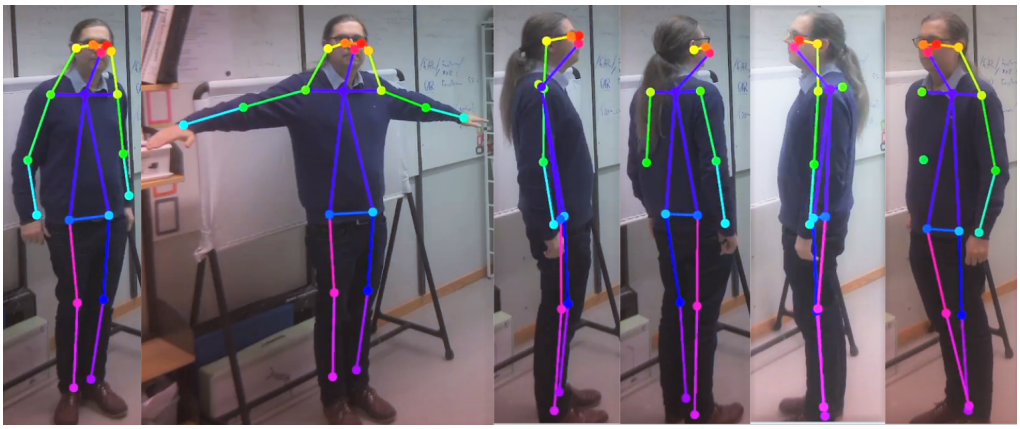

Figure 5.3 Streamed video output with overlay showing detections on standing poses from different sides.

This should mean that it is able to detect a person that uses a wheelchair. Due to the low confidence value threshold for detections, false detections should be a larger problem than the model not detecting a person. It should not matter whether the person has any visible disabilities or not.

5.3.1 Person, limb and keypoint message contents

The first argument in these kinds of messages is always "person<number>", where <number> is the id number that the person has received by the pose estimation. The numbers start at zero, which means that 'person1' would be the second person, or the pose with id = 1, found.

The second argument can have one of these values:

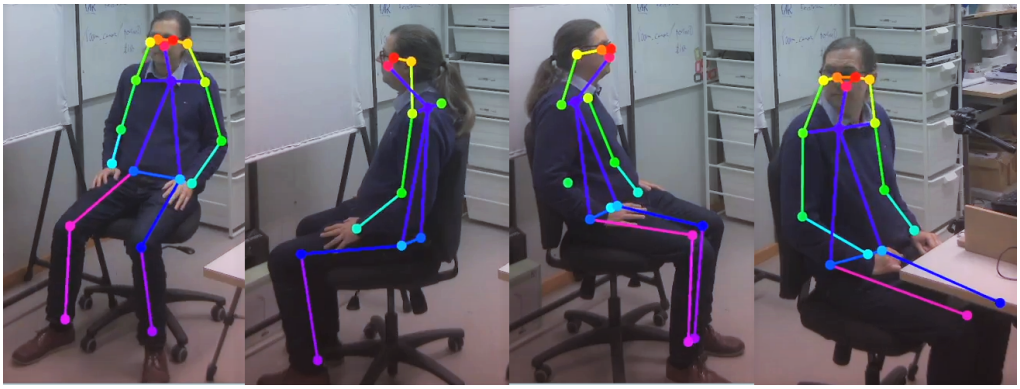

Figure 5.4 Streamed video output with overlay showing detections on sitting poses.

'center' – Each pose, i.e. person, has an invisible bounding box around them. The bounding box is used for calculating the values representing the centre position of the pose, i.e., person.

'head' – The values come from where the head is supposed to be.

'left_arm' – Values from the left arm.

'right_arm' – Values from the right arm.

'left_wrist – Values from the left wrist.

'right_wrist' – Values from the right wrist.

'keypoint' – If the application sends values from a specific keypoint instead of one of the limbs specified.

The rest of the nine values are as follows in the order of the arguments in a message:

1. The x-position in pixels.
2. The y-position in pixels.
3. The instantaneous movement speed between the last and the current video frame in pixels.
4. The average movement speed over five video frames in pixels.
5. The length of the movement vector that is the result of adding the movement vectors from the last five frames.

6. The instantaneous movement direction in the current video frame in radians. The values follow the unit circle, which means that movement along the x-axis in the positive direction has value 0 and $\pi/2$ for movement along the positive y-axis.

7. The instantaneous movement direction change between the current video frame and the last in radians.

8. The average movement direction changes over the last five video frames in radians.

9. The vector angle of the movement vector that is the result of adding the movement vectors from the last five frames.

10. The head tilt angle. This value is only sent for the 'head', and it is 0.0 for all the other limbs.

Select one of these values to control a function in the MISK patcher. It is possible to select multiple values from the same person to control multiple things at the same time, or multiple values from different persons. In theory, it is possible for each person to control ten functions each at the same time in the patcher.

5.3.2 Distance measurement messages contents

Messages containing distance measurements always have the string 'distance' as their first argument, to help the MISK patcher sort the messages correctly.

The second argument is a string that shows between which poses, i.e. persons, that the distance is measured between. The strings have the format 'p<id1>p<id2>', where <id1> is the id number of the pose that the distance is measured from, and <id2> is the id number of the pose that the distance is measured to. The string 'p0p1' would mean that the distance is measured from person one (who has id = 0) to person two (who has id = 1).

Measurements between a person's own two arms, or between one of their arms and their head always have the same two id numbers in the string. For the distance between the arms of person one the string would be 'p0p0', as an example. All the available distances calculated can be selected from a list in the MISK patcher when selecting the control method.

The third argument in the distance messages is the measured distance value itself. The distance is measured in pixels.

5.4 Available settings.

It is possible to send control messages to the MISK-Moves application to change some of the contents of what gets sent and to change what gets calculated. Here is a list of what can be changed in the MISK-Moves application, and the OSC addresses to send the change messages to. The emboldened text shows the exact name of the addresses the application listens to.

/misk_room/bodypart – Use this address to change what limb gets tracked. Valid values to send to that address are the string values:

- 'head'
- 'left_arm'
- 'right_arm'

In the final version of MISK-Moves, this has no effect since everything gets sent to the MISK patcher where the user selects which input they want to use directly. This way, the user interface in the MISK patcher gets less cluttered. Not having to put an extra menu for the limb selection meant that the interface can have a more consistent look for all the different MISK instruments in the input selection in figure 5.1. The developer of the MISK patcher also prefers to have all the data available at all times, at the time of writing this.

/misk_room/distance – When calculating the distance between two people the application needs to know which limbs to calculate the distance between. A message sent to this address needs to contain two string values. The first string is the limb of the first person from where to start the distance calculation from. The second string is then the limb of the second person to calculate the distance to. The valid strings are the same as for which limb to track above. By default, it calculates the distance between the right arms of the two persons.

/misk_room/self_distance – It is also possible to set the distance calculations between a person's own limbs. By default, the distance is calculated between the left and the right arm, but it is also possible to calculate the distance between the head and the left arm, and the head and the right arm. Just like when setting start and endpoint for the distance between two people, two strings need to be sent in the message. Valid strings are the same three strings as for the limb selection.

/misk_room/head_tilt – Use this address to set whether the application should send the head tilt angle or not. Send the string 'yes' to turn the messages on, and 'no' to turn them off. Head tilt messages are turned off by default. This is also no longer in use in the release version.

5.5 Test patcher

The currently working parts of the test patcher can be seen in figure 3.1. Packages from the MISK-Moves application are received by a standard Max 8 `udpreceive`-object, and packages to the application are sent via a standard `udpsend`-object. When sending messages, the user will have to know the IP-address of the Jetson Nano, otherwise the packages will just be sent into the void. Vice versa, the MISK-Moves application will have to know the IP-address of the computer running Max 8, or the messages sent will not be delivered.

Since there are several different instruments and devices that can send messages to Max 8, the messages from MISK-Moves will have to be specifically routed. This is done by the object `route /room_camera`, which selects messages from MISK-Moves only. The messages then need to be split depending on what kind of message it is. Currently the test patcher reads messages regarding person0, distance and head tilt.

Max 8 has no easy way (or even a way at all?) to use wildcards as part of the arguments of a `route`-object. This means that if there is a need for receiving messages from more persons than person0, then they have to be added as arguments. To give an example: If information about three persons are needed at the same time, the arguments of the route object will have to be "person0 person1 person2", and not just "person0" as it is now.

The test patcher shows the last received message for each category. There is a lot more information that could be shown since there are at least six messages sent for each captured video frame. The reason not to show everything is that it would make the test patcher even more cluttered. Each message from a limb would generate a route argument, an `unjoin`-object and nine message boxes to show values in.

It is also possible to change the settings of the MISK-Moves application from the test patcher. The first available setting is to choose between which limbs the application should measure distance, and the second is which limb to track.

5.6 I/O classes

MISK-Moves has three running threads: The main thread, a thread listening for keyboard input, and a thread listening for messages from the MISK patcher.

It is necessary to have a separate keyboard listening thread, otherwise it would not be possible to exit MISK-Moves by pressing the esc-key or the q-key when running the application remotely in a terminal window. When running locally on the Jetson Nano, keyboard input can be read by using the underlying Linux X-server. This is

what happens when a user uses “esc” to quit any of the Jetson Inference example applications that has a window showing the video output. When running programs remotely from a terminal window there is no running X-server, and anything relying on one will crash an application directly on start-up. Instead, some trickery has to be used. There are ways to poll for keyboard presses in a terminal, but polling would take up unnecessary processing time. Instead, the polling can be run in a separate thread which is put to sleep until a key is pressed, thereby almost no processing time is wasted.

The thread listening for messages from the MISK patcher is similar. There is the possibility to use a version of the Python-osc server class that uses the built in Python async functionality, but using that class increased the latency considerably. The threading version of the class is not very well documented. Instead, the simple blocking server class is used running in a separate thread. Having it run in a separate thread means that it can block the thread until a message from the MISK patcher was received, without blocking anything else. Since it only blocks its own thread, almost no processing time is wasted until a message is received from the MISK patcher. When it does receive a message, the time it takes to process the message has been negligible in testing.

5.7 Things provided by the Jetson Inference classes

By using NVIDIA’s premade Jetson Inference classes, a couple of nice-to-have features are in the MISK-Moves application. Each one of these features would have taken some time and effort to develop from scratch. These features are:

Video output in several different ways. A user can choose between one or more ways to output video. Possible output methods are:

- Playing the output video directly in a window on the Jetson Nano’s desktop. This is the default when running MISK-Moves. This can be turned off by giving the command line option `--headless` when starting the application.
- Saving the video as a file.
- Streaming the video over RTP to another device.

Video overlay on the output video. By default, all the keypoints and any links between keypoints are drawn as points and as a skeleton, see figures 5.3 and 5.4.

Video input can be either from a CSI camera, USB camera, file or an RTP or RTSP video stream. The user can choose between the h264, h265, vp8, vp9, mpeg2, mpeg4 and motion jpeg codecs

Different ways to encode the video output. The available codecs are the same as for the input.

This means that the MISK-Moves application can be run without having to connect the Jetson Nano to a display, a keyboard, and a mouse. Instead, it can be run remotely from e.g., the computer running Max 8, and the video output can be viewed in a video viewer such as VLC. Being able to watch the output video enables a user to see whether a person is in the video frame or not, and to see what keypoints the application is able to find. This can help troubleshooting and will help the user to directly make adjustments to either the camera, the lighting conditions in the room or the positioning of the persons in the room, to make the pose estimation work.

Another inherited feature is the possibility to change network models and to change the confidence value threshold for keypoint detections. The threshold can be set from 0, which means that anything is a keypoint, to 1.0, which means that the network model has to be 100% sure that something detected is a keypoint. Network models that are accepted are Caffe and ONNX models, and the translation from keypoint numbers to limbs has to be the same as the default ResNet-18 model. Otherwise MISK-Moves will detect a keypoint, but it will assign the wrong limb to it.

6 Discussion

Here is the final discussion on the current state of MISK-Moves and how it could be improved in the future. There are also some usage recommendations.

6.1 User testing

Since there has been no opportunities for access to the users at Eldorado during the software development period of the project, there has been no testing on users with any severe impairments or disabilities. Any testing done have been performed on project members or other students at the department.

There will be user testing at Eldorado after the MISK-Moves thesis work is finished. This means that testing will be done without any support from the developer. If anything important is learned during the testing and changes should be made to MISK-Moves, someone else will have to implement the changes. Potential changes include bug fixing, changing, adding, and improving functionality, or removing functionality that does not work or provide anything useful in real usage.

One of the things that probably would have benefited the quality of the final version of MISK-Moves the most is testing during real musical sessions at Eldorado. Sadly, that was not realistic due to distance, time and resources. Hopefully the next project gets more opportunities for test sessions at Eldorado.

One important thing learned during real life testing at Eldorado after the development MISK-Moves was finished was about discoverability [22, Chapter 4]. It is important to have the output video stream clearly visible, otherwise MISK-Moves will be ignored. Unlike the mat and pillow, there is nothing clearly visible in the room that hints at anything in particular going on. Correlating moving around in the room or moving the limbs to changes in the sounds or music playing, or to an instrument suddenly playing different notes is hard for anyone. This regardless of whether they have any cognitive impairments or not. Especially if there are lots of other things going on in the room at the same time. Showing the video with the keypoints and links will hopefully help attracting people to MISK-Moves and help them discover that there is in fact something going on. Seeing themselves make the skeleton-like overlay move on top of them on video will hopefully make them realise that their movements cause changes to the sound and music playing. From a purely practical

standpoint, there has to be enough bandwidth over the network connection between MISK-Moves and the device running the MISK patcher for the video not to hitch and stutter too much. The bitrate of the output video can be set at startup, and the bandwidth limits will vary from location to location. It is recommended to thoroughly try out the appropriate bitrate for each location in a couple of test runs before the actual music sessions. Encoding the video stream with h265 instead of h264 will help keep the bitrate down, while maintaining as high image quality as possible. This is also selectable when starting up MISK-Moves.

6.2 Latency and precision

Pose estimation is a very compute intensive task. Just the pose estimation alone, when running NVIDIA's pure `poseNet` example application, typically takes between 67 and 83ms to process each video frame. With the calculations and OSC message communication added by MISK-Moves, the total latency grows to between 85 and 145ms instead. The more people in the video frame, the lower the performance. This latency causes noticeable delays in MISK-Moves's reactions to user movement, which means that controlling the music or playing notes can feel sluggish and unresponsive. The latency, or any side effects of it, does not make MISK-Moves unstable or increase the risk of any software crashes. It just makes it feel a bit slow.

The `poseNet` class does pose estimation on each video frame it receives. It does not remember anything from the previous frame and the placement of each keypoint can and will vary from frame to frame. This means that the values for each keypoint jitter, and sometimes the jitter can be quite severe. Especially if a keypoint is lost, then found, and then lost again, and so on. As mentioned in the results section, it is possible to set the threshold for the confidence value for when the network decides that something is a valid keypoint or not. Setting the value is a compromise between getting too many false keypoints and the network not finding a keypoint where there should be one. A too large threshold value also increases the jitter. Setting the value too low during testing has shown less jitter, but has also made the network claim that chairs, tables, jackets, and bags are persons as well. A suitable threshold value will have to be tested out for each location a user wants to run MISK-Moves in. The default threshold value is 0.2, i.e., 20%. During testing value of 0.3 has been a better compromise, giving less false detections while still not making the jittering much worse.

In the future there might be network models that are better at finding poses in a frame, leading to more stable keypoints that do not move around as much from frame to frame. Another good idea, if there was more processing time available would be to add object tracking of the keypoints. This is what the previously mentioned MediaPipe pose does. The results of that can be seen in the MISK Face-AR app,

which produces a much less jittery output. As it is now, object tracking would add more latency, making MISK-Moves feel even more sluggish and unresponsive than it already feels. Faster processing time might also mean less movement between each captured video frame, and that could potentially lead to slightly less jittering.

Another thing that could reduce jittering is to use higher input resolutions. That means running the original Jetson Inference classes without the previously mentioned modification which lowers the input resolution. On faster hardware, inferencing done on higher resolution input might not have as much of a performance impact as it has on the current hardware. Now maximum precision has been sacrificed for performance.

When viewing the video output in VLC on a remote device, the latency seems worse than it is. This is because of most, if not all, streaming video players have built in added latency for stream buffering. It is to make sure that a video stream can be viewed without any interruptions or stutters due to networking issues. Viewing the stream is still useful to see the quality of the input video, and to see what the pose estimation detects.

6.3 Possible software performance improvements

Some performance improvements can be done to the software. The current version calculates and sends a lot of different values to the MISK patcher. Several of those values might not be useful in real use cases, but that cannot be known until extensive user testing has been done. As mentioned earlier, that fell outside the scope of this project. A recommendation for future improvements is to remove any unnecessary values and calculations. If allowed to speculate, that could improve the total processing time for each frame by perhaps somewhere between 5 and 20 ms, depending on how many people there are in the frame.

To improve the performance of the pose estimation another network model or implementation could be used. There might be new pose estimation network models that are even less compute heavy and be able to detect poses better than ResNet-18 in the future. If tracking one only person at a time is acceptable, MediaPipe pose mentioned in section 4.4.2, will give both better performance and more stable detections. The currently used `poseNet` class itself should be well optimized, since it is made by NVIDIA, the manufacturer of the Jetson Nano themselves. MISK-Moves is written in Python, but the `poseNet` class, and all the other utility classes from NVIDIA are written in C++ and pre-packaged for use with Python. This should make them well optimized and run as efficiently as can be expected. Experienced NVIDIA engineers with intimate knowledge about the hardware should be able to write more optimized and efficient code than an engineering student.

There might be faster and more efficient classes for OSC communication than the Python-osc package. Turning the communication off and on only adds a couple of milliseconds at worst, and fractions of a milliseconds at best to the total latency. There is not a great deal of performance that can be regained here.

If the application is re-written in C or C++, there might be a performance improvement. It is unlikely that an optimization like that is worth the effort. The `poseNet` class causes a minimum latency of 66.7ms. On the current hardware the latency will be in the magnitude of tens of milliseconds to between a hundred and two hundred milliseconds. It will never be in the single digit numbers, or fractions of a millisecond. The camera used will also be a limiting factor, but more on that in the next section.

6.4 Possible hardware improvements

As of 2023 there is a new Jetson developer kit, the Jetson Orin Nano. This new SOM has significantly higher performance. For the Bodypose Net in NVIDIA's Jetson benchmarking suite [24] the Orin Nano developer kit (8GB) is about 44 times faster than the original Nano. The Orin Nano processes 133 frames per second, or fps, while the old Nano only processes 3 fps. This implementation and network model seems heavier than the `poseNet` class and ResNet-18 model used in this project (about 12-15fps), so the performance gains might be different. The worst performance gain in the benchmarks is about an 11 times improvement for Action Recognition 2D. If the performance gain for `poseNet` would be about 10 times, a slightly conservative estimation, it would mean that the latency introduced by the machine learning part of the application would go down from about 67-83ms to 6.7-8.3ms. Such latency would be less noticeable and would make playing MISK-Moves feel much more responsive. The performance would come at a higher price, though. The Jetson Orin Nano developer kit has a recommended price of $499, versus the $149 of the original Jetson Nano developer kit.

Another way to reduce the latency is to make sure that the camera is able to capture video at a high framerate. The video capture's contribution to the total latency should ideally be the frame time of one video frame. The current Raspberry Pi V2 camera can capture 120fps, at 1280x720 pixels at its fastest mode. That mode results in an image heavily cropped down from the full sensor frame. The resulting frame has a narrow field of view, and the camera will have to be placed far enough back to make sure that everything fits into frame. The full sensor frame is 3264x2464 pixels and allows for a camera placement closer up, but then the framerate is limited to 21 fps. Currently this is not a problem, since the pose estimation and processing take longer than a frame, but when running on a Jetson Orin Nano the framerate of the camera would be a limiting factor. 21 fps means a frame time of ≈ 47.6 ms, 60 fps a

frame time of ≈ 16.7 ms and 120fps a frame time of ≈ 8.3 ms. As mentioned in the previous paragraph, the Jetson Nano Orin processes the compute heavy Bodypose Net benchmark at 133 fps. This means that the higher framerate the camera can capture at, the more responsive MISK-Moves will feel to the user.

6.5 Recommended usage for MISK-Moves

It is possible to play notes directly and to use MISK-Moves to trigger samples. Currently, the somewhat slow and unresponsive behaviour of the application does not make those ways to play the best experience. Playing the drums, or just making a cymbal crash would make the sound feel too disconnected to the movement, and the player would risk hearing their sounds play too late to be in time with the music. Sounds and effects would probably not activate when the player would expect them to sound. During testing it was possible to play the "air accordion", if the player was slow and deliberate in their movements. To the testers it was a somewhat meaningful way to play, or at least a novelty.

What has worked better for MISK-Moves during workshops and test sessions is to use it to control different aspects of the sound or music being played. The most satisfying results for the testers have been when the application has been used to control pitch, volume, and various sound filters. The latency has not mattered as much when the qualities and the intensity of the sound(s) and the music playing is controlled by the player, at least during the tests. Controlling the music this way has made MISK-Moves feel more responsive, than when playing notes directly or when using it as a sample trigger, even though the actual latency stays the same all the time. Examples of successfully used input values used for this has been distance between two persons, the distance between a person's left and right wrist, movement velocity, x and y positions, movement direction, and average movement direction.

Another good use of the application is to use the intensity of a person's movements to control the intensity of the music or sound. The average velocity, and secondarily the average direction change, represent how much a person moves, or how active they are. To make the sound and/or music feel more or less intense a combination of pitch and volume can be used. Usually higher frequencies, a higher tempo and a louder volume feels more intense than something with a low frequency, low tempo and at a lower volume. Setting different synthesizer filters can also make sounds feel more or less intense. It is possible to make sounds sound sharper by changing the waveform, and things like attack, decay, velocity, sustain, and other commonly used synthesizer sound parameters can also make a sound more or less intense.

A good use for x- and y-positioning has been to activate different melodies, soundscapes or sound moods when entering different parts of the video frame. That kind

of control is good to use for activating different soundscapes and/or melodies depending on where a person or one of their limbs is within the frame, and how they move around in the room.

Note that the pose estimation is only done in the two dimensions of the video frame. Only width and height count and moving closer to or further away from the camera has no real effect. This is where watching the streamed video output can help, as it shows how much vertical and horizontal movement the pose detection can see at any given moment. It also shows what keypoints the pose estimation has found for each frame, and it can help the user to adjust the camera or move the player so that the camera can see the action more clearly.

6.6 Ethical aspects

Ethics could have been a major issue in this project, since it is based on capturing the users on video and sending that input through a machine learning network, to get the desired results. None of the input is saved and it is not used for any further training of the network either. All the processing is done locally on the Jetson Nano module, and none of the input gets sent over any network for any kind of remote processing.

However, a user needs to be careful about what they decide to do with the different output options. Unless the user wants to test things out on themselves, the option to save the output video to file should not be used unless there is explicit consent from all persons captured on camera during the session. Even then, the GDPR rules and regulations have to be followed for any usage and storage of the material. As for the option to stream the output video to another device, the user needs to be careful about where they are sending the video stream. At Eldorado all the MISK instruments will run on a private network that will not be connected to the Internet. This hopefully means that no video output ends up where it does not belong.

If another project wants to train their own network model or do reinforcement training in the future some things would have to be taken into consideration. There would have to be a discussion on how to obtain and store training data, and there would have to be ethical discussions on how and where the training should be performed. As an example; using external cloud services (i.e., Google Collab) in this case might not be legal under the current Swedish law. Any such future issues, however, are left to future projects.

7 Conclusions

The final conclusions of the thesis. It sums up the most important take-aways from the thesis and has some speculation of future use and development.

7.1 Conclusions and summaries

MISK-Moves makes it possible to use the body, or individual limbs, as a musical instrument. It allows for individual as well as collaborative play. This is done by having a machine learning model do pose estimation on a video feed from a camera somewhere in the room. The machine learning model used for this application is ResNet-18 pre-trained on the COCO dataset, which in testing has proven to work very well for the intended purpose. The results of the pose estimation are then analysed and interpreted to be packaged and sent as OSC messages to some music software that generates or makes the actual sounds and music. In the MISK project the software is the MISK patcher, running in Max 8. The output is made with the MISK patcher in mind, but since the messages are standard OSC messages, PlayfulMoves can be used to control any application that accepts messages in that format as control input. It is not locked or tied to the MISK patcher and can be used with other present or future projects and software as well.

Primarily, MISK-Moves is meant to be run on the original NVIDIA Jetson Nano developer kit hardware, but it should be able to run on any present or future NVIDIA Jetson family hardware. The requirement is that the hardware supports NVIDIA's Jetpack development and operating system environment, and that it supports the NVIDIA Jetson Inference package. The application is built in Python 3.6, but as long as the Jetson Inference package works correctly, it should run on later versions of Python without any problems. The video feed can come from any camera that is supported in Jetpack. All this means that it is possible to swap out the current hardware for more modern or better performing Jetson modules and cameras in the future.

The output from MISK-Moves can be a bit jittery and uneven, making controlling the music feel and sound a bit erratic at times. There are some possible solutions to help mitigate this behaviour. There is a smoothing function built into the MISK patcher that can even out the worst spikes and jitters from any input data. Another

thing that can help calm down the jitters is to try out what confidence value works best for the pose estimation in the lighting and room conditions MISK-Moves is currently running. ResNet-18 will only accept that it has found a pose, i.e., person, if its confidence in that it has found a pose is above the threshold value. Some of the jitters occurs because ResNet-18 loses track of a limb or a person. Setting the confidence value too high means that the model loses track more easily, but setting the value too low will increase the number of false detections. It is best to avoid it detecting chairs, bags, strewn around items of clothing, etc. as people. What works the best will have to be tried out from setting to setting. The last and most expensive way to try mitigating the jitters is to run MISK-Moves on more powerful hardware. A higher video input resolution would increase the level of detail the model has to work with, and the ability to capture more frames per second would mean an increased movement resolution. A higher movement resolution means that it is possible to capture faster and smaller movements, and that things do not move around as much from one frame to the next, as when the movement resolution is low. Especially for fast movements. In theory, this should help ResNet-18 detect persons and limbs more easily, leading to less jittering.

The input latency is noticeable to most people playing MISK-Moves. Typically, it can be 83 to 150 ms, depending on the number of people in the current video frame. This much latency means that MISK-Moves currently might not be very well suited to play musical notes and melodies directly. When played very slow and purposefully it works, but when trying to play most melodies at Adagio (perhaps even Lento) or faster, the latency will be in the way. When playing percussion, the latency also gets in the way, causing drums and cymbal crashes to sound too late and be off-beat. What has worked better in testing is controlling things like intensity and the different qualities of the music. This includes volume and different sound filters. Another thing that works well is triggering different melodies and soundscapes depending on there the person, or a limb, is in the room. When a looped sample, a melody sequence, or an atmospheric soundscape is playing, controlling the pitch, filters and volume also works well. Especially if multiple samples and/or sequences are running at the same time. During development the resolution of the input video was lowered and the number of OSC messages sent was kept down to optimize for as much performance as possible. The way to radically lower the latency is to run MISK-Moves on faster hardware, like the new Jetson Orin Nano developer kit.

To draw people in and make them realize that it is their movements that control the music, it is a good idea to show the output video feed. The larger the image, the better. Just capturing movements makes it hard for anyone to realize the cause and effect between their movements and what sounds come out. Seeing yourself on video with the keypoint and link overlay will hopefully give useful hints that there is something going on in the room. Otherwise, there is nothing obvious to attract the attention of people and to raise their curiosity. During real life testing, when

running MISK-Moves without showing the output video, it has been ignored since there are other things around in the room that attracts attention and interest, rather than something that cannot be seen.

MISK-Moves is already a working instrument, but it can work better in the future with a hardware upgrade. If the latency could be reduced, the instrument would feel more responsive and probably be more enjoyable to play. It is fully possible to expand on its functionality if there is a need or interest to further develop the application as a future MISK project, or as part of another project entirely.

References

[1] Allmäna arvsfonden. *MISK - Musik, Interaktiv design, Sinnesstimulering, Kvalitet*. https://www.arvsfonden.se/projekt/alla-projekt/projektsidor/misk---musik-interaktiv-design-sinnesstimulering-kvalitet. Accessed: 2023-03-21. Jan. 2021.

[2] Anaconda Inc. *Understanding Conda and Pip*. https://www.anaconda.com/blog/understanding-conda-and-pip. Accessed: 2023-04-03. 2023.

[3] T. Attawad. *Python-osc*. https://github.com/attwad/python-osc. Accessed: 2023-03-28. Jan. 2023.

[4] COCO Consortium. *COCO 2020 Keypoint Detection Task*. https://cocodataset.org/#keypoints-2020. Accessed: 2023-03-22. Mar. 2020.

[5] Cycling '74. *What is Max?* https://cycling74.com/products/max. Accessed: 2023-03-31. 2023.

[6] Data Base Camp. *Machine Learning*. https://databasecamp.de/en/machine-learning. Accessed: 2023-03-31. 2023.

[7] Data Base Camp. *ResNet: Residual Neural Networks – easily explained!* https://databasecamp.de/en/ml/resnet-en. Accessed: 2023-03-31. 2023.

[8] Data Base Camp. *What are Tensors in Machine Learning?* https://databasecamp.de/en/python-coding/tensors. Accessed: 2023-03-31. 2023.

[9] Data Base Camp. *What are Tensors in Machine Learning?* https://databasecamp.de/en/python-coding/tensors. Accessed: 2023-03-31. 2023.

[10] Data Base Camp. *What is Deep Learning?* https://databasecamp.de/en/ml/deep-learning-en. Accessed: 2023-03-31. 2023.

[11] Docker Inc. *Docker overview*. https://docs.docker.com/get-started/overview/. Accessed: 2023-03-28. 2023.

[12] Eldorado resurscenter. *Eldorado resurscenter – om oss*. https://goteborg.se/wps/portal/enhetssida/eldorado/om-oss. Accessed: 2023-03-22. Mar. 2023.

[13] Föreningen Furuboda. *Furuboda*. https://furuboda.org/. Accessed: 2023-03-22. Mar. 2023.

[14] D. Franklin. *Hello AI World*. https://github.com/dusty-nv/jetson-inference. Accessed: 2023-03-24. 2019.

[15] D. Franklin. *Jetson Nano Brings AI Computing to Everyone*. https://developer.nvidia.com/blog/jetson-nano-ai-computing/. Accessed: 2023-03-24. 2019.

[16] V. Gupta and Kukil. *YOLOv7 Pose vs MediaPipe in Human Pose Estimation*. https://learnopencv.com/yolov7-pose-vs-mediapipe-in-human-pose-estimation/. Accessed: 2023-04-03. 2023.

[17] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[18] B. Jönsson and M. Nygren. "Certec – en udda fågel". In: S. Mårtensson et al. (Eds.). *50 år med LTH: en fingervisning om teknik*. Lunds universitet, Lunds tekniska högskola, 2011, pp. 278–289. ISBN: 9789197975605. URL: https://books.google.se/books?id=2lE7MwEACAAJ.

[19] R. Leadbetter. *Nintendo Switch CPU and GPU clock speeds revealed*. https://www.eurogamer.net/digitalfoundry-2016-nintendo-switch-spec-analysis. Accessed: 2023-03-27. Dec. 2016.

[20] MIPI Alliance. *MIPI CSI-2*. https://www.mipi.org/specifications/csi-2. Accessed: 2023-03-27. Mar. 2023.

[21] V. Nair and G. E. Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[22] D. Norman. *The design of everyday things: Revised and expanded edition*. Basic books, 2013.

[23] NVIDIA. *GeForce RTX 4090 – Beyond Fast*. https://www.nvidia.com/sv-se/geforce/graphics-cards/40-series/rtx-4090/. Accessed: 2023-05-15. 2023.

[24] NVIDIA. *Jetson Benchmarks*. https://developer.nvidia.com/embedded/jetson-benchmarks. Accessed: 2023-05-02. 2023.

[25] NVIDIA. *Welcome to the DeepStream Documentation*. https://docs.nvidia.com/metropolis/deepstream/dev-guide/text/DS_Overview.html. Accessed: 2023-03-24. 2023.

[26] H. Obinata. *YOLOv8 on the Jetson Nano*. https://i7y.org/en/yolov8-on-jetson-nano/. Accessed: 2023-04-04. 2023.

[27] A. Osthoff. *Apple M2 Pro and M2 Max analysis – GPU is more efficient, the CPU not always*. https://www.notebookcheck.net/Apple-M2-Pro-and-M2-Max-analysis-GPU-is-more-efficient-the-CPU-not-always.699140.0.html. Accessed: 2023-05-15. 2023.

[28] Raspberry Pi Ltd. *About the Camera Modules*. https://www.raspberrypi.com/documentation/accessories/camera.html. Accessed: 2023-03-27. Mar. 2023.

[29] Ultralytics. *Ultralytics YOLOv8*. https://github.com/ultralytics/ultralytics. Accessed: 2023-04-03. 2023.

[30] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao. "YOLOv7: trainable bag-of-freebies sets new state-of-the-art for real-time object detectors". *arXiv preprint arXiv:2207.02696* (2022).

[31] K.-Y. Wong. *Official YOLOv7*. https://github.com/WongKinYiu/yolov7. Accessed: 2023-04-03. 2022.

[32] M. Wright. *OpenSoundControl.org*. https://ccrma.stanford.edu/groups/osc/index.html. Accessed: 2023-03-28. Aug. 2021.

A Appendix

A.1 UML diagram

Figure A.1 shows the UML diagram for the application. Note that classes from the external classes used are not fully modelled in the diagram, and member methods and attributes are omitted. That goes for most relations, inheritances, etc. as well. In the Jetson Inference package, to illustrate what the relations regarding the ObjectPose and Keypoint structs are, it is shown in which class or struct they are incorporated as inner classes.

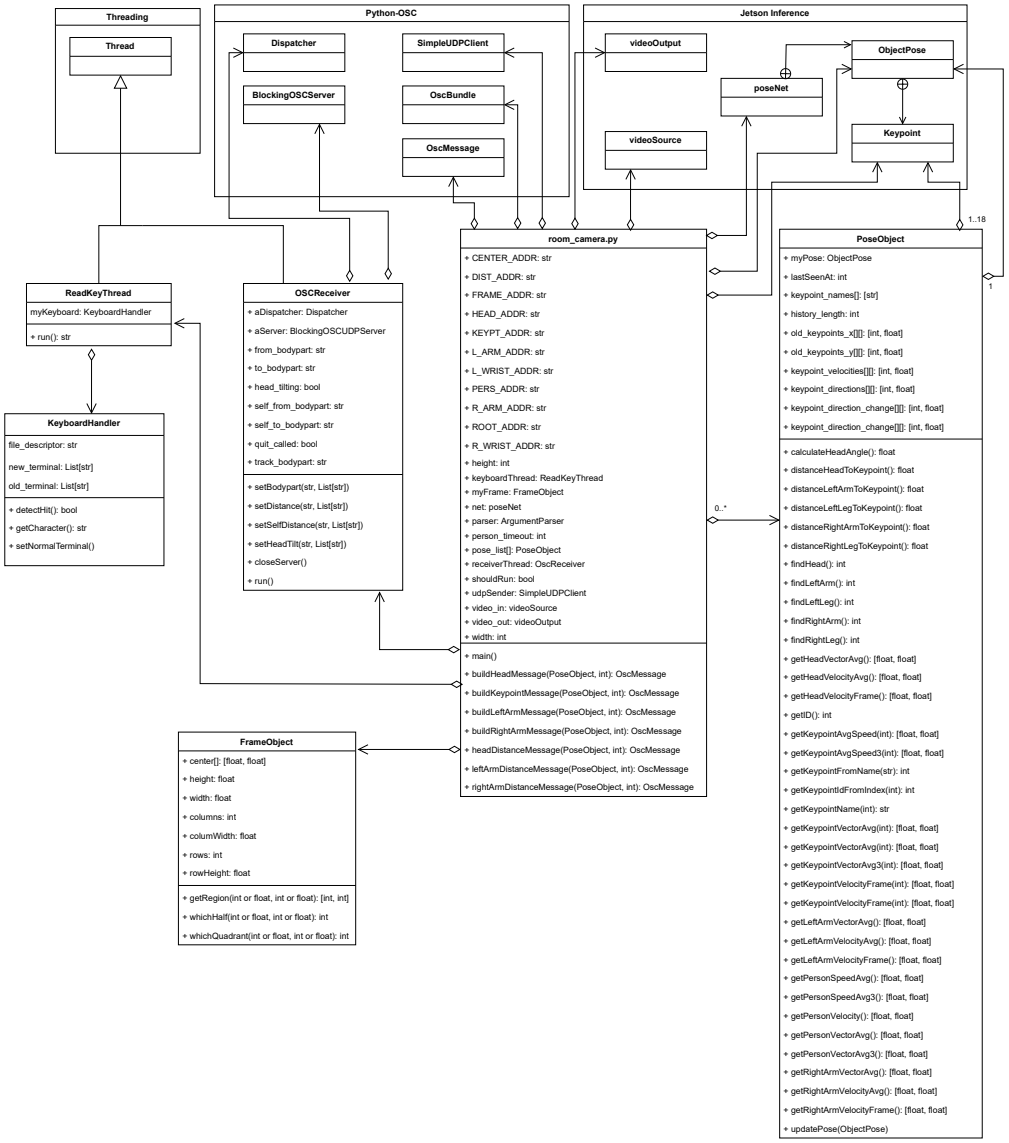

Figure A.1 The UML diagram for the project. Note that classes made by other developers and corporations are grouped together in the respective package that provides them.