FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Tactile Matrix Box
Student:	Bc. Jiří Šebele
Supervisor:	Ing. Radek Richtr, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

Cílem práce je navrhnout a následně vytvořit plaformu pro tvorbu interaktivních tangible aplikací s využitím projekčního mapování, detekce markerů a rozpoznávání 3D objektů a to pro laboratoř GLab, kde bude přípravek umístěn.

1) Proveďte rešerši
 a) v oblasti tangible rozhraní,
 b) uživatelských paradigmat ve fyzické interakci,
 c) způsobů zobrazení a projekčního mapování,
 d) metod pro rozpoznávání 2D markerů a 3D objektů z hloubkové mapy (UI, příznakové metody, atd.).
2) Analyzujte existující řešení (Tangible Matrix z MIT), jak jeho hardware, tak použité metody pro rozpoznávání a sledování objektů.
3) Prozkoumejte možnosti modularity a multiplexování hardware.
4) Navrhněte a sestavte hardware, vytvořte podklady potřebné pro výrobu.
5) Navrhněte a implementujte
 a) vhodné rozpoznávání objektů v pracovním prostoru,
 b) framework k tvorbě aplikací a vytvořte elektronickou dokumentaci k jeho užití,
 c) alespoň jednu jednoduchou aplikaci demonstrující užití frameworku.

Electronically approved by Ing. Karel Klouda, Ph.D. on 10 February 2022 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Tactile Matrix Box

Bc. Jiří Šebele

Department of Applied Mathematics

Supervisor: Ing. Radek Richtr, Ph.D.

May 4, 2022

Acknowledgements

I would like to thank my friend David for his relentless support throughout 8 grueling years of a university trying to push knowledge into my head. I would like to thank my partner for her moral support as well as impeccable reading skills she used to correct my sloppy writing. Last but not least, I would like to thank my wonderful supervisor who has seen me through my first year at FIT as well as my last.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on May 4, 2022 . . . . . .

Czech Technical University in Prague

Faculty of Information Technology

Citation of this thesis

Šebele, Jiří. *Tactile Matrix Box*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstrakt

V této práci prozkoumáváme vznikající oblast tangible rozhraní, zvláště tangible povrchů. Zkoumáme aktuální techniky pro rozpoznávání obrazu s pomocí příznaků, jakož i aktuální aplikace umělé inteligence. Naším cílem je zlepšit design MIT Tactile Matrix a vyvinout jednoduché SDK, které studentům umožní snadno vytvářet nové tangible aplikace. Zlepšujeme flexibilitu konstrukce abychom ji umožnili studentům do budoucna upravovat, a rozšiřujeme funkční dosah zařízení pomocí ethernetové komunikace. Diskutujeme možnosti budoucích aplikací a ohlížíme se za několika případovými studiemi.

Klíčová slova GLab, tangible rozhraní, MIT Tactile Matrix, ethernet, příznakové metody, umělá inteligence

Abstract

In this work, we explore the emerging field of tangible interactions, especially tangible surfaces. We research the state of the art in fiducial marker tracking, as well as current artificial intelligence approaches. Our goal is to improve upon the MIT Tactile Matrix design and to develop a simple to use SDK which students can use to develop novel tangible applications easily. We make the construction more flexible to accommodate the needs of the students, and we expand the functional range of the MIT Tactile Matrix by using ethernet communication. We discuss possible future applications and look back at case studies of the device's deployment.

Keywords GLab, tangible interfaces, MIT Tactile Matrix, ethernet, fiducial marker tracking, artificial intelligence

Contents

List of Figures

Chapter 1

Introduction

Ever since the invention of the computer, one of the greatest hurdles has been effective exchange of information between a computer and a human brain. While communication between humans and computers is a field we've somewhat perfected over the past decades, it still leaves a lot to be desired.

With the arrival of the smartphone came capacitive touchscreens – a versatile input method in more than one way. They are less prone to mechanical failure, cheaper and allow iteration of interface elements without having to modify the device. And since then, touchscreens have been silently taking over the majority of user interfaces, from fridges to cars.

This homogenization of user interfaces has however lead to a curious psychological effect. We feel distanced from our content, forever separated by a glass wall from the things we are trying to touch. This sentiment is evidenced by the recent return of interest in analog media. Be it film cameras, vinyl records or paper notebooks, it's clear that we as a society are trying to stop the touchscreens from encroaching on our lives.

In this work, we will explore the past and present landscape of tangible user interfaces and attempt to design a tangible surface device for the emerging graphics laboratory of FIT CTU. We will be doing this in hopes of easing the exploration of tangible user interface paradigms and applications for future students and helping them design novel tangible experiences.

Chapter 2

Research

In this chapter, we will be taking a look at the concept of tangible media as described by professor Hiroshi Ishii of MIT's Tangible Media Group[1]. We will also take a look at some accessibility issues present in current digital interfaces. Finally, we will research existing projects to better understand their design and implementation.

2.1 Tangible media

According to the Tangible Media Group's website, "the Tangible Media Group, led by Professor Hiroshi Ishii, pursues the vision of Tangible Bits & Radical Atoms to seamlessly couple the dual worlds of bits and atoms by giving dynamic physical form to digital information and computation." [1]. This quote gives us insight into the main motivation behind the Tangible Media Group's work: **Human-Computer interactions**. This vision of "Tangible Bits & Radical Atoms" groups user interfaces into three distinct categories:

- **GUIs** – "painted bits",
- **TUIs** – "tangible bits",
- and **Radical Atoms**.

Graphical user interfaces (GUIs) only allow users to see the digital information through a screen. We can interact with it using a mouse or a keyboard, but we are separated from it by the transition from physical to digital. Ishii

[1] `https://tangible.media.mit.edu/`

likens this to the surface of water acting as a membrane. Tangible user interfaces (TUIs) break this barrier, with a portion of the digital information emerging into the realm of the physical. This is likened to an iceberg poking through the water with the majority of the ice still hiding underwater.

As identified in [2], even tangible user interfaces have limitations in that only a portion of the digital information is represented in the real world. We can manipulate the position, size or color of the tangible elements, but they can still quickly enter a state that's inconsistent with the digital information. The "radical atoms" vision described in [2] seeks to remedy this through the use of hypothetical smart materials, which allow us to represent the digital information clearly and completely, thus bringing the physical object on par with the digital one.

The core concept is that by allowing actuation of tangibles, we can provide feedback to the user and even constrain the physical object through the digital information. We can see glimmers of this idea for example in motorized faders used by some audio equipment, which can be manipulated by the user (for example to change the volume of the audio), but also by the system (for example when switching to a preset). Some models can also force the user to manipulate the sliders in sync so that they never reach an inconsistent state.

A more advanced example of this would be Kevin Lyangh's blog post[2] detailing his attempts at replicating the actuation of magnets using magnetic fields generated by PCB traces. This shows that the actuation of very small "bits" of matter is not only feasible, but easily reproducible.

2.2 Accessibility

Since touchscreens have begun emerging, they have been steadily gaining ground in the digital media landscape. Touchscreens have proven to be extremely versatile, flexible and cost-effective solution for many devices, from phones to automotive applications. However, they can be challenging to use for people with visual or motor impairments[3]. Tangible interfaces can be a solution for this problem, as they often require less precision, less control and less effort.

[2] `https://kevinlynagh.com/pcb-stepper/`

With accessories like the Xbox Adaptive Controller[3]it's now easier than ever to create creative new interfaces that allow users with disabilities to enjoy unrestricted interactions. However, that is not the only benefit of such accomodations. Making digital content more accessible to people with disabilities often makes it friendlier and more educational for everyone – especially children. There is a growing trend of tangible interfaces in museum exhibitions[4][5], where tangibility helps bring the content closer to the viewer.

2.3 Previous Work

This section will present existing tangible devices, either appearing in published academic work or commercially available on the market. We will also explore some DIY projects, as the recent rise of commercially available consumer-grade electronics platforms like Arduino, as well as 3D printing made prototyping physical interfaces much more affordable.

Although we will briefly touch on tangible devices in general, we will focus mainly on tangible *surfaces*, since the device we will be designing will be surface-oriented as well. We also won't be spending much time on discussing devices based on simple capacitive touchscreens and other technologies with limited ability for tracking physical objects. We will attempt to glean some insight from these projects – such as which paradigms seem to be popular and which aren't – and discuss their design and implementation.

2.3.1 Reactable

Reactable is an electronic music instrument developed by Reactable Systems SL – a company founded in Barcelona in 2003[6]. Its primary mode of operation is connecting together various components that together then generate sound. These components are created by placing physical blocks on the surface of the device. It will also detect fingers placed on the projection surface, allowing for more fine-grained control of the created sound. It's designed to emphasize collaboration between users in museums or hotel lobbies, as well as to provide an interesting instrument for live artists performing on stage.

[3] `https://www.xbox.com/en-US/accessories/controllers/xbox-adaptive-controller`

Figure 2.1: Reactable[7] for exhibitions (left) and for artists (right).

Internally, Reactable uses reacTIVision, which is an open-source framework we will explore in section 3.2. The similarity in naming is no coincidence, as reacTIVision has been developed for the Reactable synthesizer by the same team. The music synthesis engine is not a part of reacTIVision but is available separately, also with an open-source license. The tracking is performed in the infrared spectrum, which allows for projection and tracking of objects from the same side.

Construction-wise, the device is very straightforward, with just a projector, camera and IR illuminator placed under a small semi-transparent surface. The blocks appear to be manufactured from a clear plastic, but the markers themselves are opaque. They are designed to be a visually appealing "amoeba" shape, which reflects the consideration of appearance in the design of the device.

2.3.2 Microsoft PixelSense

Microsoft PixelSense is a computing platform for interactive surfaces developed by Microsoft and initially released in the year 2008. It used to be called Microsoft Surface, before that name was repurposed for a line of consumer-grade desktop and laptop computers. Even today, the official PixelSense website[4] is no longer operational and instead redirects to the official Microsoft website, specifically the portion dedicated to the Surface family. As of the time of writing, the product line seems to be largely dead, while not officially discontinued.

[4] `https://pixelsense.com/`

In its original incarnation, the device is very similar in design to other devices like the Reactable. It features an array of infrared cameras, a projector and an infrared illuminator. It can detect and track objects placed on the surface, touches and gestures, as well as "actual unique objects that have identification tags similar to bar codes" [8]. This indicates that Microsoft was possibly intending to expand the device to include more tangible interactions[5] – an area in which they have also done extensive research[9].

Figure 2.2: Microsoft PixelSense 1.0 (left) and its internal layout (right)[6].

Many parts of the design fall victim to the age of the design, such as the user interface or the resolution. The last device released was the Samsung SUR40[7]in 2012. While the device itself wasn't a big success for Microsoft, it has stood at the forefront of the tangible surface devices and spawned several patents for a similar design[10].

2.3.3 SmartKnob

SmartKnob[11][12] is an open-source project providing assembly instructions, PCB schematics and firmware for a simple rotary input knob with haptic response. It provides four main functions:

- a rotary encoder,
- a simple button,
- an LCD,
- and an LED color backlight.

[5]Evidenced by some promotional material at the time, showing a possibility of connecting a phone to the device by placing it on the screen.

[6]`https://www.amazon.co.uk/dp/B00JRCWRCQ`

[7]`https://www.samsung.com/us/business/support/owners/product/samsung-sur40-with-microsoft-pixelsense-sur40/`

Figure 2.3: The SmartKnob in use[12].

The central component of the knob is a brushless DC motor with a hollow shaft onto which the knob is mounted. This motor serves to provide haptic feedback as well as exert force onto the knob when necessary. The rotary motion is detected with a single degree precision using a magnetic rotary encoder chip. The device doesn't feature a physical button, but instead opts to use a strain gauge to detect a press and provides haptic feedback via the motor. The round LCD is mounted on top of the knob (but doesn't rotate with the knob) with its wiring connected through the shaft of the motor.

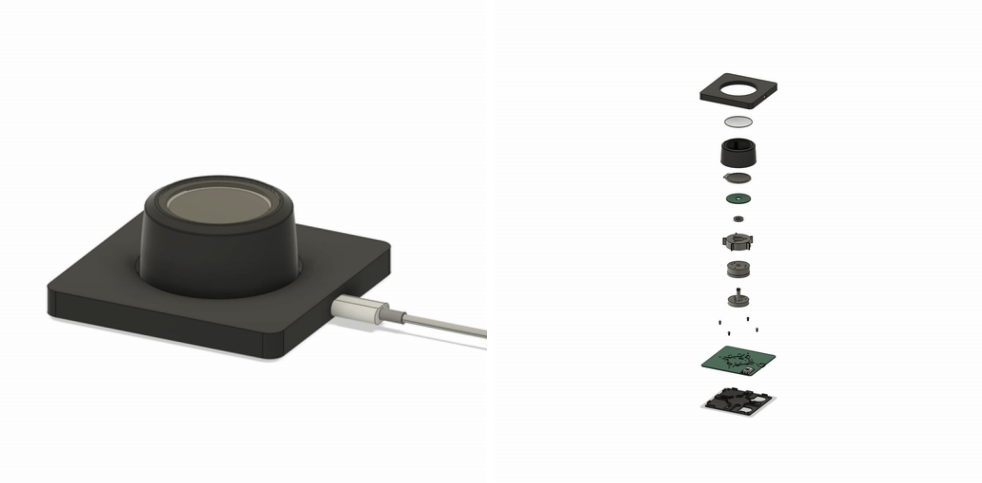


Figure 2.4: CAD model of assembled and disassembled states respectively[11].

The key feature of this device is the ability to use the motor to provide haptic feedback in several forms. The first form is "clicking", either as small clicks indicating steps on a scale (akin to a standard computer mouse wheel)

or larger clicks, such as the one imitating a press of a button. The second form are virtual detents[8], which can serve either as hard end-stops for rotation in cases where the range of motion should be limited (for example a thermostat), or as virtual snap points (such as in an on/off switch). This makes the device feel as if it were spring-loaded.

2.4 Tactile Matrix

The Tactile Interactive Matrix[13][14] (henceforth referred to as "tactile matrix" or "the device") is a device described as a "System for Real-time Digital Reconstruction and 3D Projection-Mapping of Arbitrarily Many Tagged Physical Objects" in its provisional patent[13]. While deceptively complex, the principal function loop of the device is quite simple:

1. extracting the layout of a matrix of physical elements from visual input,
2. digitally reconstructing the form and metadata of said elements,
3. generating real-time visualisation of the analysis,
4. displaying it to the user via projection mapping or a display.

The novel feature of this device, according to the provisional patent[13], is that it can track an extremely large number of separate physical objects in real-time. This is due to the fact that the physical objects themselves are constrained in placement, i.e. they can only be placed on a grid. The task of tracking objects is thus drastically simplified, since the position of defining features – in this case color-coded regions on the bottoms of the objects – is known in advance and does not have to be detected from the image. This allows the device to track an extremely large number of objects, without the need for a large amount of processing power typically needed when scanning for markers with unknown positions.

As we can see in Figure 2.5, the tactile matrix is made up of several components, the most prominent being the table itself, namely the large transparent surface. In the table there is a *camera* capturing the bottom of the surface and a *lamp* providing the light for this camera. This camera capture is processed on a *computer*, which then displays appropriate data reacting to the

[8] A mechanical or magnetic means to resist or arrest the rotation of a wheel
[9] `https://ira.mit.edu/sdk`

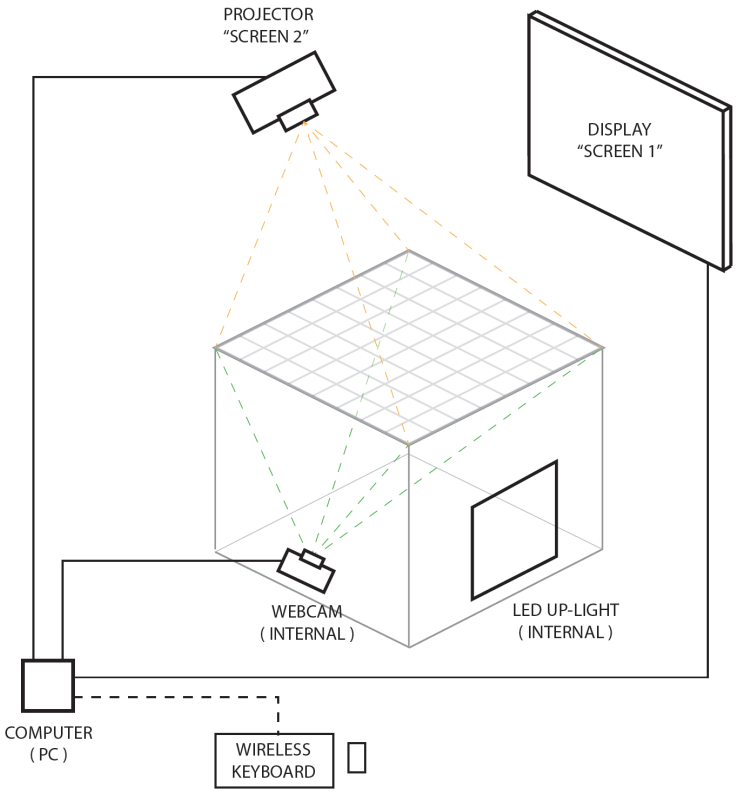


Figure 2.5: Electronic components of the tactile matrix[9].

content of the surface on the *projector* and/or the *display*. The most notable defining features of this design are:

- internal camera tracking from below the surface,
- external projection mapping from above the surface,
- transparent surface,
- requirement for internal lighting in the visible spectrum.

The projector does not seem to be affixed to the device itself in any of the prototypes, but is instead mounted in a separate place above the device. In some versions of the device, the camera is looking down onto a mirror reflecting the table. This is done to allow for cameras with narrower field of view that would be required were the camera mounted directly on the bottom of the table.

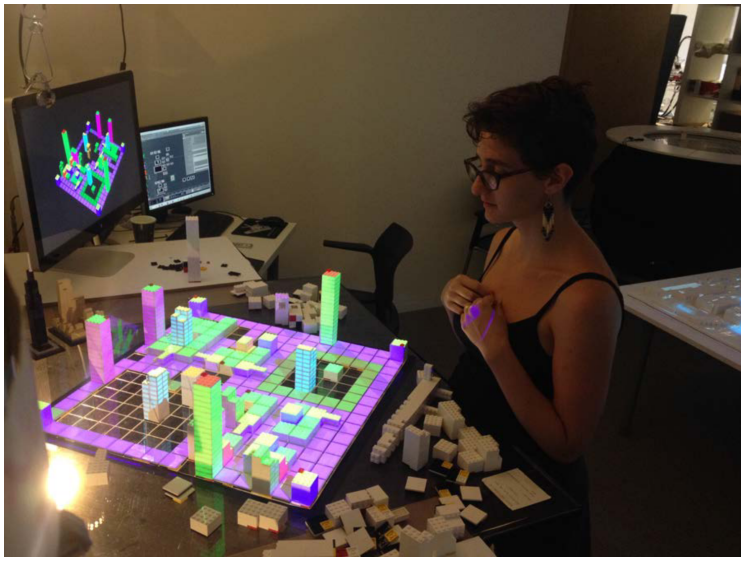


Figure 2.6: The tactile matrix in use[13].

In the published construction drawings[10] (seen in Figure 2.7), the device is constructed from metal sections held together by screws. The sides are then mounted onto this frame (also with screws). The surface, in both cases cut from clear acrylic with laser-cut grid sheets on top, rests on top of the sections without anything but the sections holding it in place. In another design (seen in Figure 2.7), the table is made of two disjoint welded stainless steel frames with the surface overhanging them.

The tactile matrix was originally created for the CityScope project (closer explored in subsection 2.4.2), but has since been adopted for use in other projects as well.

2.4.1 Colortizer

The central part of the tactile matrix SDK is an application called Colortizer. Its purpose is to provide a simple interface that allows the user to define a rectilinear[11] area in the camera input, from which the application then recognizes color-coded objects into a matrix containing their positions, rotations and IDs. The crucial difference to other approaches is that it only recognizes very simple markers placed in a grid, which makes the recognition process

[10] `https://ira.mit.edu/s/TactileMatrixHDK.zip`
[11] Contained by, consisting of, or moving in a straight line or lines

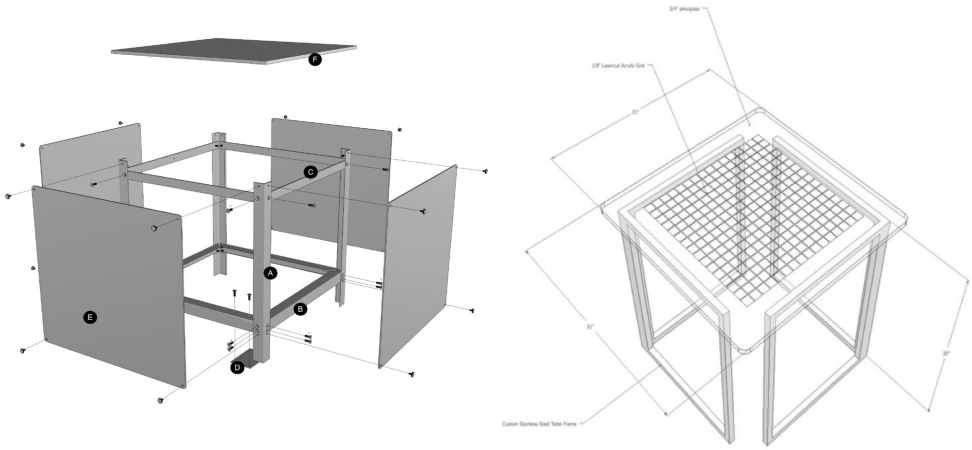


Figure 2.7: Technical drawings of the construction.

as simple as averaging the color of a small area of pixels. The application then communicates this data over UDP to a client application, for example CityScope platform.

It is developed solely by Ira Winder in Processing[12] and is published on GitHub[15] under the MIT License. As of this writing, the last activity in the `master` branch of the repository was on December 20th, 2017.

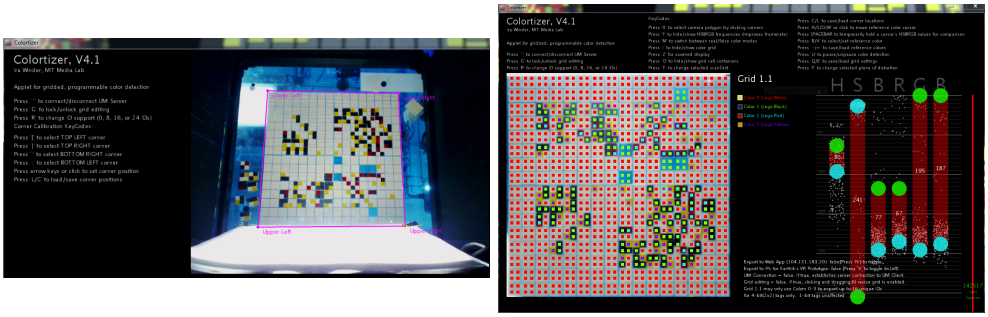


Figure 2.8: Screenshots from the Colortizer application[15].

2.4.2 CityScope

One of the main uses for which the tactile matrix is intended is the CityScope[13] project – a set of tools developed by City Science researches on MIT. The tools vary from interactive applications for collaborative design and analysis

[12] `https://processing.org/`
[13] `https://www.media.mit.edu/projects/cityscope/overview/`

to simulations of traffic flow, human movement and the environmental impact of various interventions carried out by the city.

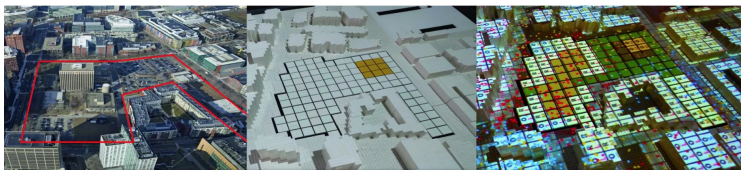


Figure 2.9: The process of converting a city block to a grid.

Some of these tools use the tactile matrix as a tangible user interface for laying out city blocks, manipulate traffic nodes and visualize the impact of these actions in real-time. This helps drive interest in city planning, helps the general public collaborate on urban planning and helps researchers to understand the impact of development in the city. One such project is the CityMatrix[16], which help non-experts make qualified decisions with the help of artificial intelligence system for evidence-based decision support.

Figure 2.10: The CityMatrix in use.

2.5 Projection mapping

Projection mapping, sometimes also referred to as video mapping, describes the technique of using projectors to map the projected image onto a real-world objects. The objects are usually irregular and can be as simple as a cube or as complex as a whole building. This creates the illusion of the irregular surface itself functioning as a screen. It is a technique that has seen a steady rise in popularity during recent years, appearing on such events as the Signal Festival[14] held in Prague.

[14] `https://www.signalfestival.com/en/`

This is most often accomplished in one of two ways: manually, or automatically (but most often a combination of both). Manual mapping involves manual distorting of the input image to fit the projection surface. This is usually done using software like Resolume Area[15] or TouchDesigner[16]. In contrast, automatic mapping is done by a computer program that uses a camera to capture the scene in relation to the projector and automatically creates a 3D model of the surface. This is a technique employed for example by the LightAct[17] program. This 3D model of the scene can also be acquired manually using various photogrammetry methods.

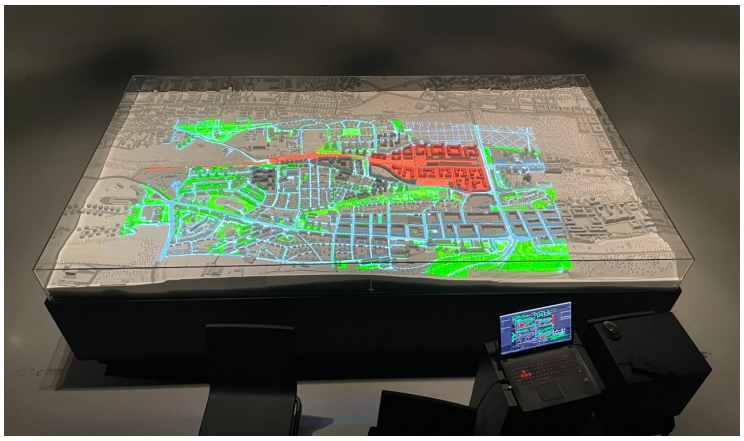


Figure 2.11: Photo of projection mapping done at CAMP[18]in late 2021.

[15] https://resolume.com/
[16] https://derivative.ca/
[17] https://lightact.io/
[18] https://www.campuj.online/

Chapter 3

Object tracking

In this chapter, we will explore object tracking and various methods we can use to extract information about the state of the world from visual data. We'll try to define some metrics on which we will judge the solutions and finally pick out one that we will use for our reference implementation in chapter 7. We do this in a separate chapter, as we refer to our specific requirements quite a bit, making the chapter lay halfway between research and analysis.

Let's take a look at the concept of object tracking and what it means in different contexts. We will be using the term "object tracking" to refer to the process of extracting data about the states of objects in a scene from a visual capture of the scene. However, even then object tracking is a very diverse problem space. There are many ways to capture the scene, such as a 2D RGB camera image, a monochromatic depth map or a LiDAR point cloud.

This presents us with a large amount of solutions with varying properties. There is several key characteristics of our problem that we need to identify before choosing a tracking solution:

- How are we capturing our scene?
- How fast do we need to track?
- Are we detecting one or multiple objects?
- Do we want to maintain object persistence?
- What properties of the object do we need to extract?
- Do we know the appearance of the tracked object in advance?

For our use case – being tracking objects on a flat surface with a camera – these questions luckily have fairly clear-cut answers. We need to be tracking at *super-realtime speeds*. That means at least as fast as the refresh rate of our visual output and with an ideal latency of less than 1 frame. We will definitely be detecting *multiple* objects, and we will want to maintain object persistence (meaning keep track of which object is which between consequent frames). We need to extract at least the *position* and *rotation* of the object.

The appearance requirement, however, is slightly more tricky and will heavily depend on the application the user will be trying to develop. Below, we'll explore several possible use cases. We will focus mostly on fiducial tracking systems, which extract information with the help of markers of known appearance. However, we will also explore the possibility of using current artificial intelligence approaches for computer vision, as well as the problem of recognizing objects from various 3D representations of the scene.

(a) reacTIVision[17] (b) BullsEye[18] (c) ArUco[19]

Figure 3.1: Various kinds of fiducial markers.

3.1 Machine Learning

In recent years, machine learning has become a very popular technique to solve many machine perception problems, usually with the use of so-called *neural networks*. In a nutshell, the process involves training a *model* on a set of data by having the model process points from the data set and then adjusting its parameters so the output matches. The data is usually generated and/or annotated by a human.

One such tool is the MediaPipe[20] framework – a framework for building machine learning pipelines that can be deployed on a variety of platforms. It presents a series of pre-trained models for various tasks, such as face detection,

pose estimation and object tracking. These models are wrapped in a library available for a variety of languages, such as C++, Python and JavaScript. Note that not every model may be available for every language, as per the feature matrix on the MediaPipe website[19].

Of particular interest to us is the **KNIFT**[20] template-based feature matching solution. That is, we can use it to detect previously seen patterns in images in real-time, for example to detect traffic signs in dashcam footage, or to detect banknotes and their respective values. The model will also output the image segment that matched the template. This can be an extremely efficient solution if we wanted to detect objects users might have on them, like business cards, trading cards or banknotes. It is, however, very inefficient for fiducial tracking, since it provides much less precision than other methods. As of the time of writing, **KNIFT** is only available on Android[21].

Another relevant solution is MediaPipe's **Objectron**, which offers real-time 3D object detection and pose estimation in 2D images. This isn't a good fit for tracking precise object positions from below, but it can be very efficient at detecting objects from above. A segmentation mask can also be predicted for each object, allowing us to extract the object from the image, or visually modify it. As of the time of writing, **Objectron** is available for all three programming languages mentioned above.

(a) KNIFT (b) Objectron

Figure 3.2: Example outputs from MediaPipe solutions. [21]

[19] `https://google.github.io/mediapipe/`
[20] Which stands for "Keypoint Neural Invariant Feature Transform".
[21] `https://www.android.com/`

Last of the solutions offered by MediaPipe we will look at is the **Box Tracking**. This model allows detection of potentially overlapping bounding boxes of objects in an image. It also tracks object instances across frames, providing stable object IDs. It doesn't provide any pose information, nor does it provide information about the object's orientation, making it effective only for presence tracking. As of the time of writing, **Box Tracking** is available for mobile platforms and C++.

But what if we wanted to track objects from a depth map and not constrain ourselves to a 2D image? With devices like the Microsoft Kinect[22] or Intel RealSense[23], as well as many modern mobile devices and computer vision platforms, acquiring a depth map is fairly straightforward. We might also want to process a point cloud acquired by a LiDAR.

Luckily, this is also an area where machine learning performs very well with multiple models having emerged in the past years. One of those is **VoxelNet**[22], which detects objects in 3D point clouds. Another example is the **Complex-YOLO**[23] model, which can estimate multi-class 3D bounding boxes and claims to outperform the **VoxelNet** and other current state-of-the-art methods in the **KITTI**[24][25] car detection benchmark. Both of these solutions are designed for highly sparse point clouds obtained via LiDAR devices and clearly intended for automotive use.

Due to their nature, machine learning models are very hard to predict and reason about, and may produce unexpected results. They also tend to be fairly computationally expensive compared to procedural approaches. This can lead to very impressive results as with the **Objectron** network, but is not always suitable for situations where users come from the general public, as any errors (such as object misclassification) that occur likely won't be intuitive.

3.2 ReacTIVision

The reacTIVision project describes itself as a computer-vision framework for table-based tangible interaction [26][27]. It is a set of open-source libraries and applications that allow users to create tangible interaction systems. ReacTIVision also defines the TUIO protocol discussed in subsection 4.1.3. It

[22] `https://developer.microsoft.com/en-us/windows/kinect/`
[23] `https://www.intelrealsense.com/`

was originally developed for the Reactable product we've discussed in subsection 2.3.1. The code base is written in portable C++ code and builds for Windows, macOS[24] and Linux.

Its fiducial markers look very stylish with their "amoeba" shape. The geometry of the markers was generated using a genetic algorithm optimizing for footprint size as well as location and rotation accuracy. The markers represent a tree with their alternating white and black areas, all of which have 19 leaf nodes and maximum depth of 2. The location is determined as the centroid of all leaf nodes (thus providing sub-pixel accuracy) and the orientation is determined by the vector from the location to the centroid of black leaves only.

(a) Default (b) Small (c) Mini

Figure 3.3: Examples form different reacTIVision fiducial sets.

The application itself is somewhat bare, with configuration done mainly through an XML configuration file and few parameters being adjustable directly in the application. It allows for manual distortion calibration using a grid superimposed onto the captured image. It performs reasonably well on any modern hardware, despite the algorithm being implemented on the CPU. The software support for the TUIO protocol client-side portion is quite extensive with an implementation being available for most major programming languages. There's also several supporting applications, such as TUIO 1.1 Simulator[25], that allow for testing the TUIO protocol without a camera.

[24] Formerly known as Mac OS X.

[25] `https://github.com/mkalten/TUIO11_Simulator`

3.3 BullsEye

The BullsEye fiducial marker tracking system is modeled after reacTIVision and boasts significantly increased speed and accuracy, specifically that "BullsEye provides sub-pixel accuracy down to a tenth of a pixel, which is a significant improvement compared to the commonly used reacTIVision software" [28]. It is implemented using Java, which means that it requires the Java runtime environment to be installed on the computer. It's also designed with parallelization in mind, which enables the tracking algorithm to run on the GPU, offering significant performance benefits compared to CPU-based approaches.

Unlike reacTIVision, BullsEye features a graphical user interface for configuration and calibration. The user can compensate for camera distortion (either manually or automatically), calibrate color and lightness of the captured image, as well as set up finger tracking. It can also handle multi-camera systems configured within the interface.

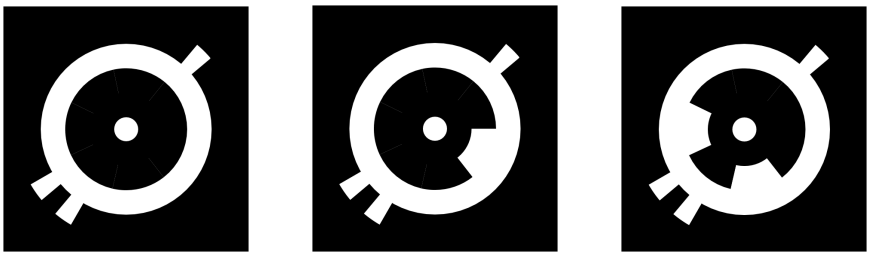


Figure 3.4: BullsEye markers with varying IDs.

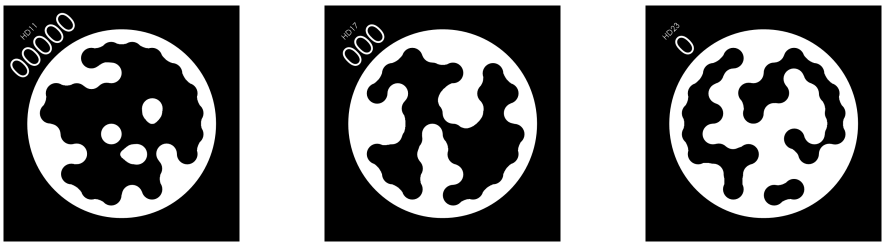
The design of the markers themselves is very different from reacTIVision with a quite distinct circular shape. This gives the BullsEye marker a unique appearance. Unlike the reacTIVision markers, they are quite similar in appearance, making them difficult to distinguish from each other by the naked eye. This requires us to make the objects otherwise distinguishable to prevent confusion during usage.

3.4 STag

The STag fiducial marker system[29] is designed to provide a stable pose estimation. Pose estimation, as compared to the previous two systems, means estimating the position and orientation of a fiducial marker in 3D space, in-

stead of just on a planar surface. It is designed to be stable against jitter factors like changes in illumination, occlusion and camera noise.

Compared to other solutions it is very resilient against even heavy occlusion. While not particularly useful for our use case of tracking markers pressed against a surface, this is extremely useful for augmented reality applications such as placing and viewing 3D models in a real-time video stream. The markers themselves are distinct from other fiducials, with sets designed around the concept of hamming distance[26], as apparent in Figure 3.5.



(a) HD 11 (b) HD 17 (c) HD 23

Figure 3.5: STag fiducials with various hamming distances

3.5 ArUco

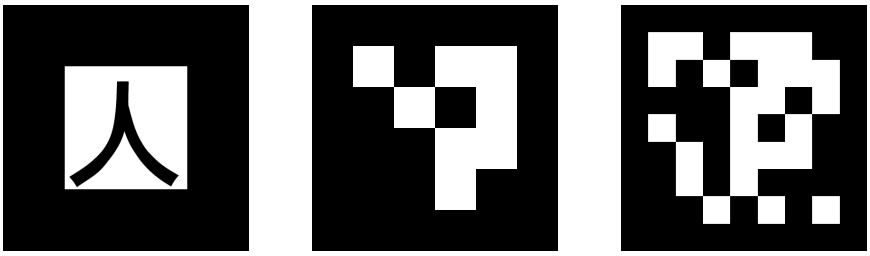
ArUco is a very lightweight and fast fiducial tracking system[30]. Similarly to STag, ArUco is designed to provide pose estimation in 3D. It supports various foreign marker types such as ARToolKit+, AprilTag and CHILITAGS, as well as providing its own marker type ARUCO with varying sizes and varying hamming distances between markers. The markers are square, which makes them trivial to print and cut out manually. At the time of writing, it's widely regarded as state-of-the-art for fiducial marker tracking on embedded devices.

There are currently two major implementations of ArUco fiducial marker detection. One is the ArUco C++ library[27] developed by the original group of researchers at the University of Cordoba. This implementation is currently the only one with support for tracking with discriminative correlation filters[31].

[26]Hamming distance is the minimum number of necessary misread bits required to match another entry in the marker dictionary.

[27]`http://www.uco.es/investiga/grupos/ava/node/26`

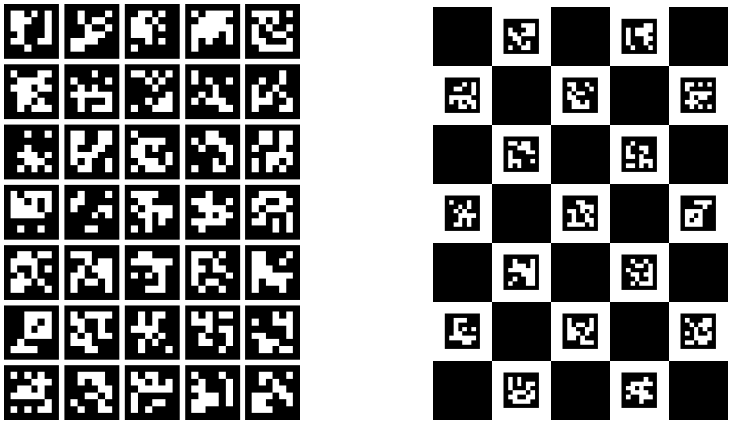
The other is included in the OpenCV library[28], which is based of the original ArUco library. It's available in both the C++ and Python versions of the library as of version 3.1.0.



(a) ARToolKit+ (b) ArUco 4x4 (c) ArUco 7x7

Figure 3.6: Example markers from dictionaries supported by ArUco.

This system is extremely vulnerable to occlusion of a single marker – the markers are basically only correctly identified when they are not occluded, including a white border around the black portion of the marker. This can be very problematic for augmented reality applications. For this reason, ArUco supports so-called marker sheets – a set of markers that are placed on a planar surface in known locations. This not only allows computing occlusion masks, but also significantly improves tracking precision. Such a marker sheet can be seen in Figure 3.7. Sometimes, this marker sheet is interspersed through a chessboard image. Such a sheet is referred to as ChArUco.



(a) ArUco Board (b) ChArUco

Figure 3.7: Example marker sheets.

[28] `http://opencv.org/`

As with many other fiducial tracking systems, ArUco too runs on the CPU. This does not pose much of a problem, however, as the algorithm is very fast. According to [30], it can achieve over 1000fps on 4K[29] images. This is well beyond the capabilities of most camera hardware and should serve most uses perfectly well.

During our tests, as well as work on the Hydropolis project discussed in chapter 9, the single biggest issue encountered with ArUco was the vulnerability to any blur – especially motion blur. While this can be slightly alleviated by sharpening, doing so amplifies the camera noise and increases tracking jitter. This issue is addressed in the subsequent work [31]. This paper, however, is at the preprint stage and has not been properly peer reviewed as of the time of writing.

3.6 Conclusion

In this chapter, we have seen multiple fiducial tracking systems. We also reviewed various machine learning methods and their application for object tracking and recognition of objects from depth data. One of the crucial properties of fiducial tracking compared to machine learning solutions is the ability to track an arbitrary amount of objects, which is not possible with all machine learning approaches. Another important facet is precision, with many fiducial tracking systems being able to track with sub-pixel precision thanks to the clever design of the markers.

For our device, we will choose to implement a tracker based on the ArUco library. While it's not optimized for tangible surfaces in the same way reacTIVision and BullsEye markers are, it's very fast and versatile. This will be important as we want to keep the computational resources required to track markers as low as possible. Support for ArUco is also included in OpenCV, which we'll be using in other stages of the tracking process like image acquisition and post-processing.

[29] 3840×2160 pixels.

Chapter 4

Analysis

In this chapter, we will analyze the problems we will be facing and the solutions we will be looking for. Afterwards, we will define the information we will need to communicate with other clients in the system and attempt to find the best way of transmitting that information to and from the device itself. This will include looking at existing technologies and protocols and how they're structured.

We will also look at the functional and non-functional requirements for the SDK. This will include figuring out what functionality we want to expose to the users using the SDK and in turn what input we want to communicate with the outside world. Finally, we will propose some enhancements to solve the issues we identified.

4.1 Networking

The device itself will have computing capabilities and could quite feasibly run any user applications on it. However, that is not a setup that could be described as ergonomic, especially considering that we want the device to be used in a university laboratory. Therefore, we will need to find a way to communicate with the device remotely.

In this section, we will look at what data we will need to transmit to and from the device, their bandwidth requirements and existing protocols available, as well as pick the physical interface we'll use to transmit the data.

4.1.1 Requirements

The device itself will need to report the tracking data, as well as exchange some configuration information with any clients. The tracking data will consist of a list of detected objects and their properties, including but not limited to:

- 2D position as coordinates in the range $[0, 1]$,
- a rotation in the range $[0, \tau)$,
- an object identifier.

Crucially, this information will be temporally limited, meaning each new value deprecates the previous one. This lends itself quite nicely to fault tolerance – if a packet is lost, the fault is completely recovered with the arrival of the next packet. It will also be crucial to discard any packets that arrive out of order or packets that are too old.

We can estimate the bandwidth necessary for transporting this information. Assuming usage of IEEE 745 32-bit floating point numbers[32], we'll require at least 2 floating-point numbers for the position and 1 for the rotation, as well as one 32-bit integer for the identifier, summing up to a total of 16 *bytes* per tracked object.

Assuming minimum object size of 2 cm by 2 cm no more than 2500 objects can be present on a $1\,\mathrm{m}^2$ of surface at once. Transmitting at 60 frames per second, we can then conclude that we will be transmitting no more than $2500 \times 60 \times 16$ *bytes* per second, or 2.4 *megabytes* per second, leaving us comfortably within the operating range of any modern network standard.

As for configuration, we'll assume any configuration values will be covered either by integers, floating-point numbers or strings. Configuration communication should be bidirectional, that is, allow setting and querying of values (though not necessarily both for some values). Possible configuration values might include:

- software version (read only),
- camera resolution and FPS (read/write),
- other camera parameters like exposure, gain, etc. (read/write),
- tracker parameters like marker dictionary (read/write),
- lighting state (write only).

Another big part of the data that needs to be transmitted is a video stream for the projector to display, as quite often we will neither want nor be able to generate this stream on the device itself. The quality requirements of this stream will be capped by the projector, but a reasonable optimum will be displaying a FullHD stream at 60Hz with an 8-bit color depth. As an aside, we might also want to include the ability to display a static image, to avoid having to turn static images into video streams just to display them.

As another major requirement, we will also want to transmit a video stream with camera data from the device to the client. This is useful for configuration purposes like camera calibration, but also for the client to be able to do custom processing on the camera data, like custom tracking that requires hardware capabilities beyond the computer included in the device. It will also allow the development of tracking systems without having to push code to the device with every iteration.

Last but not least, we need to focus on the latency of the video stream. Since we want our applications to run smoothly and feel responsive, we should aim to keep the latency of the video stream below 16 ms, or roughly the duration of 1 frame at 60 frames per second.

Let's now compare these two major requirements and their similarities and differences. They share some similarities:

- requirement for low latency,
- statelessness (low requirement for packet loss resilience).

But beyond that, the two kinds of data we'll be transmitting have very different characteristics. Namely, the tracking data transmission:

- doesn't require a lot of bandwidth and therefore any compression,
- should support variable data sizes (i.e. variable number of markers),
- needs to be flexible to accommodate changes in data structure.

Whereas the video data transmission:

- requires a lot of bandwidth and therefore compression,
- has low variance of data sizes (i.e. constant resolution),
- doesn't have a variable data structure.

We will therefore pursue different technologies for both of these requirements, to not constrain ourselves to a single protocol that would have to compromise on these fairly orthogonal characteristics. Furthermore, we will settle for using a wired connection instead of a wireless one. Not only does this provide a more stable reliable connection, but a wireless connection could prove to be extra unreliable given the presence of lots of other wireless equipment in the laboratory.

4.1.2 Open Sound Control

Open Sound Control – or OSC for short – is a simple, efficient, and extensible protocol for sending and receiving data. While it was originally designed to supersede MIDI[33] and be used mainly by sound synthesizers and other multimedia equipment, it has since been adopted for many other applications.

Compared to MIDI, OSC has been designed to run on modern network substrates with higher bandwidth (in the 10+ megabit/sec range[33]). It's therefore much more liberal with its use of this bandwidth, namely by using human-readable strings as addresses, making it much friendlier to visualize and debug as well as allowing for a much open-ended addressing scheme than MIDI. The general structure of OSC messages is as follows:

```
<address> <type-tags> <arguments>
```

With the `address` being a string containing hierarchical notation not unlike a filesystem path, and the `type-tags` being a string containing a list of types, each of which is a single character. The `arguments` are a list of values, the layout and sizes of which are determined by the `type-tags`. OSC allows the following types:

- `i` – 32-bit or 64-bit big-endian signed integer
- `f` – 32-bit or 64-bit big-endian IEEE 745 floating-point number
- `s` – `null` terminated string of ASCII characters
- `b` – 32-bit size in bytes followed by any binary data

All data is aligned on 4-byte boundaries. Each OSC packet can contain multiple messages, in the form of a bundle. Crucially, bundle messages are *atomic*, meaning their effects are to be handled concurrently by the receiver and can also contain other bundles, making the message format very flexible.

As we can see from this analysis, the OSC protocol is both very popular and well-designed, as well as fitting our purpose almost perfectly. However, while it facilitates transmission of data, it does not provide any structure to the data itself. This is a problem, as we will need to be able to interpret the data consistently.

4.1.3 TUIO 1.1

Luckily, thanks to the work of Martin Kaltenbrunner, there is already a protocol to "provide a general and versatile communication interface between tangible tabletop controller interfaces and underlying application layers" [34]. This protocol is called TUIO and is used by many applications, such as TouchDesigner.

TUIO is designed to be compatible with Open Sound Control, which itself is agnostic to transport methods. TUIO however defines a default transport method, usually referred to as TUIO/UDP. This transport method bundles TUIO messages into OSC bundles and sends them over UDP to port 3333. This means TUIO is designed with UDP – and therefore possible packet loss – in mind. Alternative transport methods are available as well, such as TUIO/TCP. The protocol itself consists of the following message types:

- `ALIVE` – current set of objects present on the surface
- `SET` – current state of an object
- `FSEQ` – unique (but monotonous) frame sequence number
- `SOURCE` – optional message identifying the application sending the data

Additionally, the protocol also defines a set of profiles for different kinds of tracked objects, each with a 2D, 2.5D and 3D variant. The following profiles exist for 2D surfaces:

- `/tuio/2Dobj` – Objects, possessing position and rotation
- `/tuio/2Dcur` – Cursors, possessing just position
- `/tuio/2Dblb` – Blobs, possessing position, rotation and area
- Custom profiles.

Profiles for 2.5D and 3D surfaces are analogous, but since those are not relevant to the device we are building, we won't discuss them further. These profiles set the attributes of the messages, and are used to distinguish between

different types of objects. As well as the values, the messages also contain a so-called Session ID, uniquely identifying the object on the surface – for example in case of multiple identical markers or finger touches. The messages can also optionally contain a velocity for position and rotation. Most notable quirk of the format is that no explicit messages exist for adding or removing objects. Clients are left to deduce these changes from subsequent `ALIVE` messages.

As we can see, this perfectly fits the data we'll be transmitting. This makes TUIO the perfect choice for communication of tracking data from our device to clients.

4.1.4 TUIO 2.0

Although TUIO 2.0 specification is considered stable as of November 16th of 2014 and even has a reference C++ implementation[30], it's still not widely used. Specifically, TouchDesigner still only supports TUIO 1.1 in its TUIO In DAT. Therefore, we'll stick to TUIO 1.1 for now.

4.1.5 Network Device Interface

Network Device Interface (abbreviated as NDI) is a protocol developed by NewTek and used to transmit high-definition video over the network. Among its main features are: the ability to transmit the video in a compressed format and thus greatly reducing required bandwidth, *frame accuracy* allowing syncing multiple output devices to a single source (useful in multi-projector setups) and relatively low latency.

For our use case, we will be aiming for 1080p at 60 Hz. According to the NDI knowledge base[31], this will require a maximum bandwidth 150 megabits per second. While this pushes us into using Gigabit Ethernet, it still leaves plenty of room for the rest of our data.

NewTek provides a royalty-free NDI SDK as well as many applications such as the NDI Studio Monitor for viewing NDI streams. However, these aren't supported on every platform. Namely, there seems to be a significant gap in compatibility with Linux. In chapter 7, we will explore these applications more closely and see if there are any alternatives, if necessary.

[30] `https://github.com/mkalten/TUIO20_CPP`
[31] `https://support.newtek.com/hc/en-us/articles/217662708`

4.2 Projection mapping

It is safe to assume that every case of projection mapping on our device will be unique (except for the default of projecting on just the surface). It can also quite easily be accomplished using just software tools and doesn't require much preparation on the device side, except maybe tweaking the projector sharpness or keystone configuration. Therefore, we don't need to support any extra features in our work as it can be done entirely by the end user of the SDK.

4.3 SDK

To make the use of our device and allow a wide variety of interactive content to be created for it, we will be implementing a custom SDK. This will allow users of the device to create an interactive tangible application without having to focus on the underlying hardware or network communication. The SDK should allow at least the following actions to be performed:

- receive tracking information from the device,
- set configuration parameters on the device,
- transmit video information to the device.

For the purposes of this thesis, we'll be implementing an SDK for our device as a Unity Plugin. Unity is a powerful and easy-to-use 3D application framework, which makes it a good tool for creating interactive applications. It will be designed to communicate with software running on the device.

4.4 Proposed enhancements

In this section, we will look back at some of the designs we have seen in chapter 2 and try to propose some enhancements to them to better fit our use case. We will focus especially on improving on the MIT Tactile Matrix discussed in section 2.4. But first, let's look back at what our primary use case will be and how that impacts our needs.

The device will be placed inside the GLab graphics laboratory on FIT CTU, to be used by students for their projects. Right off the bat, this makes things *very* complicated, since we can't possibly predict how the students

will attempt to use the device. We might also want to evolve its capabilities over many years, upgrade the hardware and/or relocate the whole device into another room.

This means that we want to be able to handle all of these scenarios (as well as the ones we have not thought of yet). This will be reflected mostly in the physical construction of the device, but it will also propagate into the way we will structure the SDK, namely into its modularity and extensibility.

4.4.1 Mobility

One of the biggest issues we can identify on the MIT Tactile Matrix is its mobility – or rather lack thereof. The device is designed to be placed in a fixed position, with only the camera mounted on the frame itself. The computer and most importantly the projector are mounted in an external location (as apparent in Figure 2.5), meaning they do not move with the device.

The introduces two major problems. Firstly, every movement of the device requires at the very least manual adjustment of the projector location, but more likely having to completely repeat the slow projection mapping process. Secondly, the camera is connected to the computer with a USB cable, which *severely* limits its range of movement. The recommended cable length for USB 2.0 is 5 m, with USB 3.2 further reducing that to 3 m. This means that we could run into connection issues with as little as three connected tables, *especially* if we're also using the USB cable to power the camera as the voltage drop may result in brownouts[32] of the camera circuit.

To address the problem with projector location, we will attach the projector directly to the device in a fixed position relative to the surface, eliminating the problem altogether. To allow the mounting of the projector *above* the surface, we will introduce an additional frame to top of the device. The frame will be removable, allowing for easier mounting of devices. This will serve as a mounting point for any external attachments, such as overhead projectors, lights and/or devices for depth map capture. This frame will be further discussed in subsection 4.4.2.

[32] A partial drop in voltage not large enough to cause a blackout. This can cause integrated circuitry to enter an undefined state and is very hard to debug.

To address the problem of connectivity, we will include a computer in the device, which will manage both the camera and the projector. We will connect to this computer using a network cable. The recommended range for both Cat5e and Cat6 cables is 100 m (a whopping 20 times the range of USB 2.0) with this range dropping to 55 m for 10GBASE-T (still 11 times the range of USB 2.0). This will allow us much more freedom in placing and connecting the tables while sacrificing very little speed.

This will also allow us to reuse existing network wiring, for example to connect devices across several rooms. This in turn makes the device much easier to transport between rooms, since it can be easily accomplished by a single person without having to interfere with internal wiring. Minimizing these interferences will further reduce the chances of untrained personnel incorrectly installing the device.

This improvement does however come at the increased cost of each table requiring its own computing unit. Thanks to the rise of low-cost computing platforms like the Raspberry Pi (of which version 4 B is currently available for as little as $35), we can safely accept this trade off.

Finally, we will design the table with the ability to mount wheels onto the frame to make moving it around the room as easy as possible.

4.4.2 Flexible design

Another issue with the MIT Tactile Matrix design (and indeed many such devices) is that it is very hard to adjust the construction on the fly. As apparent in Figure 2.7, the first design is composed of steel sections held together by screws which require holes to be drilled in the sections. While this does allow for changes to be made – for example changing the lengths of the sections – many of these adjustments require drilling new holes and/or essentially discarding existing material. This makes changes cumbersome and expensive.

The second design (also in Figure 2.7) is even worse as it is made of several steel sections welded together. This makes the design nearly impossible to change without a great deal of effort – larger or comparable to manufacturing the device in the first place, in fact. Despite that a construction consisting of

two opposing corners allows for great variation in surface size before becoming problematic, but this comes at the expense of making the design much less stable and more prone to misalignment.

Broadly speaking, this is not a big issue for devices made for a single purpose, but as our device will be used in many different contexts, we will need to be able to adapt to the different needs of the users. We will therefore focus on creating a design that can be changed *easily* (with minimal need for tools and time spent) and *drastically* (possibly transforming it into a different device altogether).

To address these shortcomings, we will make the frame *modular*. This will allow us to make very broad changes to it with minimal effort, as well as manufacturing many compatible attachments in the future. We will make the top frame introduced in subsection 4.4.1 removable as well as designing a removable tabletop that can be attached over the projection surface. This will be useful as storage for objects we don't want tracked.

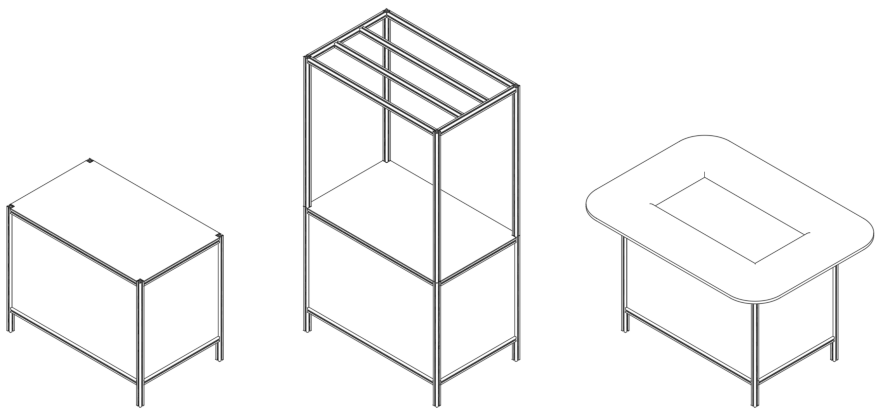


Figure 4.1: Technical drawings of variation examples

We can see that adding these options will allow us to easily adapt the device to a wide variety of situations where the MIT Tactile Matrix would not be suitable. Here are some examples of the different ways we could repurpose the device:

- mounting a Leap Motion[33] on the top frame will allow for tracking of hands and other objects above the surface,
- replacing the projection surface with a tub of sand with a Kinect mounted overhead to create an interactive sandbox,
- using the tabletop variant as an interactive map to play Dungeons and Dragons[34] on.

4.4.3 Multiplexing

The final improvement we will make over the design of the MIT Tactile Matrix is the improvement of multiplexing capabilities – that is, the ability to use multiple tables within one application at the same time. This will allow creation of a wider variety of applications with better opportunities for collaboration, hosting more users and increase the usable space for tracking objects.

You, dear reader, might be curious as to why we would do that, as the MIT Tactile Matrix already supports such a feature. The reason is that it requires each table's tracking camera to be connected to a single computer during such scenario, which quickly runs into the same constraints we already mentioned in subsection 4.4.1 due to the limits of the USB standard.

That's why the changes proposed to us will increase the operating range of the devices as well as increasing the number of tables that can be used at the same time. We will accomplish this by using a computer network to communicate between the tables and the computer, as well as designing the tables to allow daisy-chaining[35].

We do this by including a *network switch* in every device with enough ports to connect the internal computer to any amount of neighboring tables. This not only completely alleviates any requirement for us to do the multiplexing ourselves, it also protects us from any topological errors in the wiring due to the fact that the network switches will perform a spanning-tree algorithm to figure out the minimal skeleton graph of the network and ignore redundant connections (shown dashed on Figure 4.2).

[33] `https://www.ultraleap.com/`

[34] This use case is further expanded upon in chapter 8.

[35] Connecting several devices together in a *series*, with each device's output connected to the next one's input.

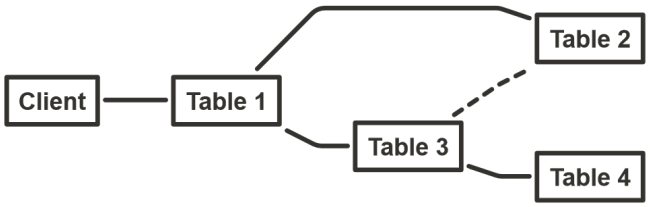


Figure 4.2: Daisy-chaining schema

Our **SDK** will also include the ability to define the position of each table in a configuration file and automatically translate received tracking coordinates to a global coordinate space. This will be further discussed in chapter 7.

Chapter 5

Hardware

In this chapter, we will surmise what electronic devices will be needed to build the device. We will take an in-depth look at the choices involved and requirements imposed by the device itself. This is done in a separate chapter than the construction, since the two are related only tangentially.

Most of the purchases made for this device will be influenced by external factors, such as the price, current stock availability or even by the purchases being made before this analysis was concluded due to time constraints. Despite these factors, we will attempt to discuss any rationale behind the product selection and the various trade offs in respect to price and availability, so that anyone trying to replicate this work may safely select a proper alternative should the devices we purchase not be available for them.

5.1 Connectivity

To enable the table to connect to the client computer as well as other tables when multiplexing, it's important to include a network switch in the table. Considering our table contains a single computer and can neighbor up to 4 other tables, we will require a network switch with at least 5 ports. We also need the switch to support a Gigabit Ethernet connection so that we don't run into bandwidth issues.

Luckily, network hardware has become shockingly cheap and such a switch will only cost around $20. The switch is a fairly passive network component

from a user's point of view, which means that we don't have to spend too much time weighing the alternatives available on the market. For this project, we've chosen the reasonably priced TP-Link TL-SG105, which is a 5 port Gigabit Ethernet switch available for $15.99 on Amazon[36] as of the time of writing.

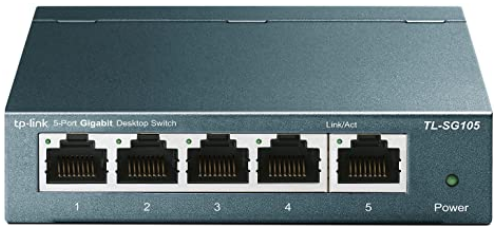


Figure 5.1: TP-Link TL-SG105

5.2 Camera

The camera is the centerpiece of the tracking solution, as it's solely responsible for the quality of the tracking data. There are many parameters we will need to consider, from overall image quality and frame rate to fine details such as the lens parameters and sensor noise.

We will preface these sections with the fact that we will be using one of the many camera models based on the `OV2710` sensor. These cameras provide very good quality and frame rate at a very low price, and price is something that constrains us the most at the moment. The following sections will discuss the various aspects we want to consider when choosing the camera, even when those might not be perfectly reflected in our chosen camera.

5.2.1 Frame rate and Latency

Perhaps the most noticeable effect on the result will come from the frame rate of the camera, as we can only react to the movement of the tracked object as fast as we can see it. This is not the whole story though, as the time it takes us to process the captured frame – latency – plays an equal, if not larger role in the "look and feel" of the device. There's only a limited amount of speed we can gain on "our" side of the equation, so picking a camera that offers sufficiently high frame rate and sufficiently low latency is crucial.

[36] `https://www.amazon.com/dp/B00A128S24`

At first glance, it might seem that any frame rate over the refresh rate of the projector might be wasted, as we won't be able to change the output fast enough for the user to notice. However, things are slightly more complicated than that, since there's another factor that the frame rate influences and that is *motion blur*. With higher frame rates, the motion blur of the image is significantly reduced, and therefore it's easier to recognize any tracked objects in motion. This can lead to significantly faster feeling response times, even though the visual frame rate is still the same.

5.2.2 Infrared Spectrum

A crucial part of our design is the ability to track objects in the infrared spectrum. This is because we will be projecting an image and tracking objects from the same side of the projection surface, and tracking in the visible spectrum would run the risk of the projected image interfering with the object markers. Constraining the camera to the near-infrared spectrum[37] prevents this, allowing us to freely project content without obstructing the markers.

Figure 5.2: Comparison of regular (left) and infrared (right) camera images

To be able to track in the infrared spectrum, our camera needs to be able to *see* the infrared spectrum. Not only that, we need to see *only* the infrared portion of the spectrum. This is usually not desirable for most cameras, causing them to be fitted with an IR-cut filter. This is the opposite of what we want. While infrared industrial cameras tend to be expensive, we can turn most regular cameras into an infrared one by removing the IR-cut filter, or simply use a "night-vision" camera, where the filter is typically absent.

[37]The "infrared spectrum" means a band of light with wavelengths between 700 nm and 1 mm, which is a considerably wider band of light than visible light. For the purposes of this work, whenever we refer to infrared light, we're talking about light with the wavelength of 850 nm.

Additionally, we have to prevent visible light from hitting the sensor. We can do this with an IR-pass filter, which is a filter that only allows a specific infrared wavelength of light through. These can be purchased as filters for regular DSLR cameras. We will be using the **Zomei 850nm filter**, available for around $30 from a local reseller. This filter has to be mounted in front of the lens, which can be accomplished with an adapter.

Figure 5.3: Zomei IR filters[38].

5.2.3 Optics

To actually capture the whole of the tracked surface from sufficiently up close, we need to consider the optics of the camera. The `OV2710` sensor is not a very good choice for this, as it's very small compared to other cameras. Thankfully, this can be compensated for with a sufficiently wide lens, which will in turn introduce a significant amount of distortion to the image. This can be compensated for with software calibration, which will transform the image to preserve the rectilinear layout of the tracking area.

Since we will be capturing a 16 : 10 surface with a width of 1 m from around 60 cm away, we will need a horizontal FOV of at least 80°. This might be more depending on the aspect ratio of the camera, which is more likely to be 16 : 9.

Another thing to consider is the lens mount on the camera, which can come in many form factors. The most usual lens mounts with small cameras like these are either C-Mount, CS-Mount or M12. The compatibility of these

[38] `https://davidkennardphotography.com/blog/990-zomei-ir-filters-review-comparison.xhtml`
[39] `https://docs.opencv.org/3.4/d4/d94/tutorial_camera_calibration.html`

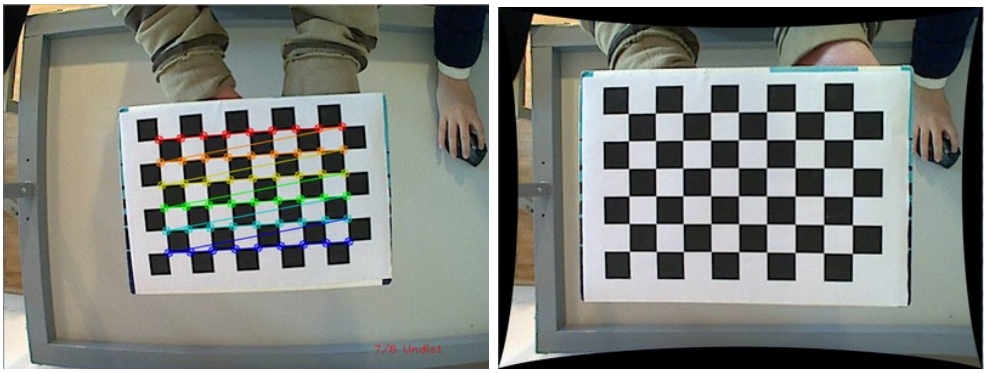


Figure 5.4: Comparison of a distorted (left) and rectified (right) images[39].

mounts can be found in Figure 5.5. The main issue is the incompatibility of C-Mount lenses with CS-Mount fixtures, since they have different flange heights. It's fairly unimportant which mount we choose, as long as it's compatible with the lens we want to use. This might even be completely irrelevant if the camera comes with a built-in lens.

	C-Mount	**CS-Mount**	**M12**
C-Mount	Compatible	With adapter	With adapter
CS-Mount	Unfocusable	Compatible	With adapter
M12	Maybe	Maybe	Maybe[40]

Figure 5.5: Lens (vertical) and fixture (horizontal) compatibility.

For infrared tracking, it is also important to keep in mind that not only does the camera have to be able to capture the infrared spectrum, but it has to pass through the lens first. Some lenses feature a built-in infrared filter, which can block the infrared light before it reaches the sensor. In such cases, it is sometimes possible to remove the filter, but purchasing a lens without a filter from the manufacturer is always a better option.

In conclusion, the most important aspects to keep in mind for the lens is the compatibility with the camera fixture, FOV produced in combination with the camera's sensor and the presence of an IR-cut filter in the lens itself.

[40]The design of M12 lenses is not standardized across the board, so they might not be compatible between manufacturers.

5.3 Lighting

For the infrared camera to actually see anything, the object it's looking at needs to actually be illuminated with infrared light. That's why we are going to need a light source that emits enough light in the infrared spectrum to illuminate the surface of the device, be it from above or below. Fortunately, there are many infrared light sources available on the market, because they are very commonly used as floodlights for outdoor security cameras at night. These offer a durable and cheap option with lots of mounting options (as the light sources are often mounted on buildings and similar structures).

There is one factor worth considering thought, which is the glare created by the light source on the surface. This will almost certainly cause clipping to occur in the camera with subsequent loss of information due to the low dynamic range of the sensor. It might therefore make sense to use a diffuse light source such as an infrared LED strip, which can quite conveniently be built into the aluminum profile of the final design discussed in section 6.6.

For the purposes of this work, we weren't able to procure any such LED strip with our monetary constraints and schedule, so we will be using a cheap infrared lamp purchased from Amazon[41]. Many of these outdoor spotlights come with a daylight sensor that turns them off during the daytime. This is usually a photoresistor[42] and simply disconnecting it will cause the light to be permanently on.

Figure 5.6: IR Spotlight

[41] `https://www.amazon.de/-/en/dp/B07XDXZ329`

[42] An electrical component that decreases its resistance with respect to the amount of light it receives.

5.4 Projector

Right after the camera, the projector is probably the most prominent hardware included with the device. A sort of inverse of the camera, its parameters are fairly similar. Our choice of projector was "set in stone" due to external factors, which meant we had to adapt the design to its parameters a little.

The main parameter a projector has (other than the image characteristics like resolution and frame rate) is what's called a *throw ratio*. This describes the ratio between distance from the lens and the *width* of the projected image. For example, a projector with a throw ratio of 0.5 : 1 will project an image with a width of 1 m from around 50 cm away.

The projector we will be using is the **Optoma ML1050ST+**, which comes with a short-throw lens with a throw ratio of 0.8 : 1, resolution of 1280×800 and a 60 Hz frame rate. It can be mounted in any orientation using the built-in camera screw on the bottom, which makes it perfect for mounting to our frame. It's also very compact and lightweight, coming in at 420 g, which means it doesn't weigh the device down and makes it easier to move and safer to use. The projector requires an HDMI connection to the computer.

Overall, the main thing to keep in mind when picking a projector is its throw ratio, its form factor and mounting options and its connection to the computer. Picking a heavier projector would run a serious fall risk when mounted above the glass projection surface and picking an unnecessarily large one would cost us distance from the lens when mounted from the bottom.

Figure 5.7: Optoma ML1050ST+

5.5 Computing Platform

Lastly, the device has to include a computer which will process the data from the camera, as well as incoming and outgoing network streams, including outputting the incoming image stream using the projector. This means that at the very least, we will require the computer to have:

- 1x USB 2.0 (or higher) port
- 1x Gigabit Ethernet port
- 1x HDMI (out) port

The biggest constraint when selecting a computer will be its price. Fortunately, our requirements are satisfied by the Raspberry Pi 4 Model B, which in its 8 GB incarnation is available for around $75 or CZK 2250. This makes it cheaper than options like the Intel NUC family of computers, which generally tend to cost more. The Raspberry Pi comes without any sort of casing or power supply, so additional components are required. However, these don't tend to influence the price quite as much.

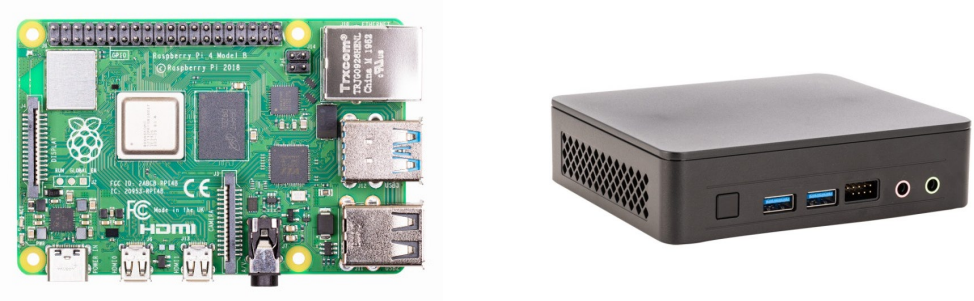


Figure 5.8: Raspberry Pi 4 Model B[43](left) and NUC11ATKPE[44](right).

The crucial difference between the Raspberry Pi and other small computers such as the Intel NUC is the fact that the Raspberry Pi is based on the ARM architecture. This makes running some software more difficult or impossible, but since we're writing most of the software ourselves, this shouldn't be a problem.

[43] https://rpishop.cz/2611-raspberry-pi-4-model-b-8gb-ram-0765756931199.html
[44] https://simplynuc.eu/product/nuc11atkpe-full/

5.6 Conclusion

In this chapter, we have reviewed some crucial pieces of hardware included in the device and detailed some rationale behind our choices as well as other options available. We have discussed the primary aspects one should focus on when assembling a similar device.

Chapter 6

Construction

This chapter will go over the process of designing physical form of the device itself as well as the challenges encountered during this process and their respective solutions.

In the first section, we will be designing the frame of the device upon which all other components will be attached. We will go over the materials chosen for this, their properties and the rationale behind the choices. Then we will explore the opportunities for modular design and how to make the device as extensible as possible, as well as looking for possible use cases and designing some extensions to better suit the devices for them.

The following sections will explore the issue of choosing a surface for projection, the interior of the table as well as connectivity of the device. Lastly, we will present two iterations the design went through during the design process.

6.1 Frame

The most integral part of the whole device is the frame. Its design will influence most aspects of the device, which means we'll have to first ask ourselves some important questions before we settle on a design, like:

- What's the general shape of the frame?
- What should the dimensions be?
- What materials should we use?

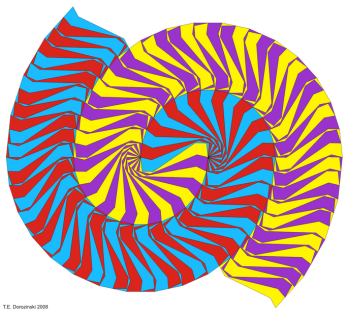
These questions likely aren't going to have clear-cut and/or objective answers, so we will have to consider our specific use cases and make certain trade offs to make the design fit our purpose as cleanly as possible. We will, however, try to identify these trade offs and consider alternative or better solutions for cases with different requirements. Let's reiterate what our specific requirements are:

- *flexibility* – allowing students to adapt the design for their projects,
- *multiplexing* – enabling multiple devices to be used in tandem,
- *mobility* – making the device easy to transport.

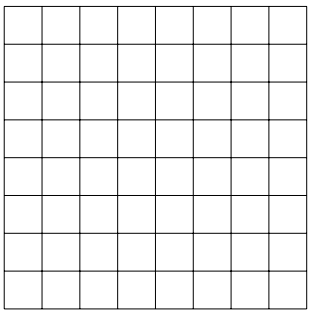
6.1.1 Shape

Let's look at what shape we want our table to have. Since we would like to be able to use multiple tables in our setup, it would make most sense to choose a shape that *tessellates*[45] easily.

Furthermore, we might want all the tables to have a *congruent*[46] shape, for example to make the setup as immune to individual device failure as possible, since it will allow us to easily swap out parts between them and easily manufacture more of them. Or, if not congruent, than at least *similar*[47]. A tessellation in which all tiles are congruent is called a *monohedral tiling*.



(a) Voderberg tiling[36]



(b) Square tiling

Figure 6.1: Various kinds of monohedral tilings.

[45] Tessellation is a covering of a surface using one or more shapes with *no gaps or overlaps*.[35]

[46] Two geometrical figures are congruent if they are identical in shape and size. This includes the case when one of them is a mirror image of the other.[35]

[47] Two figures are similar if they are of the same shape but not necessarily of the same size. This includes the case when one is a mirror-image of the other.[35]

Sadly, as we can see from Figure 6.1, even constraining ourselves to monohedral tiling leaves us with a considerable range of options, meaning we will have to employ some common sense to choose the best one.

First off, let's consider other components, namely the camera and the projector. It's safe to assume that the camera will capture, and the projector will project a rectangular image, as evident from a cursory look at any consumer electronics store at the time of writing. Moreover, these devices will most commonly be found in the 16:9 or 16:10 aspect ratios. To avoid loss of pixels due to mismatched aspect ratios between the table and the camera or projector, we should choose any one of the common aspect ratios.

Another viable (and quite more interesting) solution would be to make the sizes follow the same scheme as ISO 216 international standard for paper sizes, commonly also known as the A-series papers. This standard specifies the aspect ratio of the sheets of paper as $\sqrt{2} : 1$, meaning that two sheets of $A(n)$ papers will compose exactly to form one sheet of $A(n+1)$ paper when connected at one of their long edges. Furthermore, the standard defines the area of an $A0$ sheet of paper to be exactly $1\,\mathrm{m}^2$, which makes the area calculation for any $A(n)$ sheet of paper equal to $\frac{1}{2^n}\,\mathrm{m}^2$.

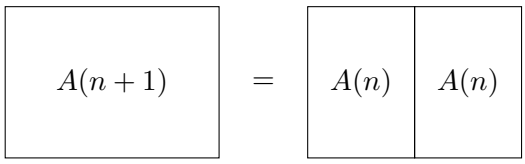


Figure 6.2: Schematic of A-series paper sizes.

Adapting this to the table, we would have to take into account the outer rim of the surface, shrinking our projected area appropriately on each side. Then, however, it would allow us to manufacture tables of arbitrary degree in the system and allowing us to simply stack two smaller tables to complement a larger one. It would however lead to loss of pixel data when using projectors with different aspect ratio than $\sqrt{2} : 1$. For example, a 16:9 projector would lose up to 21% of pixels when projecting onto such a surface, which is a significant loss.

Another alternative would be to make the tables completely square. This would make them very pleasant to use by a group of people, since everyone

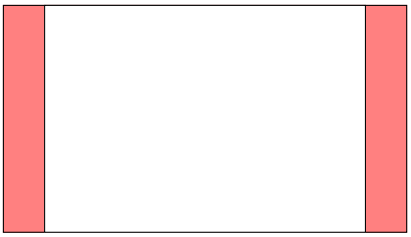


Figure 6.3: Cropping from $16:9$ to $\sqrt{2}:1$.

would have access to a table edge of the same length. It would also make them rotationally symmetrical allowing for easier tiling and setup, as well as letting the content itself be rotated in any direction. Sadly, as with the A-series paper based design, it would lead to a loss of $\frac{7}{16}$-ths, or 44% of pixels when using a $16:9$ projector. Alternately, we could project with two $16:9$ projectors with blending, but this would still lead to a loss of $\frac{1}{9}$-th, or 12% of pixels.

Since our case is constrained by price and not by any specific form requirements, we will opt to follow the aspect ratio of our projector to minimize the loss of pixels. This is however a fairly arbitrary decision which won't be suited for all or even most situations.

6.1.2 Dimensions

Once we know the shape our table will be, choosing the actual dimensions is fairly straightforward, but still complex. While our width and height will be constrained by one another, there are still multiple other dimensions for us to decide:

- width *or* height of the projection surface,
- clearance between the bottom of the table and ground,
- height of the surface from the ground,
- and clearance between the surface and the top of the hat.

We will start by deciding that the width of the projection surface will be 100 cm. This will greatly reduce the necessary mental work when figuring out relationships between screen units and real world units, as well as make it easy to measure any additional equipment for the table. Using the $16:10$ aspect ratio of the projector we picked out in section 5.4, we can calculate the height of the projection surface to be 62.5 cm.

The height of the surface from the ground will be based on the height of normal surfaces people might encounter in the world. The recommended height of a laboratory countertop in an academic institution as per **ČSN EN 13150** is 90 cm, which is a good benchmark considering we expect people to be mostly standing around the table. Therefore, we will aim to make the frame 85 cm high, to leave some room for optional wheels to take us over 90 cm, while still allowing for the table to be operated comfortably while sitting down on an appropriately high chair (for example a bar stool).

For the ground clearance, we will aim for 20 cm including optional wheels, or around 14 cm without wheels. This leaves plenty of space inside the table, while allowing any regular person to comfortably stand close to the table without hitting their toes on the frame.

As for the hat clearance, we will make the total height slightly more than the average height of a person, to allow most people to comfortably see and lean under it. The average height of a person in the Czech Republic is 178 cm[37], which allows us to round the hat clearance to 1 m, making it the same as the width of the surface and creating a perfectly square area under two sides of the hat.

Dimension	**Value**
Surface width	100 cm
Surface height	62.5 cm
Surface height from ground	85 cm
Ground clearance	14 cm
Hat clearance	100 cm

Figure 6.4: Overview of the chosen dimensions.

It is important to add that these planned dimensions are only approximate and like any plans, they are unlikely to survive confrontation with reality. They are subject to slight changes and tolerances as well as adaptations to any unforeseen issues that may arise during the actual assembly process. They also do not include dimensions of the frame itself, which will influence the total dimensions of the table.

6.2 Surface

The centerpiece of the whole device and likely the part users will interact with the most is the surface. It will serve several critical roles which we should consider in its construction:

- a target for the projector,
- a barrier for the tracking camera,
- a support for the tracked objects.

Sadly, all of these requirements are at odds with each other in subtle ways. This means we will certainly have to make some trade offs when selecting a projection surface. It will also be nearly, if not completely impossible to select a single surface for every use case. We will therefore try to make it as easy to replace as possible, to keep the goal of *flexibility* we set with the frame in section 6.1.

To support the tracked objects and other objects placed on the surface, we will want it to be as rigid and sturdy as possible. This roughly translates to being as *thick* as possible since material properties can only take us so far. However, increasing the thickness may lead to quality degradation in the camera image when tracking from the bottom as well as chromatic dispersion in content projected from the bottom.

Figure 6.5: Example of dispersion (left) compared to a correct image (right).

Projection and tracking may also be in conflict if done from opposite sides. For tracking we want the material to be as clear as possible, so we can discern the markers on the bottoms of objects. For projection, however, we want it to reflect as much of projected light as possible. Not only that, but

we want the surface to be *diffuse*. This is to maximize the viewing angle of the projected image from the top, as well as to hide the bright spot of the projector when it's placed at the bottom.

This causes a stark conflict between the two. If we want to have any chance of capturing a clear image from the bottom, the diffuse layer has to be placed directly on the surface to minimize any area between it and the marker. Some diffuse surfaces like projection films or frosted glass may be exceedingly susceptible to scuffing and mechanical wear, making them unsuitable to put objects on.

For our device, we have chosen to go with a 6 mm thick tempered glass. This glass is very clear and reasonably thin, making it a minimal issue for the camera. With a rating of 120 N/mm^2 it also has more than enough strength to hold any reasonable object we may want to place on it. But most importantly, were the glass to shatter, the tempering will cause it to break into very small chunks instead of shards as ordinary annealed glass.

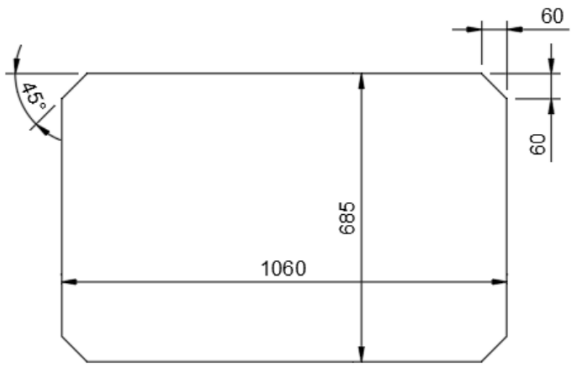


Figure 6.6: Technical drawing of the glass surface.

One important consideration is that it's exceedingly difficult to cut this material into a non-convex shape, as any internal angles are very prone to breakage during cutting. This can make such a cut easily increase the price of the glass by up to 200%. For this reason, we've decided to cut corners[48] and leave the cuts straight, as per Figure 6.6.

[48] Pun intended.

Figure 6.7: Surface during projection film testing.

6.3 Interior

The design of the actual interior of the device will be fairly straightforward, as most of it will need to be empty to not obstruct the view of the projector and the camera. We will, however, need to house the following components we picked in chapter 5 as well as the necessary wiring in the interior:

- the projector,
- the tracking camera,
- the infrared light,
- the computer,
- the network switch,
- a power strip.

We will designate an "safe area" in which no objects should be place to avoid interfering with the tracking and projection. We will establish this area as roughly the frustum of the camera, since entering the frustum of the camera will be the most difficult issue to diagnose without looking at the camera stream. Any object blocking the projector will be immediately apparent on the projection surface when the projector is turned on, making it much less of a concern.

There needs to be a place to mount all the components, since we don't want them loosely sitting in the device. For this, we will include a small platform at the bottom of the frame to provide a place for the components. We also need to consider a housing for the necessary power and network cabling to prevent it from accidentally entering the safe area. We will accomplish this by including a cable box for any excess cable length as well as placing some cable fasteners on the frame to keep it in place.

6.4 Connectivity

Due to the invariable and cruel constraints of the physical universe we reside in, as well as the limits of current technological advancement, we will need to connect the device with the rest of the world using cables. As we established previously, the main mode of data communication with other devices will require a wired network connection. Additionally, we will need to supply sufficient power to the device.

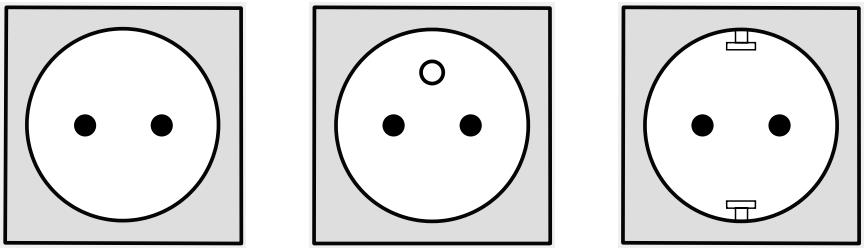
6.4.1 Power

While it's possible to transmit power over ethernet, it would not be sufficient. The original IEEE 802.3af-2003 **PoE** standard provides up to 15.4 W of DC power. The updated IEEE 802.3at-2009 **PoE+** standard can provide up to 25.5 W, and the IEEE 802.3bt-2018 **PoE++** standard expands this up to 71.3 W. Even that falls short of our selected projector's (section 5.4) maximum power consumption of 77 W and is therefore unsuitable for our project.

This means we will need to provide power to our device from a standard wall outlet. In the case of the Czech Republic, outlets come in the form of **Type C**, **Type E**, or **Type F** sockets.

Since all parts of the device come with their own power supplies, we can simply mount a regular power strip inside the device to provide a socket for each device and route the plug from the power strip outside. We will require a power strip with at least four sockets. One for each interior device (projector, tracking camera, and the computer) and one to ground the frame with as an additional safety measure.

[49] Yanpas, CC BY-SA 4.0, via Wikimedia Commons (`https://commons.wikimedia.org/wiki/File:Plug_types.svg`)



(a) Type C (b) Type E (c) Type F

Figure 6.8: Socket types commonly found in the Czech Republic[49].

6.4.2 Ethernet

To connect our device to the ethernet network, we will rely on standard Ethernet cables. But first, let's take a detour to explain some cloudy terms involved in this. There is much misconception about networking cables prevalent in the industry today, which would mean that the widely accepted terminology would not be factually correct, while the correct terminology would unnecessarily confuse most readers.

You might be used to "regular ethernet cables" being referred to in multiple ways, such as **Cat5e**, **UTP**, **RJ45**, **1000BASE-T** or **Gigabit Ethernet**. The first two of these refers to the cable itself, while the third refers to the cable's connector. The last of the terms refer to the standard for communicating information over the cable. To make matters worse, what's usually referred to as **RJ45** is actually a **8P8C Modular Connector**, which is compatible with **RJ45** jacks[50], but not with **RJ45** plugs. For the rest of this work, we will refer to this connector simply as **8P8C**.

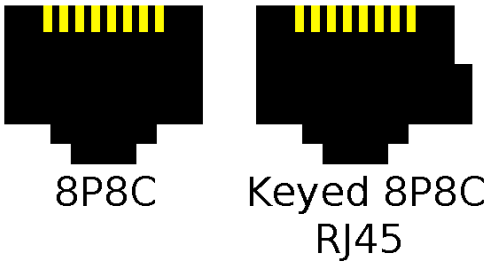


Figure 6.9: Difference between 8P8C and RJ45 jacks[51].

[50]The term "jack" here referring to a *socket*, not a *plug*. This is unlike the word's common usage with phone connectors (also known as audio jacks), where the word can refer to either.

Let's start with the first of the bunch, that is **Cat5e**. This denotes what is called a cable's *category*, which represents the bandwidth and distance capabilities of the cable. The most commonly encountered categories as of the time of writing are **Cat5**, **Cat5e**, **Cat6** and **Cat6a**. You may encounter lower (or indeed higher) categories than these, but it's unlikely – at least in a normal household context at the time of writing. The relevant parameters of the categories are detailed in Figure 6.10. To satisfy our bandwidth requirements from section 4.1, we will need to use a cable category of **Cat5e** or higher.

Category	**Bandwidth**	**Max. Data Rate**
Category 5	100 MHz	100 Mbit/s
Category 5e	100 MHz	1 Gbit/s
Category 6	250 MHz	1 Gbit/s
Category 6a	500 MHz	10 Gbit/s

Figure 6.10: Overview of common cable categories

Up next is the mysterious **UTP** acronym. This stands for "Unshielded Twisted Pair" and refers to the internal layout of the cable, specifically that the cable contains twisted pairs of cables[52] without any additional shielding. This is also referred to as **U/UTP**. The pairs can also have individual shielding (**U/FTP**), overall shielding (**F/UTP**, **S/UTP** or **SF/UTP** depending on the type of shielding), or both (**F/FTP** and so on).

Shielding reduces interference and shouldn't generally make much difference in our case, since we won't be running large numbers of cables in parallel, or running them through areas of large interference. However, it is worth noting that in case of network issues, it is worth checking that any excess length of the network cable is not coiled up.

Despite all the confusion about the naming of the actual **8P8C** connector, it's relatively hard to go wrong when buying the cable. Any "normal ethernet cable" should be equipped with the **8P8C** connector. The same goes for most normal network equipment, such as switches.

[51] Keministi, CC0, via Wikimedia Commons (`https://commons.wikimedia.org/wiki/File:8P8C_vs_RJ45_female_connectors.png`)

[52] Interestingly, the practice of cable twisting dates back to the end of the 19th century. The installation of electric trams in some cities caused interference in telegraph lines, which was mitigated by transposing the telegraph wires every few poles.

It is also possible to expose ethernet jacks on the outside of the device using panel mounted connectors, such as the etherCON series manufactured by Neutrik[53]. EtherCON panel mounts in combination with etherCON cables create a lockable connection which is much more resilient to damage than standard **8P8C** connectors. They are also optimized for audio and video use. They are, however, fairly expensive (with just the panel mounted connector coming in at around CZK 450), meaning we won't use them for our device.

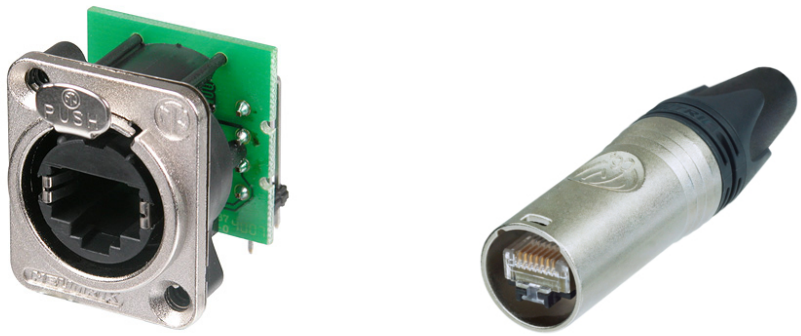


Figure 6.11: EtherCON panel mount and cable respectively[54].

In conclusion, when purchasing a cable for use with our device, make sure it's **Cat5e** or higher, so it supports at least **Gigabit Ethernet**, is capped with an **8P8C** connector (frequently mislabeled as **RJ45**) and don't worry about shielding unless you know you'll need it. And if you think the cabling poses a tripping hazard, use a secure panel mount.

6.5 Original design

This section will detail the original design of the device. You, dear reader, might find yourself wondering why the design doesn't seem to incorporate many of the decisions we've established in the preceding sections. This is because this design was created in a very early stage of development, and served mainly as a proof of concept, as well as learning experience to base the future design on. With that in mind, let's talk about the design and point out all the issues that we have encountered.

[53] `https://www.neutrik.com/`
[54] `https://www.neutrik.com/en/products/lightning/ethercon`

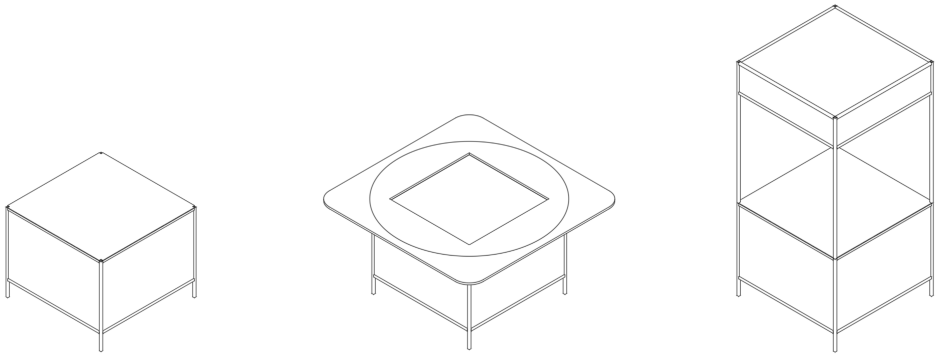


Figure 6.12: V1 drawings from left to right: base, with tabletop, with hat.

Even this early in the design process it was clear that modularity is an important goal. Therefore, the design was composed of three parts: the main body, the tabletop and the hat. Each part was composed of 20 mm square steel sections (detailed in Figure 6.13) welded together. This made the design very sturdy, but near impossible to modify.

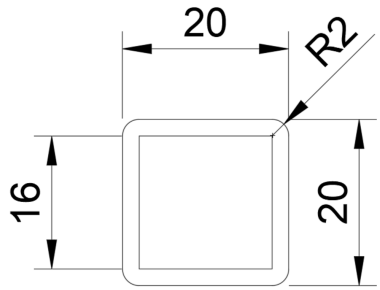


Figure 6.13: Steel section drawing.

The main difference to the final design is the dimension, which comes in at 1 m by 1 m for the projection surface. This made projection very cumbersome, as detailed in subsection 6.1.1. This brought an interesting issue in that most internal doorways in the Czech Republic tend to be between 70 cm and 90 cm wide. While not immediately obvious, this makes the device almost impossible to fit through most internal doorways as it's impossible to disassemble the welded frame. Even the wider doorways will require flipping the device on it's side and removing the glass first.

Material	Amount	Price per Unit
Steel section	24.560 m	CZK 96
Plywood	3.6 m^2	CZK 688
Perforated sheet metal	2 m^2	CZK 1730
Total		CZK 8295

Figure 6.14: V1 design's bill of materials.

The interior of this design was slightly bulkier, with the side paneling made from plywood and the bottom fully covered with a perforated sheet metal which made it very easy to mount anything to the bottom of the interior. This was later replaced with another plywood panel to give further freedom of placement in the interior, as well as prevent sagging which would occur with the perforated sheet metal.

The design also features a bulkier hat, with its own side panels. This was meant to conceal any hardware mounted on the hat as well as shield it from outside light, but this proved to not be worthy trade off for the extra weight the paneling would add. Its legs were fitted with smaller steel sections welded in place, allowing it to be easily slotted into the base with little effort.

The tabletop was also present in this design and required custom support pieces similar to the legs of the hat to hold it in place. It was designed to be manufactured in either square or round shape (both of which are reflected in the technical drawings) from an 18 mm plywood sheet. The tabletop slightly infringes on the projection surface, trading less projection space for more table space and increased stability.

This design proved to be fairly cheap thanks to the low price of steel sections and plywood[55]. This price difference was however outweighed by the time and effort required for manufacturing as well as the impossibility of future modification. A rough bill of materials with estimated cost per unit (including VAT) can be found in Figure 6.14. It does not, however, include the cost of painting the frame or the cost of assembly. It also does not consider any excess material created due to the dimensions the material is sold in.

[55]The price of plywood (and wood in general) has sharply increased during the COVID-19 pandemic starting in 2020, in some cases by as much as 150%, making this statement less true than it would usually be.

The technical drawings for this design can be found in Appendix A. The complete design files can be found on the enclosed storage medium in the `design_v1` folder, as per the listing in Appendix D.

6.6 Updated design

The second iteration of the device diverges from the first design in multiple ways. The main difference is that the frame is now composed of *anodized*[56] aluminum sections joined by simple locks tightened with a single hex key. This makes the frame easy to assemble and disassemble, as well as easily modifiable and extendable. This comes at the cost of a higher price for the material and less stability (although hardly noticeable when properly tightened).

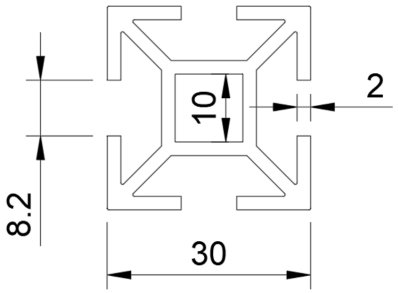


Figure 6.15: Aluminum section drawing.

Interestingly, despite having much lower density than steel[57], the aluminum frame's weight is very similar to the steel frame. This is due to the fact that the new aluminum frame uses thicker sections (30 mm compared to the original 20 mm) as well as much more complex shape, as per Figure 6.15.

As we can see in Figure 6.16, the design went through several iterations, before we arrived at the final design. The hat was made significantly more lightweight and the dimensions have shrunk to fit the projected area as described in subsection 6.1.1. The plywood side paneling was also replaced with MDF which reduces its weight and makes installation of panel-mounted components easier.

[56] Anodization of aluminum prevents it from degrading or staining everything it touches. I highly recommend building from anodized aluminum only.

[57] Around 2700 kg/m^3 for aluminum vs. 7850 kg/m^3 for steel.

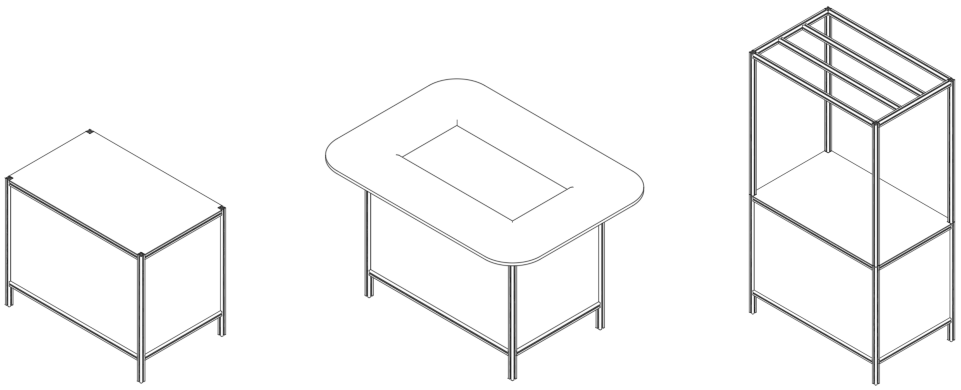


Figure 6.16: V2 drawings from left to right: base, with tabletop, with hat.

The most drastic difference in function is the absence of any kind of bottom or top. This is because the nature of the aluminum sections allows them to double as a mounting point for virtually anything – either directly on the frame with a screw mount, or to a panel slotted between two sections. This makes it possible to quickly swap out components using prepared sections. It also allows us to mount any cabling or other parts such as handles directly on the frame.

One thing missing from this design is a built-in way of attaching the hat to the base, which was solved using a custom-designed 3D printed part glued into the hat frame. This prevents the hat from slipping off the frame and also allows it to be easily removed without any tools. The design of this part is detailed in Figure 6.17. As for the tabletop, it now doesn't require a custom support piece but can instead be attached with a standard 90° joint secured with an M6 screw.

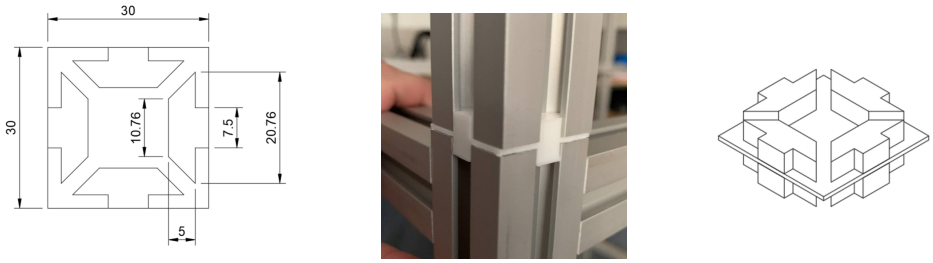


Figure 6.17: Hat joint pictures.

As stated before, this decision made the design significantly more expensive, as a meter of an aluminum section can come in at roughly two to three times the price of a steel section. This was compensated for more than enough by the fact that the design is now easy to change, and the assembly time was cut down by an order of magnitude as we now don't need to weld or paint anything. It is however worth noting that the T-junctions require a hole to be drilled at the end of the aluminum section, which may cause burring that must be manually removed.

As with the previous design, the technical drawings for this design can be found in Appendix B and the complete design files can be found on the enclosed storage medium in the `design_v2` folder, as per the listing in Appendix D.

Figure 6.18: The frame before and after assembly.

6.7 Conclusion

In this chapter, we have covered various aspects of the device's design and the rationale behind the choices we made during the process. We discussed the shape of the device, the materials used and the considerations for the hardware that will be present in the interior.

We then looked at the first design which was created before any of the analysis, identified the issues the design had and how they were resolved during the following iterations.

Finally, we looked at the actual final design of the device and went over the manufacturing process and issues we encountered. We also discussed pricing and material choices. This concludes the physical design chapters, leaving us to move onto the framework implementation and software architecture design in chapter 7.

Personally, I found this part of the thesis to be the most rewarding, since it allowed me to learn more about physical manufacture, which is something I have been looking forward to doing. I hope that this chapter has been helpful to you and that you find it entertaining to read and useful when designing your own device in the future.

Chapter 7

SDK

To enable users (which will be mostly comprised of students in the graphics laboratory) to effectively use the device without having to deal with the internals of tracking and network transport, we will provide an SDK – software developer kit. This will allow users to interact with the device using a high-level API. The SDK will also include some supporting software, for example a tracking marker generator.

This chapter will describe the architecture of the SDK, its features and limitations as well as outline how to use it. It will also go over some third party libraries and programs involved in the SDK's operation. Where relevant, we will present fragments of code showing the usage of the SDK[58].

7.1 Architecture

Broadly, the SDK is composed of two parts: the software on the device responsible for all the functionality of the device, and the framework which users can use to communicate with the device. While the framework is fairly monolithic in its Unity Plugin form, the device itself runs multiple applications at the same time, each focused on a different part of the functionality. The two parts communicate over the network using the NDI and TUIO protocols respectively.

[58]The formatting of the code samples has been heavily edited to better fit the narrow pages of a printed medium and does not reflect the personal choices and/or opinions of the author.

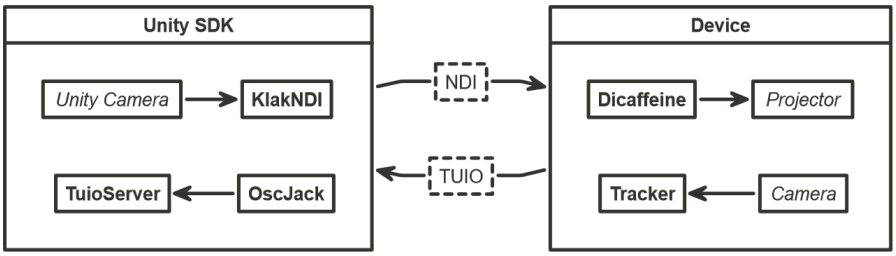


Figure 7.1: Architecture diagram.

The tracker is actually built as a C++ library first, which means it can be easily used to create custom trackers that are not built into the SDK. There's a number of utilities that are included in the SDK, such as a marker generator, that are discussed in later sections.

7.2 Tracker

Most of the heavy lifting on the device side is done by the `tracker` program. This program is responsible for all the tracking functionality of the device, as well as sending data over the network. It was written with our main goal of *flexibility* in mind, which is reflected in the structure of the program (depicted in Figure 7.2).

As we can see it's written in a very modular fashion, with each piece of functionality being encapsulated in a single class (derived from `Capture` for input, `Driver` for tracking and `Sender` for network communication). This makes it easy to replace or change each aspect without having to understand the rest of the program.

Out of the box, it comes with a number of built-in components which can serve as a starting point for customization. There's only one provided capture class called `CameraCapture`, which receives input from a camera. There's the `ArUcoDriver`, which provides the tracking functionality for markers discussed in section 3.5, and the `TestDriver` which simply outputs dummy data for testing purposes without a camera. There's also a `TUIOSender`, which sends data over the network using the `TUIO` protocol. The `ScreenSender` doesn't actually send any data but instead views it in a window.

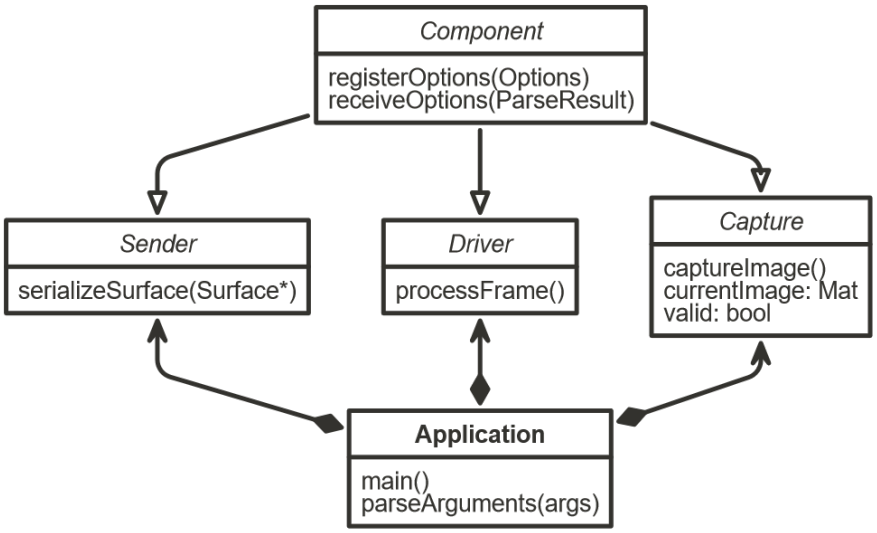


Figure 7.2: Tracker class diagram.

The `Application` class makes it easy to compose a custom tracking solution from multiple `Driver` and `Sender` implementations. It also takes care of parsing any command-line options and arguments provided when running the program using the **cxxopts**[59] library. Each component can register custom options to be parsed. These options in turn make their way into the automatically generated command line help printout (seen in Figure 7.4), which is displayed when the user runs the program without any arguments. This leads to very straightforward composition, as seen in Figure 7.3. Duplicate options are not handled, nor is any option sharing between components.

```
Application application { "tracker", "Test tracker" };

application.captures.push_back(std::make_unique<CameraCapture>());
application.drivers.push_back(std::make_unique<ArUcoDriver>());
application.senders.push_back(std::make_unique<TUIOSender>());

application.main(argc, argv);
```

Figure 7.3: Code necessary to create a complete tracking application.

[59] `https://github.com/jarro2783/cxxopts`

The `CameraCapture` class can also perform planar rectification of the stream. To do this, it needs to know the quadrilateral area of the input to extract, which can be provided as a JSON file using the "-c" option. While this file can be easily read and produced manually, there's also a small utility called `cropper` provided (shown in section 7.4), which can be used to generate the file interactively.

There are many possibilities for extending the provided classes. We could add a `FileDriver` which reads data from a file, a `HokuyoDriver` which drives the surface using a Hokuyo LiDAR[60], or a `TUIODriver` which receives data over from the network. We could create a `DMXSender` class to drive lights using the tracker, or a `3DSender` to visualize the tracking data in 3D. All of these additions would require minimal (if any) changes to the rest of the program.

```
ArUco object tracker
Usage:
  tracker [OPTION...]

  -s, --surface arg       Surface name (default: HOSTNAME)
  -v, --video_id arg      Video device id (default: 0)
  -w, --video_width arg   Video device width (default: 640)
  -h, --video_height arg  Video device height (default: 480)
  -c, --corners arg       Planar rectification JSON
  -a, --address arg       Server address (default: 127.0.0.1)
  -p, --port arg          Server port (default: 3333)
```

Figure 7.4: Usage help generated by code in Figure 7.3.

7.3 Generator

The attached media also includes a small utility called `generator`, which can be used to generate ArUco markers for an arbitrary number of ID numbers provided. The program is written in C++ and generates the markers using the `OpenCV` library. It will output the markers as PNG files in the current directory.

[60] `https://hokuyo-usa.com/products/lidar-obstacle-detection`

7.4 Cropper

Cropper is a small utility that can be used to interactively generate a JSON file containing the quadrilateral area of the input to be extracted by the `CameraCapture` class. It presents the user with a window showing a camera stream and allows the user to select four points on the image. The points are outputted as JSON to the standard output.

```
[
  { "x": 307.0, "y": 127.0 },
  { "x": 577.0, "y": 172.0 },
  { "x": 596.0, "y": 420.0 },
  { "x": 316.0, "y": 424.0 }
]
```

Figure 7.5: Sample output of the `cropper` utility.

7.5 Dicaffeine

Since NewTek does not currently provide an NDI Monitor application for Linux nor mobile devices, we need to use a third party application to provide this functionality. There's several notable candidates available as of the time of writing, such as Nsm[61] developed by Keijiro Takahashi. We will, however, use an application called Dicaffeine[62], developed by CESNET and FIT CTU's own Jiří Melnikov.

This has several advantages over the alternatives, namely the ability to easily consult the application's author in the event of any issues. It will also allow us to provide feedback on the application's functionality and collaborate on any improvements in the future. The application itself is built using libyuri [38], a framework for audio/video processing developed by Zdeněk Trávníček.

On supported platforms, Dicaffeine can be installed using the package manager. It is configured via a web interface, which will allow us to easily change any necessary settings from the client computer as well. While it is formally still designated as a "beta" version, it runs reasonably well and should be a perfectly adequate solution for the environment of a graphics laboratory.

[61] `https://github.com/keijiro/Nsm`
[62] `https://dicaffeine.com`

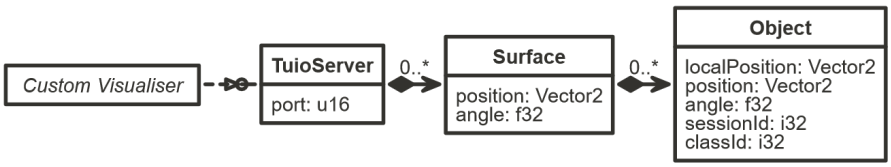


Figure 7.6: Class hierarchy of the SDK's tracking portion.

Additionally, Dicaffeine supports creating virtual devices, which allows us to easily stream from media files, images, generated text or a screen capture. As of the time of writing, Dicaffeine is offered for free without any need for licensing[63].

7.6 Unity Plugin

The central point of the SDK is the Unity plugin – a collection of C# scripts and assets that can be summarily imported into Unity. The Unity plugin is designed to be as simple as possible and relies on existing libraries quite heavily, namely **KlakNDI**[64] and **OscJack**[65], both developed by Keijiro Takahashi. The framework ties these libraries together and provides a high-level API for device specific functionality, as well as reimplementing the TUIO framework using **OscJack**. An overview of the class hierarchy can be found in Figure 7.6.

As we can see in Figure 7.7, the resulting API for object tracking is very simple as it is implemented using the C# event system. The user is notified of any incoming data in an asynchronous manner by providing callback functions. Crucially, these callback functions are called from the Unity main thread, which is the only thread that can safely modify the Unity scene.

This makes the callback functions behave predictably even for inexperienced Unity developers. This is done using a custom dispatch queue encapsulated inside the `TUIOServer` component. The dispatch queue is processed during the `LateUpdate` stage in Unity's frame lifecycle, which is guaranteed to be called after all the other components have been updated. This is done

[63] `https://dicaffeine.com/faq`
[64] `https://github.com/keijiro/KlakNDI`
[65] `https://github.com/keijiro/OscJack`

```
osc = GetComponent<TUIOServer>();

osc.OnObjectAdded += (int sessionId) => {
    var cube = GameObject.CreatePrimitive(PrimitiveType.Cube);
    cube.transform.localScale = Vector3.one * 0.1f;
    cubes.Add(sessionId, cube);
};

osc.OnObjectRemoved += (int sessionId) => {
    Destroy(cubes[sessionId]);
    cubes.Remove(sessionId);
};

osc.OnObjectUpdated += (Object o) => {
    cubes[o.sessionId].transform.position =
        _camera.ViewportToWorldPoint(new Vector3(
            o.position.x, 1.0f - o.position.y, 10.0f
        )
    );
    cubes[o.sessionId].transform.eulerAngles =
        new Vector3(0.0f, 0.0f, -o.angleRad * Mathf.Rad2Deg);
};
```

Figure 7.7: Sample usage of the `TUIOServer` component.

to avoid hard to debug issues arising due to inconsistent order of GameObject updates. Since the actions are only ever enqueued to the dispatch queue from a single thread, any concurrent access protection is not needed.

One crucial piece of functionality provided by the SDK is the ability to receive data from multiple devices and translate them into a single global coordinate system. These surfaces are identified by the name specified in the TUIO `SOURCE` message and are then mapped to the Unity scene inside the `TUIOServer` component. They can be easily configured in the inspector of the `TUIOServer` component (as shown in Figure 7.8). Special fallback surface with the name `*` can be defined, which will accept messages from any device.

As per the TUIO specification, the position of the tracked objects on each surface is given in normalized coordinates. The origin is in the top-left corner of the screen, with the x-axis pointing to the right and the y-axis pointing down. The same coordinate space is used for the surface coordinates as well.

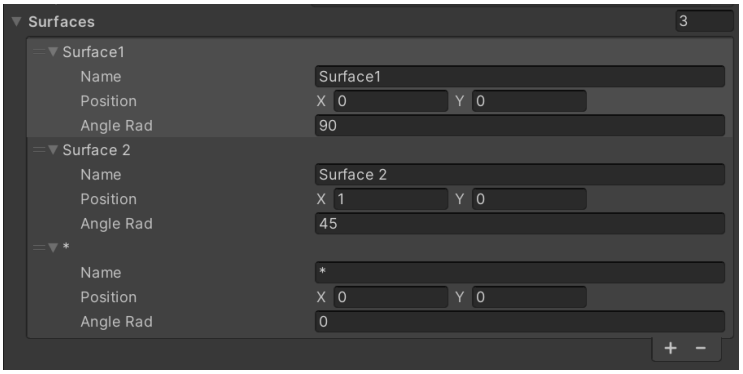


Figure 7.8: List of recognized surfaces in the Unity inspector.

This is something that needs to be taken into account when configuring the surfaces, as the relative position of the surfaces is given in units of surface dimensions. While this somewhat complicates the configuration, it the case of surfaces connected edge-to-edge much more easy. This seems like it's going to be the most common configuration, which makes this a reasonable trade off.

The latter half of the SDK relies on **KlakNDI** pretty much verbatim, as there is not much left to be done to make everything work as we expect it to. This is one of the major benefits of building on an existing technology like NDI and not reinventing the wheel. **KlakNDI** provides two main components: the `NDI Receiver` and the `NDI Sender` components, whose function is pretty self-explanatory. **KlakNDI** is ready to work with different components of Unity, such as cameras or textures, which makes it extremely straightforward to integrate into any existing projects with little effort.

Chapter 8

Applications

In this chapter, we will discuss the various applications that can be built using the SDK. We will explore the potential for video games and data visualization. We will also discuss some examples that are already built using the SDK and available on the storage medium that can serve as starting points for future projects.

8.1 Attractors

A simple physics sandbox has been developed to test the device during development. It accepts any objects from any trackers and represents all of them as a simple sphere. Small particles spawn all over the scene and are attracted to the spheres from a sufficiently close distance. Any number of attractors are supported, and the tracking markers have no effect on the objects placed.

The Unity project as well as the source code for this application is included on the storage medium (as per the listing in Appendix D) in the `attractors` folder.

8.2 Hydropolis

A simple game has been developed for the table built for the Hydropolis installation. The premise is simple – the players place objects in the play area which represent various waterworks present throughout the city, such as storage cisterns or water treatment plants. The goal is to pump water all the

way from dirty water sources to the fields and cities, while treating it along the way. The game is very simple but fosters cooperation between players. We will explore the Hydropolis project a bit closer in chapter 9.

8.3 Chess

One of the most basic computer games is chess. We won't go into too much detail here, as the rules of the game are explained in many places and encompassing them all in this thesis[66] wouldn't make much sense. Instead, let's focus on the mechanics of implementing this game on our device.

The most important part of the game are the pieces. We can use any real set of chess pieces, but we should make sure they are decently large so that we can fit a tracking marker on them. Most chess sets feature pieces with a round base, which would lend itself well to round markers (like BullsEye), or we can find a stylized version of the pieces that are more suitable for square markers.

We can deduce the state of the game from figures present on the screen at any given time. If we want to make the application more interactive – for example displaying the available moves – we can detect when a figure has been lifted and mark it as "selected" and perform a move or capture when it has been placed again. This use case also lends itself quite well to the use of a "virtual board" that is used to represent the state of the game on an external screen for an audience to view during professional chess matches.

8.4 Dungeons and Dragons

Building on the previous example, we will explore the mechanics of the Dungeons and Dragons[67] game. The game is a turn-based strategy game, where the players take turns moving their characters around the board. We won't go into more detail since the various rules for various D&D versions are enough for several books. We will, however, go over some of the mechanics required to design a tangible interface to play the game with.

[66] Looking at you, en passant!
[67] `https://dnd.wizards.com/`

The game is played on a grid of squares, with each square having a number of different types of terrain. We can represent this terrain using a visual map (either created by hand or generated automatically) which can be projected onto the table. The player characters and various other enemies are often represented with physical figurines, which can be placed on the table and tracked from below, same as the chess pieces.

The mechanics themselves will require more human intervention than the chess game, possibly by making a companion mobile app to be used by the dungeon master to change maps, spawn enemies and so on. We can easily use projected map to simulate fields of view, calculate spell ranges and create a sense of immersion greater than a physical map.

8.5 Load Balancing

A very interesting option for employing this device is using it to reason about Kubernetes[68] clusters. Kubernetes is a container orchestration system that allows for the deployment of multiple containers across multiple servers. Among Kubernetes' many capabilities is the ability to move containers between devices to balance the load on the computing resources employed. As with the previous examples, we won't go into too much detail here, as that information is readily available elsewhere.

We could easily visualize our various servers as areas on a single surface, or indeed as different surfaces altogether. Placing an object representing a container onto such area would move the container to the appropriate device. Here we have a very exciting opportunity to start adding more properties to the objects to make them more meaningful. We could use objects with different weights to represent containers with different memory footprints. We could use their shape to denote their requirements, such as the need for a discrete GPU. We could use their color to texture to denote even more properties.

This makes the objects physically meaningful, which is a very important addition over standard digital interfaces. It allows us to employ much more brain power when reasoning about the system, empowers us to collaborate easily and serves as a monitoring tool at the same time.

[68] `https://kubernetes.io/`

8.6 Conclusion

In this chapter, we have explored some ideas for applications that can be developed using our SDK and device. We have shown that the device has great potential to augment existing applications by making them more interactive and easier to reason about, as well as providing a new medium for creating completely new applications. We have seen that it's very easy to prototype and test new ideas quickly, which ultimately fulfills our goal of creating a *flexible* device.

Chapter 9

Case study: Hydropolis

Hydropolis[69] is the name given to a revitalization project for the water tower located in the Vinohrady distric of Prague. The project is conducted by the city of Prague, Pražská vodohospodářská společnost a.s., Pražské vodovody a kanalizace a.s. and the private group Veolia. Under the company st.dio[70], we've developed an interactive table to be included as a part of the educational installation.

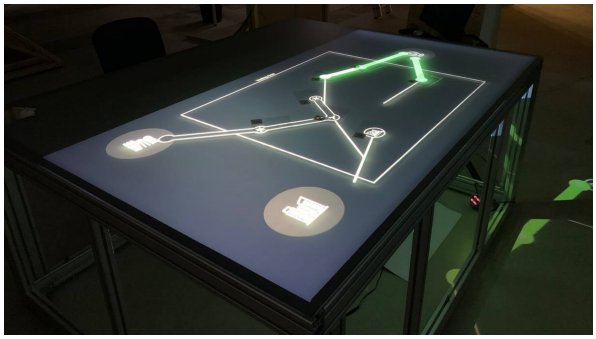


Figure 9.1: Look of the device when projecting in the dark.

Built on an aluminum frame very similar to our revised design, the device features a projection surface with a size of 2 m by 2 m, making it considerably larger than the device we developed during this thesis. This meant that the glass had to be significantly thicker, coming in at 10 mm. In this case, however, we didn't adjust the projector to the square surface at all, instead projecting

[69] http://www.hydropolis-praha.cz/
[70] https://stdio.cz

onto only a part of the surface. While this did create an empty zone on the surface, it served quite well as a space to store the playing cards and therefore wasn't an issue. The surface is made of tempered glass and covered with a projection film, which optimally diffuses the light coming from the projector. This blocks any glare from the projector and provides very good image quality.

Since the table was even lower than our device, the large projection surface meant we had to employ an "ultra short-throw projector" from Epson, which managed to project a 2 m wide image from a very short distance. However, in combination with the extra thick safety glass, this created a significant chromatic dispersion in the projected image, which meant we had to instead replace the projector with several smaller ones.

It's very hard to capture such a large surface with a single camera. While we've had some success with Ximea[71] and FLIR[72] cameras with large sensors, we ran into trouble with the IR sensitivity as well as their very high price. Even then, the extreme angle of view caused similar refraction errors as it did for the projection, not to mention the high distortion caused by such a lens. That meant we settled on capturing the surface with multiple cameras stitched together. As of the time of writing, this system hasn't been finished.

We also tried using an Intel RealSense camera to capture the underside of the table, but were unable to get the markers into focus, resulting in the blurry image seen in Figure 9.2. Sharpening with a convolution was very effective for making the markers recognizable, but it also amplified any noise from the sensor, which in turn increased the jitter in the tracked positions.

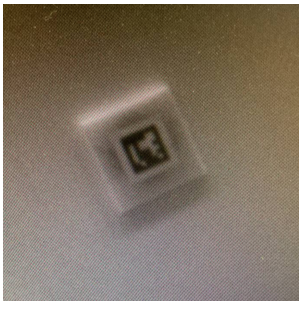


Figure 9.2: Blurry marker seen through the Intel RealSense.

[71] `https://www.ximea.com/`
[72] `https://www.flir.eu/`

The table features a simple game meant to educate especially the younger visitors about the structures involved in the waterworks of Prague. Players pick cards representing each structure from a deck and place them on the table. This creates the structures in the play area. The structures include but are not limited to: a turbine, a water treatment plant and a splitter. Each structure accepts connections from any side and can connect to one other component. The goal is to transport the water from a source to a destination while maintaining the throughput of the network and treating the water along the way.

One of the challenges we've had to solve when working on this project was designing the cards themselves. We settled on a transparent design with markers in the corners of the cards, which allows us to project directly under the cards. This creates a futuristic look as the cards are brought to life by the light from the projector. The only issue we've run into is the burring on the cards scratching the projection surface, which we solved by placing a thin protective layer on top of the projection film.

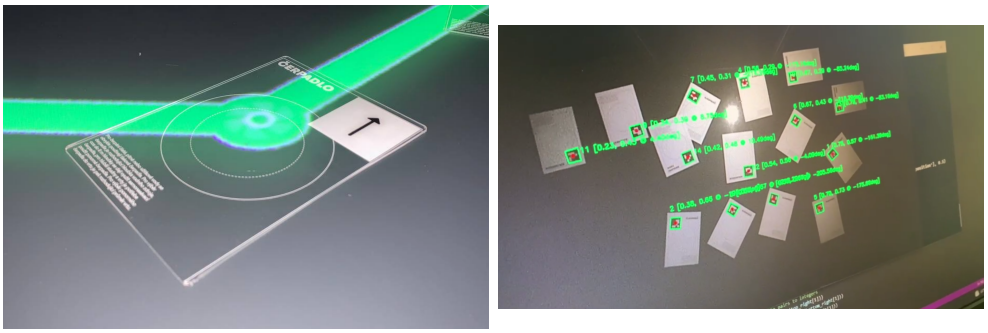


Figure 9.3: A close-up of a card (left) and a camera view (right).

Overall, this project was very rewarding and informed the design of the device we developed in this thesis quite a lot. The device didn't need to be quite as flexible due to the constraints being very well-defined, which enabled us to make a sturdier frame and make more informed choices about the design.

Chapter 10

Conclusion

In this thesis, we have done research on the current state of tangible interfaces with focus on tangible surfaces, specifically the MIT Tactile Matrix. We have also surveyed the landscape of current fiducial marker solutions as well as the viability of using current artificial intelligence techniques in their stead.

Based on this research we have designed, manufactured and assembled the device, discussed design decisions made during to process to help guide future modifications and provided the necessary documentation for replicating the construction. Most importantly, we have expanded on the original tactile matrix design, enhanced its operating range and made the design more flexible, to accommodate the needs of any future project.

Finally, we have designed and implemented an SDK that will allow students to easily create their own applications using the device. We have provided a simple example of how to use the SDK to create a simple interactive experience, and explored a number of ideas for potential applications.

Thus, we can conclude that the goals we have set for ourselves in this thesis have been achieved, and we have a solid foundation for future research. Whether our design will stand the test of time remains to be seen, but thanks to the efforts made to make the design flexible, we should be able to adapt the device to any feedback with little effort.

Bibliography

[1] Tangible Media Group. Tangible Media Group. Accessed on 2022-05-03. Available from: https://tangible.media.mit.edu/

[2] Ishii, H.; Lakatos, D.; et al. Radical atoms: beyond tangible bits, toward transformable materials. *interactions*, volume 19, no. 1, 2012: pp. 38–51.

[3] Trewin, S.; Swart, C.; et al. Physical Accessibility of Touchscreen Smartphones. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '13, New York, NY, USA: Association for Computing Machinery, 2013, ISBN 9781450324052, doi:10.1145/2513383.2513446. Available from: https://doi.org/10.1145/2513383.2513446

[4] Vaz, R. I. F.; Fernandes, P. O.; et al. Proposal of a Tangible User Interface to Enhance Accessibility in Geological Exhibitions and the Experience of Museum Visitors. *Procedia Computer Science*, volume 100, 2016: pp. 832–839, ISSN 1877-0509, doi:https://doi.org/10.1016/j.procs.2016.09.232, international Conference on ENTERprise Information Systems/International Conference on Project MANagement/International Conference on Health and Social Care Information Systems and Technologies, CENTERIS/ProjMAN / HCist 2016. Available from: https://www.sciencedirect.com/science/article/pii/S1877050916324012

[5] Okerlund, J.; Segreto, E.; et al. SynFlo: A Tangible Museum Exhibit for Exploring Bio-Design. In *Proceedings of the TEI '16: Tenth Interna-*

tional Conference on Tangible, Embedded, and Embodied Interaction, TEI '16, New York, NY, USA: Association for Computing Machinery, 2016, ISBN 9781450335829, p. 141–149, doi:10.1145/2839462.2839488. Available from: https://doi.org/10.1145/2839462.2839488

[6] Reactable Systems SL. About us. Accessed on 03. 05. 2022. Available from: https://reactable.com/about-us/

[7] Reactable Systems SL. Reactable. Accessed on 03. 05. 2022. Available from: https://reactable.com/

[8] Microsoft. Microsoft Surface Fact Sheet. 2007, accessed on 2021-05-01. Available from: https://web.archive.org/web/20070614045633/http://www.microsoft.com/presspass/presskits/surfacecomputing/docs/MSSurfaceFS.doc

[9] Microsoft Research. Hands-On Computing. 2009, accessed on 2021-05-01. Available from: https://web.archive.org/web/20090926222526/http://research.microsoft.com/en-us/projects/hands-on-computing/default.aspx#secondlight

[10] Bell, M.; Gleckman, P.; et al. Computer vision based touch screen. Oct. 11 2011, uS Patent 8,035,624.

[11] Bezek, S. SmartKnob. Available from: https://github.com/scottbez1/smartknob/

[12] Bezek, S. DIY haptic input knob: BLDC motor + round LCD. Available from: https://www.youtube.com/watch?v=ip641WmY4pA

[13] Winder, I. System for real-time digital reconstruction and 3D projection-mapping of arbitrarily many tagged physical objects. *Provisional Patents*, 2015.

[14] Winder, J. I.; Larson, K. Bits and Bricks.

[15] Winder, I. Colortizer. Available from: https://github.com/irawinder/Colortizer

[16] Zhang, Y.; et al. *CityMatrix: an urban decision support system augmented by artificial intelligence*. Dissertation thesis, Massachusetts Institute of Technology, 2017.

[17] Bencina, R.; Kaltenbrunner, M.; et al. Improved Topological Fiducial Tracking in the reacTIVision System. 07 2005, ISBN 0-7695-2372-2, pp. 99–99, doi:10.1109/CVPR.2005.475.

[18] Klokmose, C. N.; Kristensen, J. B.; et al. BullsEye: High-Precision Fiducial Tracking for Table-based Tangible Interaction. In *ITS '14*, 2014.

[19] Garrido-Jurado, S.; Muñoz-Salinas, R.; et al. Generation of fiducial marker dictionaries using Mixed Integer Linear Programming. *Pattern Recognition*, volume 51, 10 2015, doi:10.1016/j.patcog.2015.09.023.

[20] Lugaresi, C.; Tang, J.; et al. MediaPipe: A Framework for Building Perception Pipelines. 2019, doi:10.48550/ARXIV.1906.08172. Available from: `https://arxiv.org/abs/1906.08172`

[21] Google. MediaPipe. Accessed on 03. 05. 2022. Available from: `https://google.github.io/mediapipe/`

[22] Zhou, Y.; Tuzel, O. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[23] Simon, M.; Milz, S.; et al. Complex-YOLO: Real-time 3D Object Detection on Point Clouds. *CoRR*, volume abs/1803.06199, 2018, `1803.06199`. Available from: `http://arxiv.org/abs/1803.06199`

[24] Geiger, A.; Lenz, P.; et al. The KITTI vision benchmark suite. *URL http://www. cvlibs. net/datasets/kitti*, volume 2, 2015.

[25] Geiger, A.; Lenz, P.; et al. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, volume 32, no. 11, 2013: pp. 1231–1237.

[26] Kaltenbrunner, M.; Bencina, R. ReacTIVision: A Computer-Vision Framework for Table-Based Tangible Interaction. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*, TEI '07, New York, NY, USA: Association for Computing Machinery, 2007, ISBN 9781595936196, p. 69–74, doi:10.1145/1226969.1226983. Available from: `https://doi.org/10.1145/1226969.1226983`

[27] Kaltenbrunner, M. ReacTIVision and TUIO: A Tangible Tabletop Toolkit. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '09, New York, NY, USA: Association for Computing Machinery, 2009, ISBN 9781605587332, p. 9–16, doi: 10.1145/1731903.1731906. Available from: https://doi.org/10.1145/1731903.1731906

[28] Klokmose, C. N.; Kristensen, J. B.; et al. BullsEye: High-Precision Fiducial Tracking for Table-Based Tangible Interaction. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*, ITS '14, New York, NY, USA: Association for Computing Machinery, 2014, ISBN 9781450325875, p. 269–278, doi: 10.1145/2669485.2669503. Available from: https://doi.org/10.1145/2669485.2669503

[29] Benligiray, B.; Topal, C.; et al. STag: A Stable Fiducial Marker System. *CoRR*, volume abs/1707.06292, 2017, 1707.06292. Available from: http://arxiv.org/abs/1707.06292

[30] Romero-Ramirez, F.; Muñoz-Salinas, R.; et al. Speeded Up Detection of Squared Fiducial Markers. *Image and Vision Computing*, volume 76, 06 2018, doi:10.1016/j.imavis.2018.05.004.

[31] Romero-Ramirez, F.; Muñoz-Salinas, R.; et al. Tracking Fiducial Markers with Discriminative Correlation Filters. 06 2020.

[32] Institute of Electrical and Electronics Engineers. IEEE Standard for Binary Floating-Point Arithmetic. IEEE Std 754-1985, 1985.

[33] Wright, M. J.; Freed, A. Open SoundControl: A New Protocol for Communicating with Sound Synthesizers. In *ICMC*, 1997.

[34] Kaltenbrunner, M. TUIO Protocol Specification 1.1. Accessed on 2022-03-26. Available from: https://www.tuio.org/?specification

[35] Clapham, C.; Nicholson, J.; et al. *The concise Oxford dictionary of mathematics*. Oxford University Press, 2014.

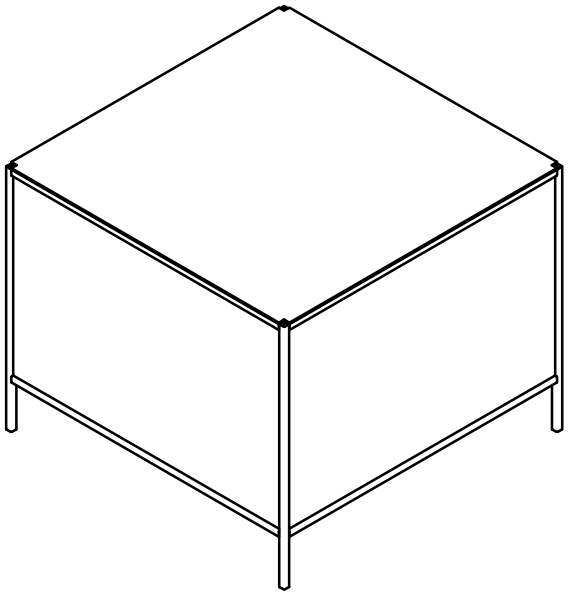
[36] Waldman, C. H. Voderberg Deconstructed & Triangle Substitution Tiling. 2014.

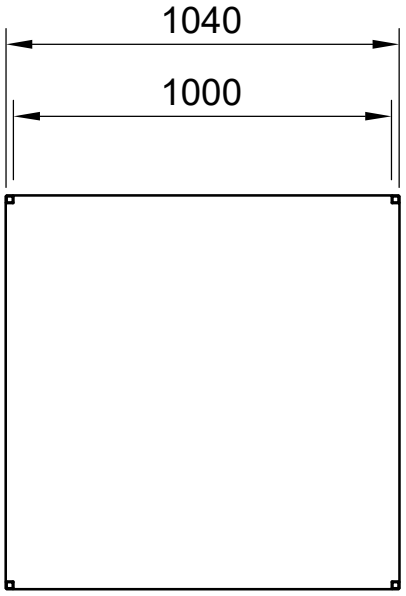
[37] Cífková, R.; Bruthans, J.; et al. 30-year trends in major cardiovascular risk factors in the Czech population, Czech MONICA and Czech post-MONICA, 1985–2016/17. *PloS one*, volume 15, no. 5, 2020: p. e0232845.

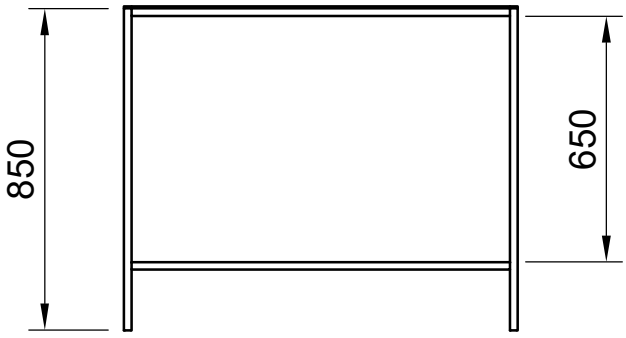
[38] Trávníček, Z. Libyuri framework (in Czech), report, CESNET and Czech Technical University, 2015.

Appendix A

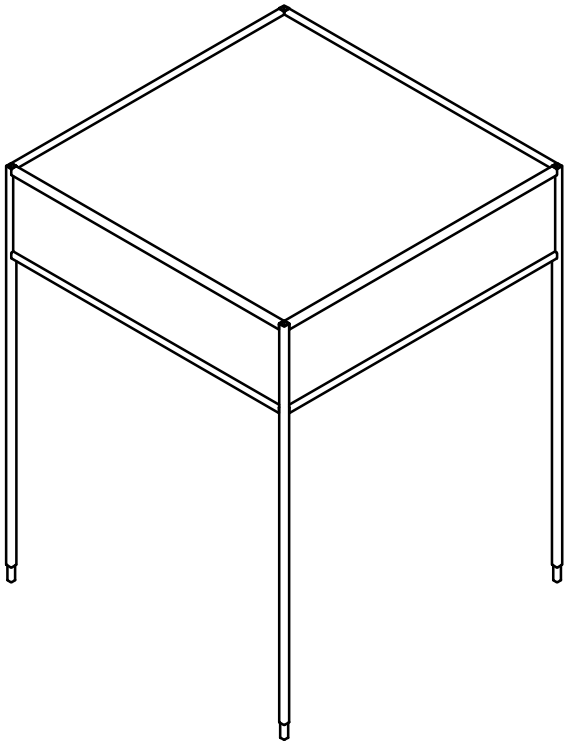
Design Blueprints V1

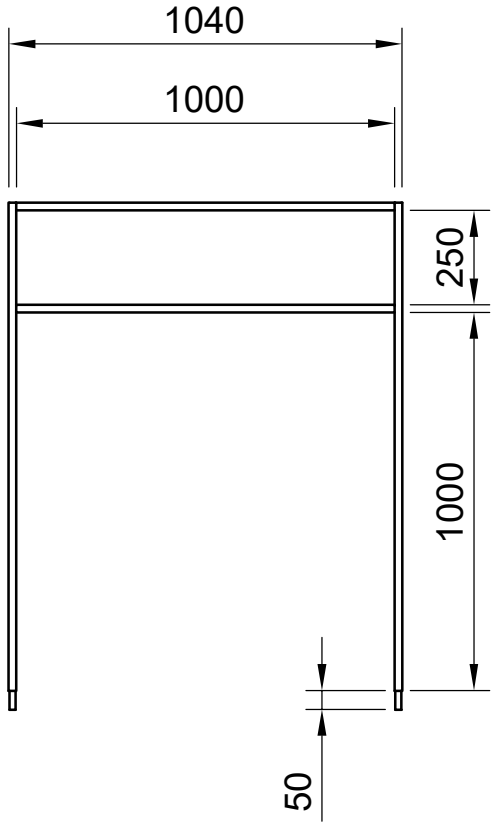




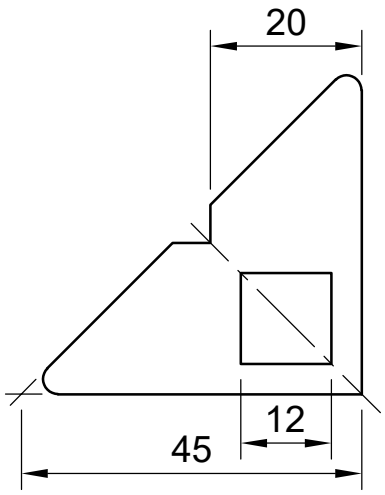


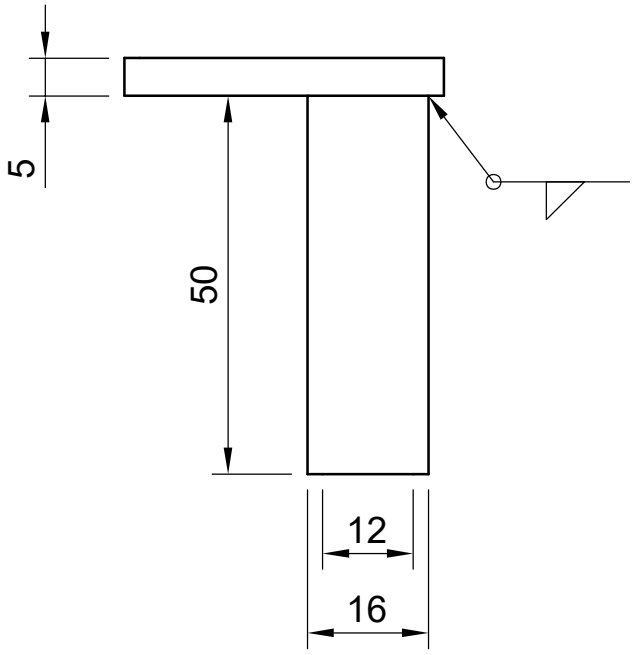
Dept.	Technical reference	Created by Jiří Šebele	Approved by	
		Document type	Document status	
		Title GLab Table Table base	DWG No.	
			Rev.	Date of issue / Sheet 1/1

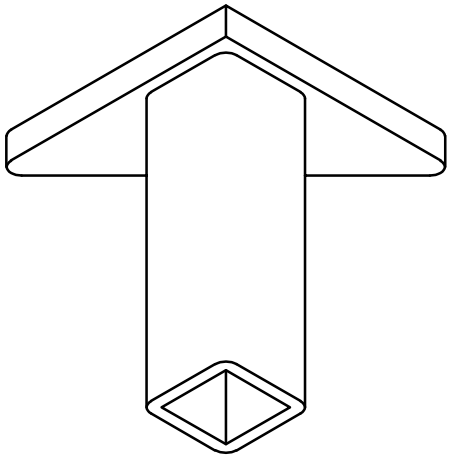


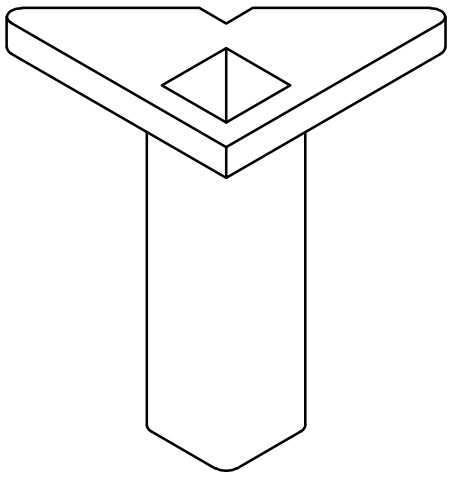


Dept.	Technical reference	Created by Jiří Šebele	Approved by
		Document type	Document status
		Title GLab Table Table hat	DWG No.
			Rev. / Date of issue / Sheet 1/1

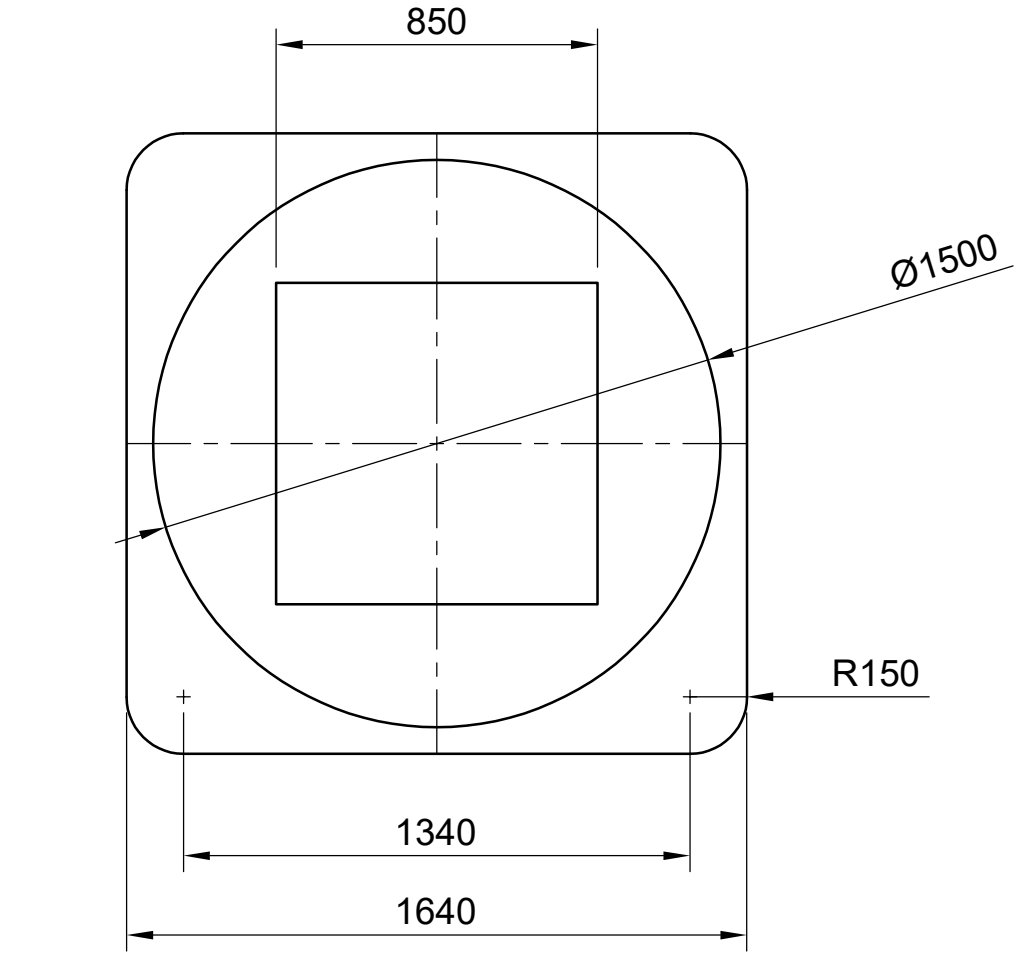


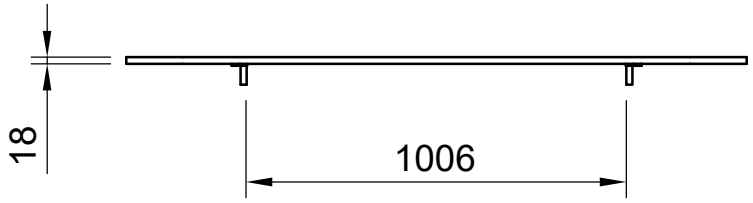






Dept.	Technical reference	Created by Jiří Šebele	Approved by	
		Document type	Document status	
		Title GLab Table Tabletop support	DWG No.	
			Rev.	Date of issue / Sheet 1/1

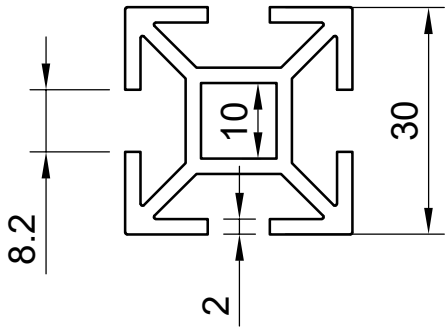


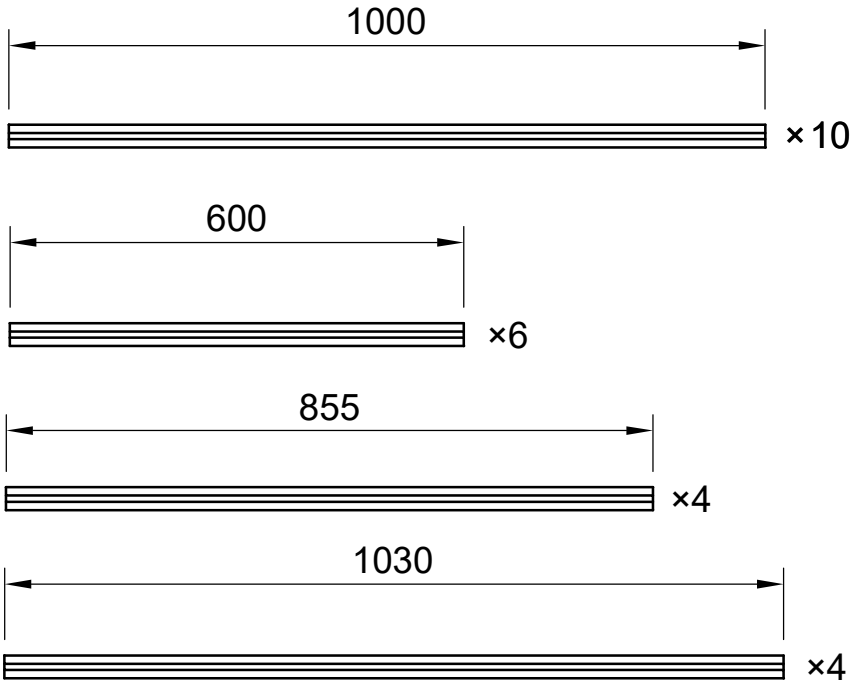


Dept.	Technical reference	Created by Jiří Šebele	Approved by	
		Document type	Document status	
		Title GLab Table Tabletop	DWG No.	
			Rev.	Date of issue / Sheet 1/1

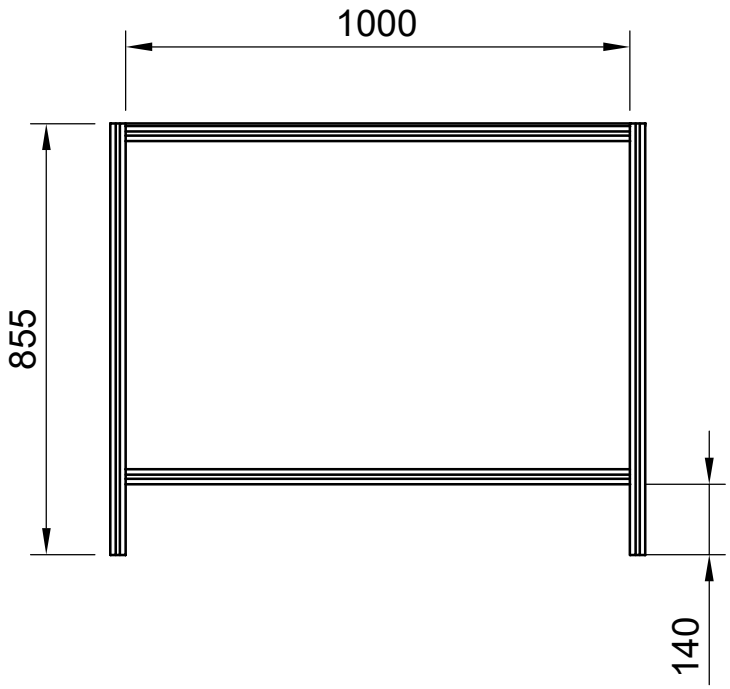
Appendix B

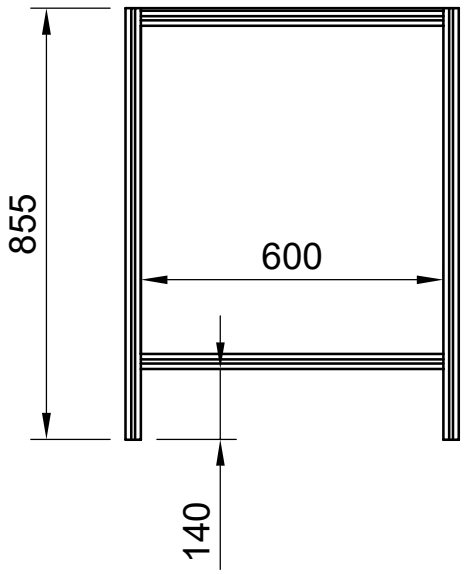
Design Blueprint V2

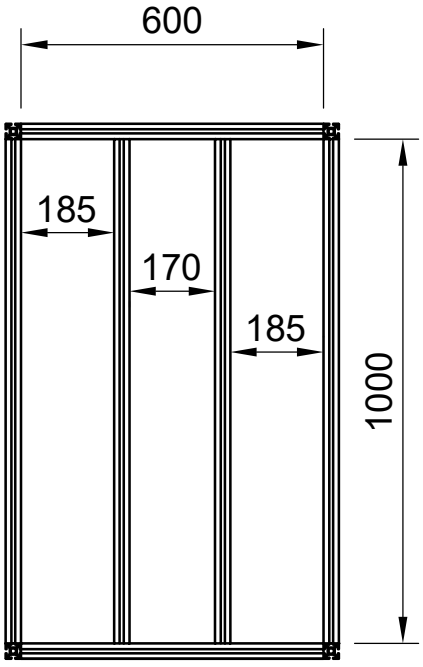


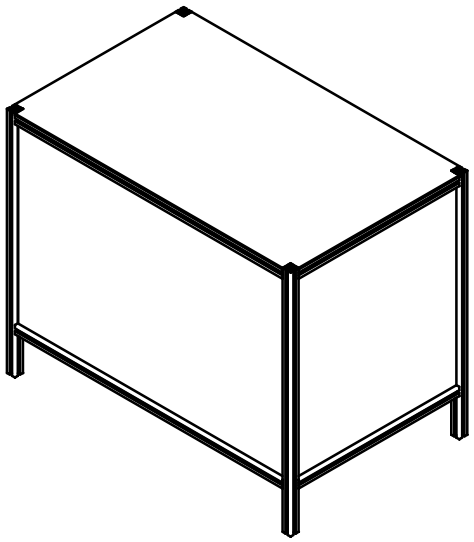


Dept.	Technical reference	Created by Jiří Šebele 03.05.2022	Approved by
		Document type	Document status
		Title: GLab Table v2 Material	DWG No.
			Rev. / Date of issue / Sheet 1/1

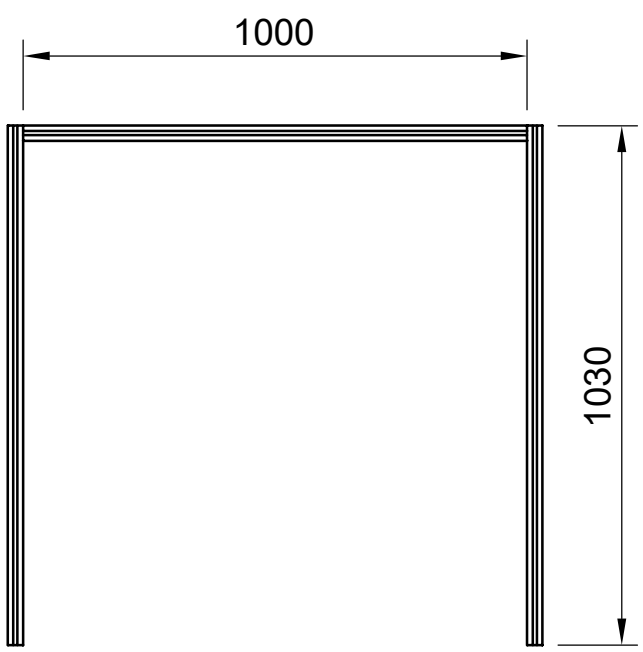


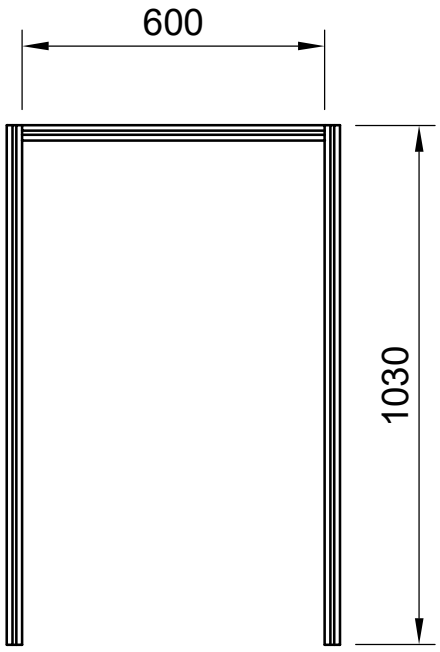


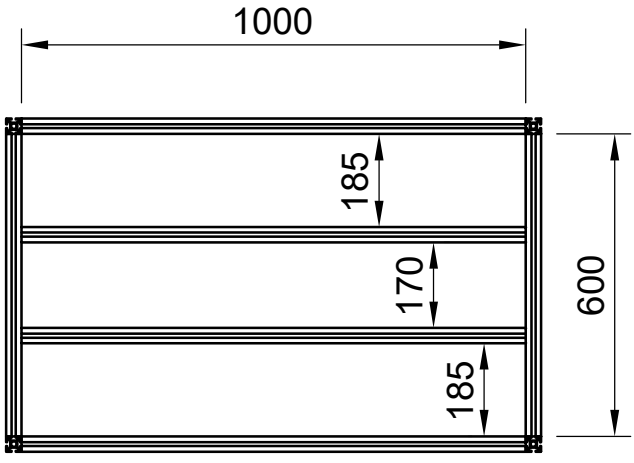


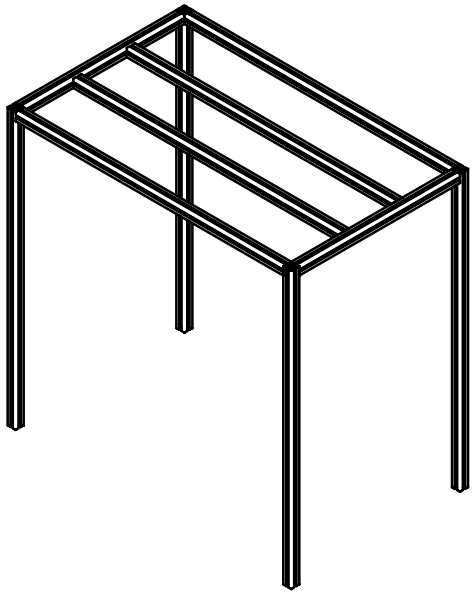


Dept.	Technical reference	Created by Jiří Šebele 03.05.2022	Approved by
		Document type	Document status
		Title: GLab Table v2 Base	DWG No.
			Rev. / Date of issue / Sheet: 1/1

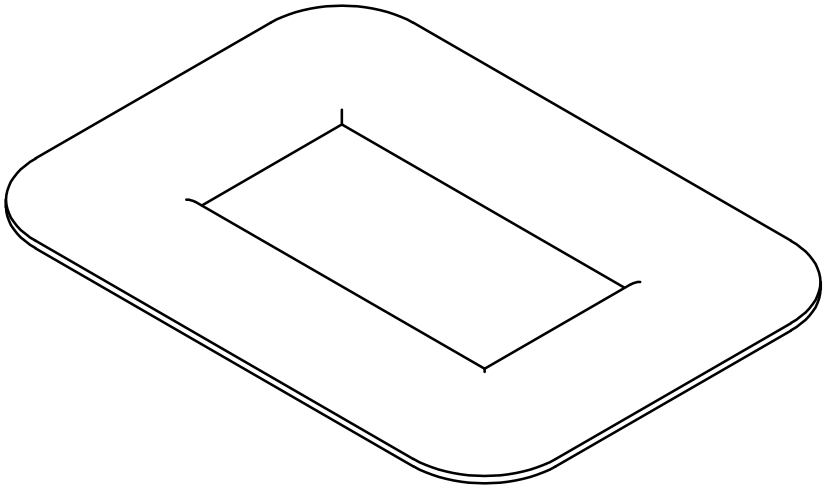


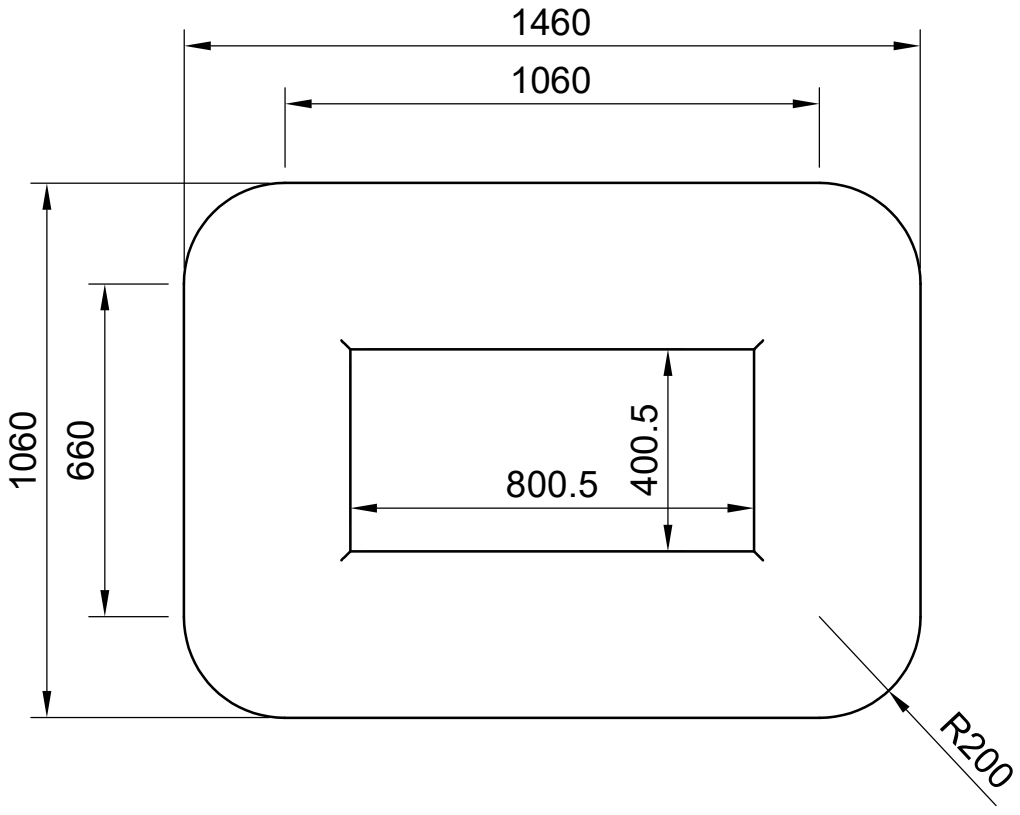




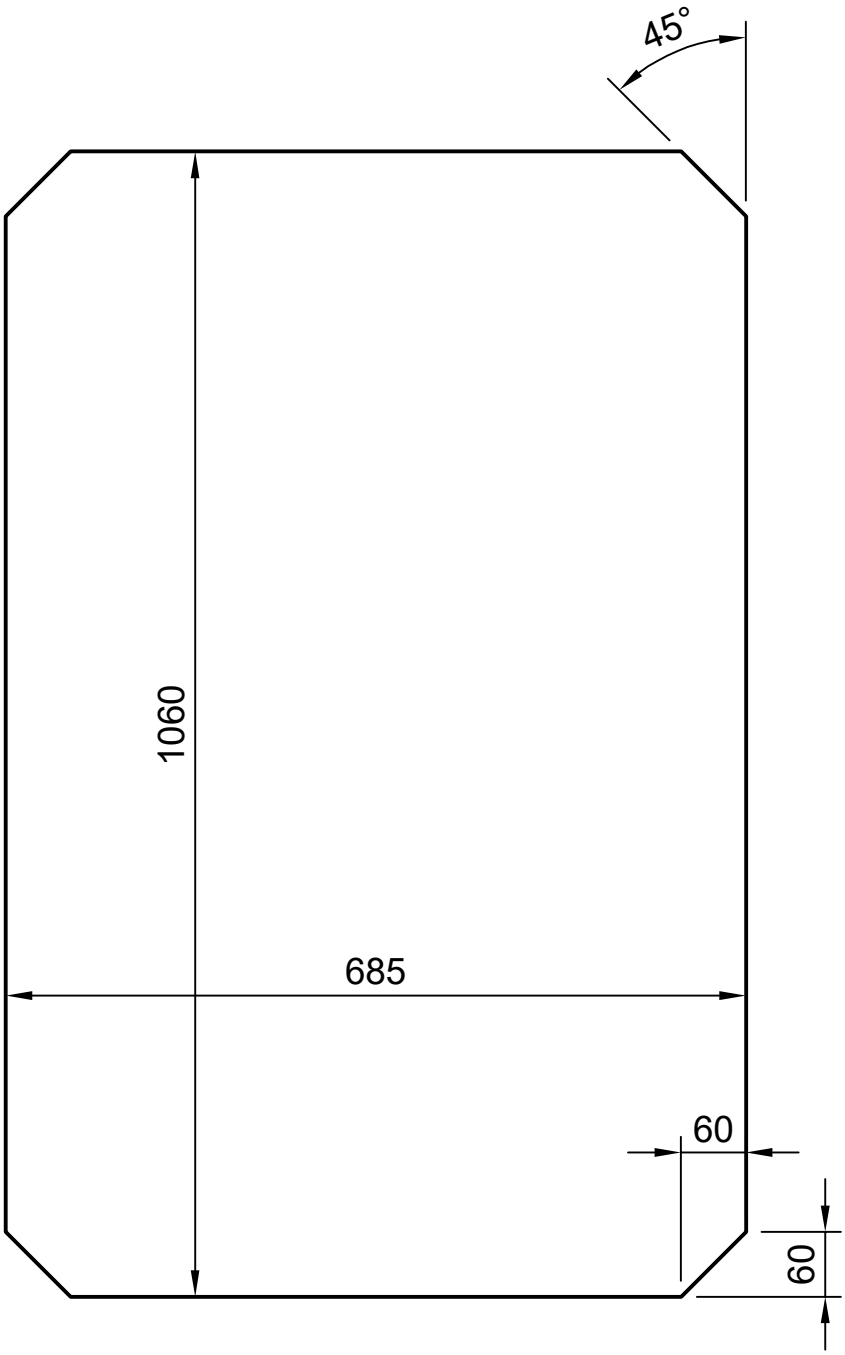


Dept.	Technical reference	Created by Jiří Šebele 03.05.2022	Approved by
		Document type	Document status
		Title: GLab Table v2 Hat	DWG No.
			Rev. / Date of issue / Sheet 1/1





Dept.	Technical reference	Created by Jiří Šebele 03.05.2022	Approved by
		Document type	Document status
		Title GLab Table v2 Tabletop	DWG No.
			Rev. / Date of issue / Sheet 1/1



Dept.	Technical reference	Created by Jiří Šebele 13.04.2022	Approved by
		Document type	Document status
		Title Projection surface	DWG No.
			Rev. / Date of issue / Sheet 1/1

Appendix C

Acronyms

MIT Massachusetts Institute of Technology
GUI Graphical user interface
XML Extensible markup language
OSC Open Sound Control
MIDI Musical Instrument Digital Interface
NDI Network Device Interface
PCB Printed Circuit Board
LCD Liquid Crystal Display
PoE Power over Ethernet
MDF Medium Density Fiberboard
CAMP Centrum Architektury a Městského Plánování
LCD Liquid Crystal Display
FOV Field of View

Appendix D

Supplemental Material

The complete source code of this thesis and the projects described within as well as any technical drawings can be found on the attached medium.

```
README.md ..... Media contents description
thesis ..... LaTeX source code of the thesis
pdf ..... Rendered PDFs of the thesis
    thesis.pdf ..... Thesis text in PDF format
    thesis_print.pdf ..... Thesis text in PDF format without hyperlinks
design_v1 ..... Design files for version 1
design_v2 ..... Design files for version 2
sdk ..... SDK source files
    src ..... C++ source files
    plugin ..... SDK Unity plugin
    attractors ..... Attractors example Unity project
```