Augmenting play rooms with intelligent digital playmates

David Niggli

Master Thesis
November 2016

Prof. Dr. Robert W. Sumner
Dr. Steven Poulakos
Dr. Martin Guay

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

cgl
computer graphics laboratory

Abstract

This master's project aims to create an augmented digital character that enhances a child's creativity when playing with toys. The main challenge is to give the character knowledge about the environment, the toys and the possible interactions with both the toys and the child. All these challenges are research topics in them selves. In order to create a system that succeeds in solving them, we explore and combine cutting-edge technologies. Our result is an Augmented Reality application running on the Microsoft Hololens in which a digital character visualized within the playing space responds to user action and speech.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

AUGMENTING PLAY ROOMS WITH INTELLIGENT DIGITAL PLAYMATES

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
NIGGLI	DAVID

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 16 November 2016	D. Niggli

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Disney Research

cgl
computer graphics laboratory

Master's Thesis

Augmenting play rooms with intelligent digital mates

David Niggli

Introduction

"True play is creativity and creativity is play"
–John Lasseter, Principal Creative Advisor, WDI.

The goal of this project is to augment a child's playroom (filled with toys) with a digital playmate capable of interacting with the environment and child. Digital characters would allow movements unfeasible in the real world, as well as stimulate the child's creativity through interaction.

Creating an intelligent autonomous character aware of its environment and capable of interacting with children is a tremendous technological challenge that intersects multiple (traditionally separate) areas of computer science: AI, computer vision, computer graphics and animation. Realizing this concept involves solving problems such as localization, object recognition, speech recognition, and animation. While each of these problems today have individual solutions, they require engineering skills to setup and combine together, and are thus unavailable to designers and artists. Our solution is to devise an intuitive platform for specifying intelligent autonomous characters that encapsulates state-of-the-art AI, computer vision and animation technologies.

Task Description

The first goal is for the character to become aware of its environment (localization). We aim to realize a dynamic 3D scanning solution similar Microsoft's Holoportation [1.1]. A first step will use Kinect fusion and scan the environment for static 3D geometry. (Eventually we would need to support dynamic environments). The second goal is to understand its environment. We will use Google's deep neural network (DNN) technology "TensorFlow" [2] for image-based object recognition. Our challenge will be to allow designers to easily specify the type of knowledge the digital mate has. Current image-based object recognizers can learn up to 100 different object categories. The third goal is for the character to communicate with the user. For example, it could recognize a car, and ask "Ah, what will you do with this car?". For the user to talk back, we can use Google's speech-to-text solution [3], or the user could also communicate using gesture recognition [4].

Milestones

Task	Due date
Start	**16-Mai-2016**
Localization (1)	**1-June-2016**
Object recognition (2)	**1-Aug-2016**
Animation (motion graph) (3)	**1-Sep-2016**
Speech-to-text or gesture interaction (4)	**1-Oct-2016**
First complete thesis draft	**15-Oct-2016**
Final implementation and thesis	**16-Nov-2016**
Thesis presentation	**TBA**
Hand in printed thesis and CD	**TBA**

Remarks

A report and an oral presentation conclude the thesis. It will be overseen by Prof. Bob Sumner and supervised by Martin Guay, with support from other animation group members.

References

[1] Kinect Fusion
https://msdn.microsoft.com/en-us/library/dn188670.aspx
[1.1] Holoportation from Microsoft for localization.
http://research.microsoft.com/en-us/projects/holoportation/
[2] Image-based classifier: google tensor flow.
https://www.tensorflow.org/
[3] Google speech-to-text:
https://cloud.google.com/speech/?gclid=CJfqvpXhrswCFcrjGwodrcoO4g
[4] Hand gestures using open CV and cameras:
http://www.intorobotics.com/9-opencv-tutorials-hand-gesture-detection-recognition/

A *ludobot* is a type of artificial human companion: an entertainment robot, from Latin *ludo* (play) and *bot* (robot). https://en.wikipedia.org/wiki/Ludobot

Acknowledgment

I am grateful to my supervisors Steven Poulakos and Martin Guay, who spent a lot of time during our meetings to support me when I encountered difficulties. I would also like to thank Professors Robert W. Sumner and Markus Gross for giving me the opportunity to write my master's thesis at Disney Research Zurich. Last but not least I would like to thank my family and various friends for their moral support during these six months.

Contents

List of Figures

List of Tables

1

Introduction

1.1 Motivation and Objectives

The aim of this master's project is to augment a child's playroom with an intelligent digital character that enhances the playing experience. The three main challenges are to make the character aware of the environment, understand it and interact with it as well as with the child. To tackle these challenges, we break them down into well defined goals.

Environment Awareness. The first challenge is for the character to become aware of its environment. We want to enable navigation through the physical space, which requires scanning the geometry of the real world surroundings and using a navigation algorithm to find path trajectories for the character to walk on. We explore technologies such as Kinect Fusion and project Tango for 3D reconstruction but in the end we settle for the Hololens. The device is worn on the head and thus frees the user's hands for other forms of interaction. We use a Unity asset called A* Pathfinding Pro which implements the A* algorithm, that we integrate into the rest of our system by designing a custom interface.

Environment Understanding. The second challenge is for the character to understand the environment. We want to give our system knowledge about the toys that are available in the real world for the child to play with. For this we need to first detect and identify them, and then localize them in space and track them over time. We explore the possibility of using 3D reconstructions to take pictures with a virtual camera for object recognition, but then we settle with using webcam images instead. We implemented image based object recognition using both Tensorflow and Faster RCNN, which are leading technologies in deep learning.

Interaction. The third challenge is to have the character perform a wide range of interactions

with the child and the recognized objects. Forms of interaction include gazing, speaking and gesturing, as well as placing the physical toys. Speech to text can make the character understand what the user says and text to speech allows the character to speak; this helps us to achieve a natural form of communication. Since we also want the character to respond to the child's play with the toys, we need to create some artificial intelligence and realistic behavior. We aim to define story arcs that the character can execute while choosing the participants at runtime depending on the available toys. For this we explore the use of an Interactive Behavior Tree-like design, which is a way of controlling the character's behavior that allows us to decouple the evaluation of the system's state and the user's interactions from the story arc execution.

Our result is an Augmented Reality (AR) application running on the Hololens, in which a digital character is visualized within the playing space as a hologram and responds to the child's interactions with the toys, its gaze, speech and gestures. To the best of our knowledge this thesis presents the first system that uses image based real-world object recognition for AR purposes and allows for story arcs that can associate involved participants at runtime and also be interrupted and resumed. Our contributions are the functionality to interrupt and resume story arcs, as well as setting event participants at runtime, depending on the toys that are present in the scene.

1.2 Outline

After this introduction we have the following chapters: Related Work, Environment Awareness, Environment Understanding, Interaction, Results, Limitations and Future Work, and Conclusion.

In Related Work we describe how other projects relate to ours, what we learned from them and what the differences are. In Environment Awareness we discuss how we scan the room's geometry and use it for the character to navigate and find complex paths around obstacles. In Environment Understanding we specify the design and implementation of the object recognition and tracking module. In Interaction we describe in more detail how our system controls the character, how the character's behavior is programmed and how stories can be authored. In Results, Limitations and Future Work we discuss what we achieved, where there is room for improvement and what we plan for the system in the future.

2

Related Work

In this chapter we go through some of the related work, most of which lie in areas where we needed to find existing technology to solve all of the individual challenges of this master's project.

2.1 Environment Awareness

One of the challenges is to make the character aware of the environment. This means that the character grasps the spatial properties of the room. For this we need to give the system knowledge about the room's geometry which it then uses to perform pathfinding. This allows the character to navigate between places.

2.1.1 Spatial Mapping

The first part of environment awareness is to capture its geometry. For this we use modern 3D reconstruction techniques.

Developing Augmented Reality applications with Unity 3D and Vuforia. Vuforia [Mor15] is a technology used in AR applications to track the camera's position and place digital entities by using texture on surfaces. The AR experience is limited to an area in which this pattern is visible for the AR device's camera. Our AR application is built in Unity and we use Vuforia in the beginning of the project before moving on to other AR technologies.

KinectFusion: Real-Time Dense Surface Mapping and Tracking. KinectFusion [RAN11] presents a system for accurate real-time mapping of complex and arbitrary indoor scenes in

variable lighting conditions. Our need for a real-time mapping of the environment makes this work relevant for us. Of course we do not want to require the user to walk around with a Kinect. We considered using a Kinect to scan a static mesh of the room once, which we could then later use in our application. But since we want to allow our character to interact with a dynamic environment, we did not not settle for this approach.

3D Tele-Immersion Platform for Interactive Immersive Experiences between Remote Users. In this project [ZAD+16] the use of several Kinects enables real-time 3D reconstruction of users and their placement into a pre-authored 3D environment. 3D reconstruction is a hot research topic in itself and yet we need it as a building block for our project. The setup required for this kind of realtime 3D reconstruction is very complex and hence not suited for our AR application. Furthermore this technology aims to reconstruct a user's geometry, whereas we need to achieve a spatial mapping of an entire room.

Tango. Project Tango [Tan] is being developed by Google to create smartphones and tablets that are aware of the surroundings' geometry and are able to track the user's position and movements. This technology is very promising for our application. One limitation of using tablet devices for AR is that the user's hands are required to hold the device, whereas we would like to have the user's hands free for interaction with the toys.

Hololens. Hololens [Hol] is Microsoft's new AR device designed to place program windows and/or digital objects in the home environment. It provides not only tracking of the user's position and movements, as well as a spatial mapping of the environment's geometry, but it also allows the user to use his or her hands freely since it is worn on the head.

2.1.2 Pathfinding

The character must be aware of obstacles and find a way around them. This involves finding complex paths on the room's surfaces that fulfill given constraints, like the slope that the character can climb or the radius of collision.

A* Pathfinding Pro. A* is an algorithm that is widely used in pathfinding and graph traversal. We bought the Unity asset called A* Pathfinding Pro which implements this algorithm.

2.2 Environment Understanding

Understanding the environment means to distinguish foreground from background, in order to detect meaningful entities.

2.2.1 Image Based Object Identification

Because we want our application to respond to a child's play with toys, we want it to recognize toys. We aim to recognize all toys without needing any form of preparation. Machine learning is a very strong tool because once a model is trained on enough data it can recognize objects of a very general category.

Tensorflow. Tensorflow [AAB+15] is Google's Machine Learning API and it can be used to classify images across a thousand different categories. Because we want to use image-based object-identification for our digital character to be aware of the toys in the room, we explored this technology. We need to recognize several objects within a single image, but using Tensorflow only allows us to label an image as a whole. This implies that if there are two objects in an image, we are not able to detect them both, and the position of the recognized object within the image is also not known. Therefore we need to first detect individual objects before identifying them.

Caffe. Object recognition is a research topic and recent progress made in machine learning enables machines to identify objects in images with an accuracy defying human performance.Caffe [JSD+14] is a deep learning framework which can be used for image based object recognition.

Faster R-CNN: Towards Real-Time Object Detection With Region Proposal Networks. Faster Region-based Convolutional Neural Network [RHGS15] is a Caffe based technology to simultaneously enable efficient region proposals and object identification in images. We use this technology, because it enables us to detect multiple objects within a single image and also to localize them in space by using their bounding boxes..

Microsoft's CNTK. This deep learning framework by Microsoft could be used in future work to facilitate integration of the image processing functionality on the Hololens. That is, once the device has enough computing power to handle it.

2.2.2 Object Localization

Once an object in an image is recognized, its 3D position in the real world still needs to be computed.

Learning Spatial Object Localization from Vision on a Humanoid Robot. This work [LHFS12] presents a combined machine learning and computer vision approach for robots to localize objects. It allows a humanoid to quickly learn to provide accurate 3D position estimates of objects seen. This work is about how to use machine learning and genetic programming in order to achieve stereo vision without having to calibrate the robot's cameras. This work relates to ours, because we also want to recognize and localize toys' coordinates in 3D space. However, since our humanoid is digital and possesses no cameras in the real world, the covered method

is not applicable to our application.

2.3 Interaction

To provide the user with an immersive experience, the system must be interactive and present a variety of possible events.

2.3.1 Augmented Reality

AR is a fast growing concept and a lot of effort is put into it, resulting in very inspiring related work concerning this area.

Augmented Creativity: Bridging the Real and Virtual Worlds to Enhance Creative Play. This work [ZRM+15] proposes the concept of Augmented Creativity as employing AR on modern mobile devices to enhance real world creative activities, support education, and open new interaction possibilities. It presents examples of AR applications to achieve this goal. Our application aims to enhance a child's play with toys, keeping its hands free for interaction.

2.3.2 Speech Interaction

For an application to be truly interactive, speech recognition is a requirement, as speech is an important form of communication between a user and a digital character. We want to enable the character with speech recognition and synthesis.

Animated Agents Capable of Understanding Natural Language and Performing Actions. Kairai [TTS04] is an interactive system with virtual objects and agents which follow instructions given by the user, like "push the blue ball toward the horse". This work is related to ours, because we also want to allow the user to interact with the digital character. Giving him instructions on what to do next is one way of interacting. However, the Kairai system is limited to predefined scenes, objects, agents and actions, which are all entirely digital, whereas in our AR system the character needs to interact with real world objects, and the knowledge about them, like their positions, is not known a priori. In addition we expect our character to perform a large variety of actions.

Our work is not mainly about Natural Language Understanding (NLU), which is, despite being old, still a debated research topic. We will therefore be left with similar limitations as the Kairai system. But progress is being made in that area and since the long term vision of our project is to have a fully conversational character, we need to implement a modular interface which allows full compatibility between a future NLU Module, focusing entirely on conversation between the digital character and the user, and the rest of the system, meaning the character's actions that

result from such conversations. Even though our project tackles several more challenges than the Kairai system, we can nevertheless still learn a lot from it and apply some of the described concepts. We showcase conversational interaction by having the user give instructions to the character. And similar techniques are applied: words are mapped to objects in the scene and actions that can be performed. One challenge for example is to make the system understand what the word "it" refers to in a later sentence. For that, grammatical rules are used to track entities that were previously mentioned.

SyntaxNet. SyntaxNet [Syn] is an open-source neural network framework implemented in TensorFlow that provides a foundation for Natural Language Understanding (NLU) systems. Future work in our project could make use of this framework to enable natural conversation between the user and the character. If such a conversation module can be decoupled from the rest of the application, it could be used to find out what the user wants through complex communication and then access the rest of the system to execute the required actions. It could also permit the system to learn over time, using the user's speech as teaching method.

2.3.3 Physical Interaction

The digital character should be animated and movements should reflect interaction with the toys, as well as the user.

ADAPT: The Agent Development and Prototyping Testbed. ADAPT [KFS+16a] is a flexible platform for designing human characters in a rich virtual environment. It incorporates character animation, navigation and behavior. In our system, we use a modified version of ADAPT called KADAPT.

PRECISION: Precomputing Environment Semantics For Contact-Rich Character Animation. PRECISION [KXN+16] presents an automated approach for analyzing both motions and environments in order to represent the different ways in which an environment permits a character to move. Future work for our project could integrate such functionality to enable the character to perform complex animations like climbing up tables.

Creatures. Creatures [GCM97] is a commercial home-entertainment software package that provides virtual agents with which the user can interact in real-time. A creature's behavior is the result of a decision making simulation which takes into account things like the visual field (entities that it sees), drives (like growing or appeased hunger), or user interactions (like mouse stroke), and past experience (which allows positive or negative behavior reinforcement). This work relates to ours, as we also create a digital agent that should have some realistic behavior. Unfortunately, this simulated system is very complex and we did not have the time to focus on it, but adding some behavioral simulation for the character could be considered in the future to make our interactive character more human-like.

2.3.4 Storytelling

The system should allow for designers to author complex stories that the character can execute depending on recognized toys, so that the system does not get boring for the user. The application should be expandable with newly designed user experiences.

Narrative in Virtual Environments - Towards Emergent Narrative. Emergent Narrative [Ayl99] considers the clash between the confining bounds of a pre-scripted character in a precise narration and the freedom afforded by a Virtual Environment (VE). In a movie, for example, the user is, in a sense, to see the specific image on the screen, which are predefined by the story plot. Therefore the story is scripted in advance to be made interesting and the visual effects can then be generated from the story. But in a VE, one has no a priori knowledge about what the user will be looking at. This work relates to our project because in augmented reality the user's perspective is also not known in advance.

Of course, many of the challenges we have in our AR application, like recognizing toys, are not present in a VE, but we can still use some of the concepts on a storytelling level, even if their resulting application is only interesting for children with very low age. The concept of emergent narrative is that it is much more interesting to let the user participate in the storytelling process, by creating a story bottom up, starting with the user's interactions, creating short term goals from there, and finally fitting everything into a larger story arc. Our digital agent is meant to be a playmate, hence he should at the same time respond directly to the child's interactions, as well as encourage creativity to bring a more complex story to life.

IBT: Computer-Assisted Authoring of Interactive Narratives. Interactive Behavior Tree [KFZ+15] presents a design formalism that enables content creators to easily author complex narratives while empowering players with the agency to freely interact with the characters in the story. We use this design in our project to have the character perform story arcs that depend on user input.

CANVAS: Computer-Assisted Narrative Animation Synthesis. CANVAS [KFS+16b] provides an accessible, yet expressive interface for story authoring that enables the rapid prototyping, iteration, and deployment of narrative concepts. Such a system could possibly be used to create story content for our application. One challenge is to enable authoring story arcs with dynamically chosen participants and to export the resulting behavior trees in a format that can be used in our application. Future work in our project could integrate more of the CANVAS functionality into the system to facilitate such compatibility.

3

Environment Awareness

In this chapter we discuss the challenges of making the system aware of the environment. Firstly, this involves tracking the position of the AR device's position to render the augmented reality on the screen in such a way that it blends realistically into the real world. Secondly, we need to map the environment's geometry to give the character spatial knowledge. Finally, we need to achieve pathfinding in an ever changing, dynamic environment, in order for the character to find a way around obstacles and move realistically through the physical space.

3.1 Camera Tracking

Since our final result is an AR application, we need to use a device capable of rendering the augmented reality on top of the real world. To get familiar with AR development, we started by creating an AR App running on a tablet, using a technology called Vuforia to track the camera position and to define a flat surface on which our character can walk (see Figure 3.1 (a)). This technology has the limitation that a printed pattern needs to be in the camera's visual field in order for the tracking to work.

We later gained access to Google's Tango device, which is able to track the devices position without such a pattern (see Figure 3.1 (b)). Finally, we ended up deploying our AR application on Microsoft's Hololens, which tracks the user's position with high accuracy.

(a) Vuforia. (b) Tango.

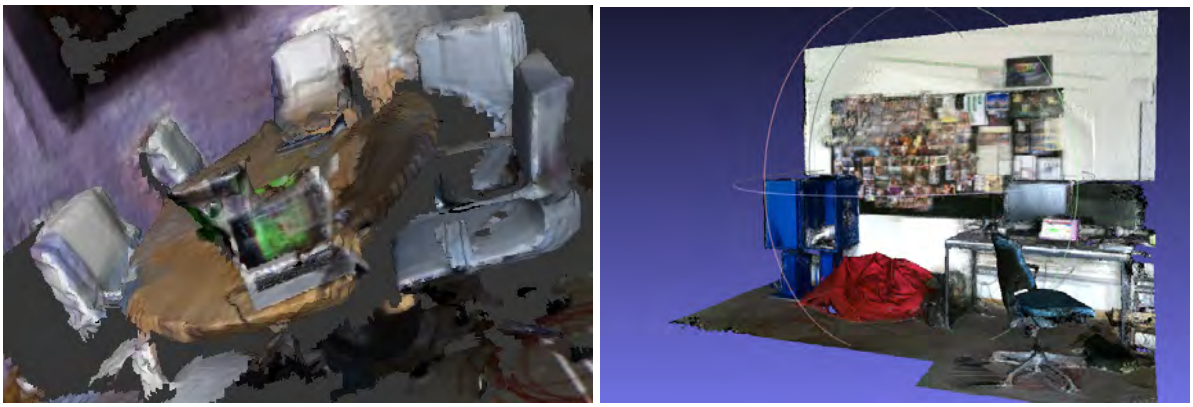
Figure 3.1: *Camera Tracking Using Vuforia (a) and Tango (b).*

3.2 Spatial mapping

Several modern technologies allow for 3D reconstruction. If our digital character is to truly blend into the real world environment, our system needs to have knowledge about the boundaries of the area the character can exist in and the surface it can walk on. The character should neither float in the air nor walk into walls. To obtain a mesh of the physical environment we tried a few of these technologies.

Microsoft Kinect is the first option we considered. The device is very popular and widely used in combination with the well known game console Xbox. It is very good at recognizing humans and the poses they are in. Kinect Fusion [RAN11] is a technology that provides 3D object scanning and model creation using the Kinect device. Our first aim was to scan an environment once, in order to have a static mesh of the room, which we can then use in our application for the character to gain knowledge about the room's geometry. We tried to achieve a 3D reconstruction of the room using Kinect 360 and Kinect One (see Figure 3.2). One limitation of this approach is that we are not able to update the character's knowledge once the application is running and can hence only have him interact with a static environment.

Because we want a dynamic reconstruction of the room's geometry without requiring the user to actively scan it, we thought of simultaneously using several Kinects placed at different loca-

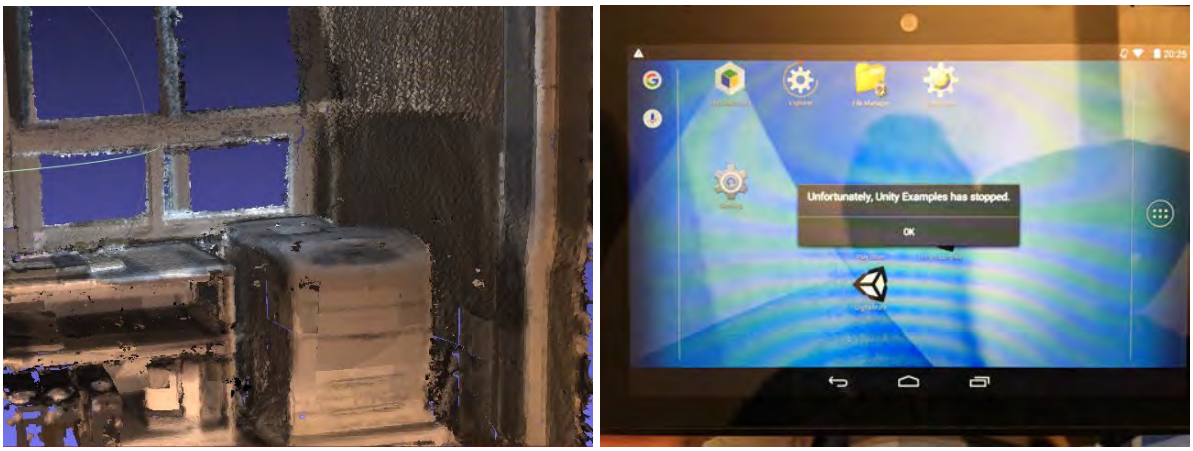


(a) Scan With Kinect 360. (b) Scan With Kinect One.

Figure 3.2: *3D Reconstructions using Kinect 360 (a) and Kinect One (b).*

tions in the room in order to have depth information from different perspectives. We stumbled upon a project that performs a real-time 3D reconstruction of a player in a game, using multiple Kinects [ZAD+16], but since the setup is complex and requires calibration and because we need a mapping of an entire room rather then the reconstruction of a single object, this technology is not ideal for our application.

Google's Tango device is able to scan the environment's geometry dynamically (see Figure 3.3 (a)). Unfortunately, the performance of the reconstruction depends a lot on the chosen voxel resolution and the scanning module crashed regularly even when running Tango's official Unity example project without altering it (see Figure 3.3(b)). Furthermore, the use of a tablet binds the user's hands which we would like to use for interaction purposes.



(a) Scan with Tango. (b) Tango's Example Application Crashing.

Figure 3.3: *Tango's 3D Reconstruction (a) and Limitations (b).*

We considered a few other applications used to create 3D models using just a normal camera, but since we need realtime scanning of a large surrounding environment rather then a reconstruction

of a single object, and because we need a solution that is integrated into our application, none of them were ideal for our application. Fortunately, we gained access to one of Microsoft's Hololens devices. This is the deployment device we settled with, since it provides a fairly accurate reconstruction of the environment's geometry and frees the user's hands for interaction, because the device is worn on the head (see Figure 3.4).

Figure 3.4: *Hololens' Spatial Mapping.*

Luckily, the Hololens Toolkit for Unity provides a very easy to use module for spatial mapping. The environment's geometry can directly be used by the pathfinding module to generate paths on the physical surfaces. The HoloToolkit also provides a spatial understanding module which allows to refine the spatial mesh and to query for positions on the wall, floor or table. This functionality was not yet exploited, but in future work it could provide proposals for where to spawn augmented objects (see Figure 3.5).

Figure 3.5: *HoloToolkit's Spatial Understanding Module Spawning Digital Objects On Walls.*

3.3 Pathfinding

Of course, navigation on a surface that is not only unknown before the application starts, but may also change during its execution, requires some complex pathfinding techniques.

Our character control solution (see vhapter Interaction, section Character Control) uses Unity's pathfinding script called *navMeshAgent*, which computes paths along which the agent can move. Unfortunately, this solution requires navigation meshes to be pre-baked and does not provide functionality to compute paths on dynamic navigation surfaces reconstructed at runtime.
To solve this problem, we chose to make use of A* pathfinding algorithm and bought a Unity asset called A* Pathfinding Pro that implements this algorithm. It comes with some high level scripts to enable a game object to navigate through the scene. But it does not provide an interface to Unity's *NavMeshAgent*; functions have different names, are not implemented, or handle things differently. Therefore we had to reimplement the interface ourselves: we created an *AStarSteeringController* similar to the *UnitySteeringController* and an *AStarAgent* that offers similar functionality as Unity's *NavMeshAgent* class, which is used internally by *UnitySteering-Controller* (see Figure 3.6). All functions and variables needed were implemented and others have been introduced for possible future implementation. We tweaked configuration options for best navigation results.
The *AStarAgent* asynchronously finds a path from the current position to the desired destina-

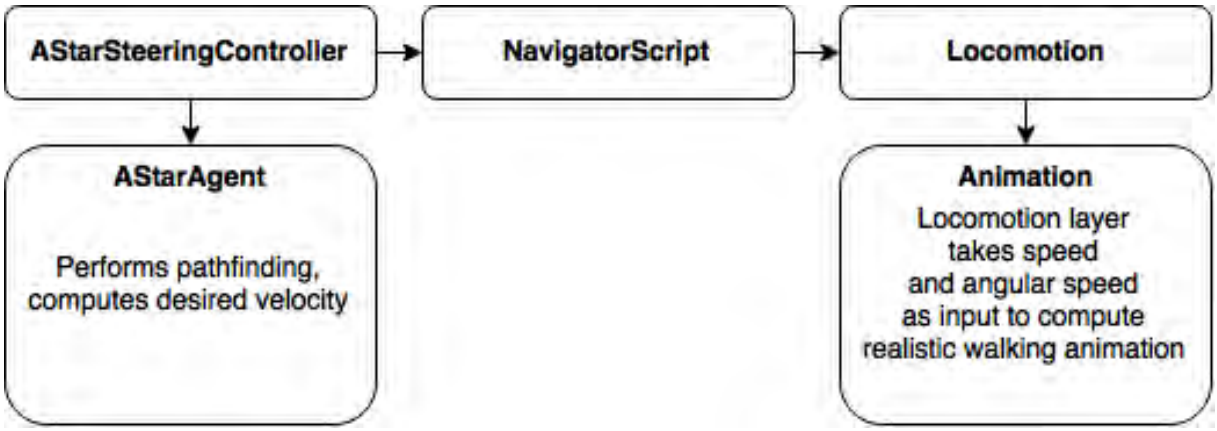


Figure 3.6: *AStarSteeringController and AStarAgent.*

tion. This path is a set of waypoints which are the endpoints of path segments. The character's steering position is computed as the point that is closest to the next waypoint to reach, is closest on the current path segment to walk on, and has a distance from the character's current position smaller or equal to the distance the character can walk in one second.

4

Environment Understanding

The second challenge is to give the character knowledge about specific interactive objects in the environment. First we need to identify those objects based on images, then we need to localize them in 3D space, and finally we need to track them over time.

4.1 Server-Client Architecture

Due to the time limitation of a master's thesis, we sometimes had to aim for solutions that were easier to implement, in order to complete the work in time. This is why we chose a client-server based architecture which enables us to delegate the task of object recognition to a platform that facilitates the implementation. Keeping the bigger picture of a final product in mind, where all of the functionality is integrated into a unified system, we designed clear interfaces between the modules in the main application. This makes it easier for the future, to port all of them on the main deployment device in order to bypass the network latency for better performance. To run image based object recognition on the Hololens, one could for example use Microsoft's Computational Network Toolkit (CNTK). But probably the device does not have the required computing power yet.

4.2 Lookup Server

At first we made our main AR application connect to the server by hardcoding the server's IP address in the client application's code. But the machines used during the development receive new IP addresses on every reboot, so this was not a viable solution. Instead, we implemented one single lookup server which needs minimal computing power and can run on a Raspberry Pi

which minimizes the effort to keep it at a fixed IP address. It is implemented as a python script that stores an IP address which can be set or queried. The server informs the lookup service about its IP address when it boots, and the client application queries the lookup service for the server's IP address when it boots (see Figure 4.1).

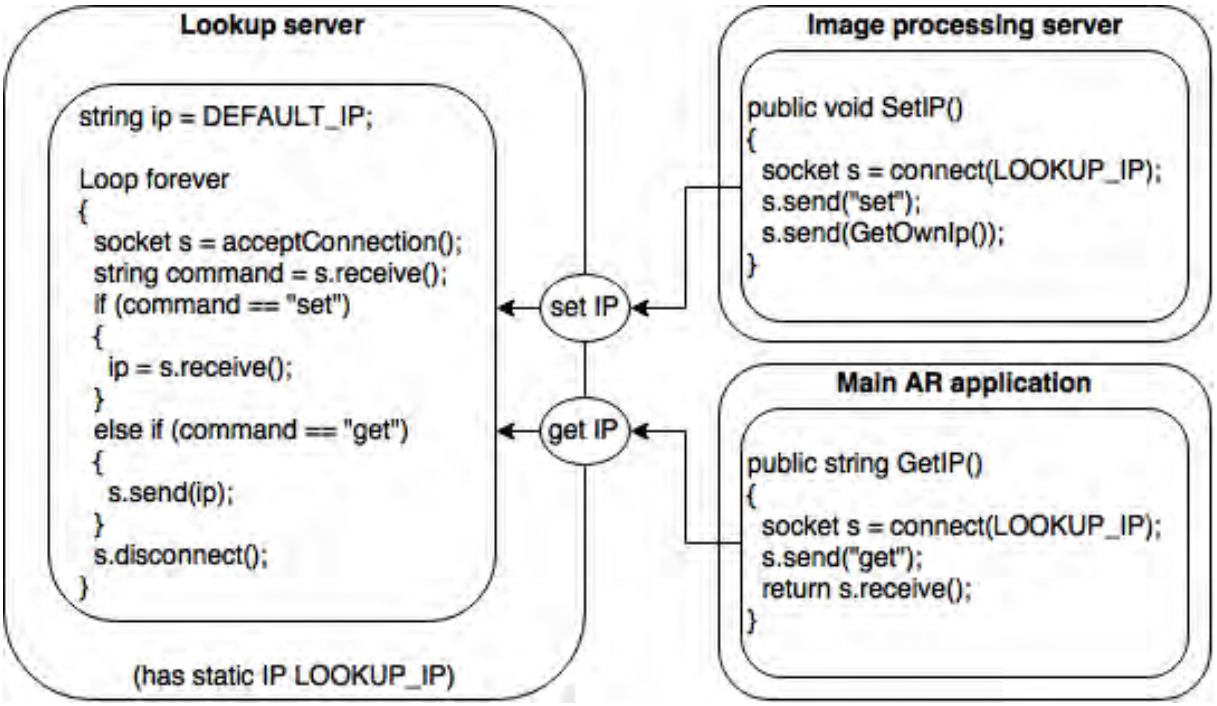


Figure 4.1: *Pseudo Code of the Lookup Server and its Interface.*

4.3 Main AR Application

4.3.1 Interfaces

The main AR application is programmed in C# and built with Unity, which makes it intuitive and easy to extend by artists and designers. Hololens provides much of its functionality through Unity Prefabs, so called Managers, which have modular interfaces consisting of a set of public functions that can be accessed by one's own game objects and scripts. For reusability purposes in any of our future projects, we chose to implement our object recognition functionality in a similar way with a Unity prefab called *ObjectRecognitionManager*.

The interface our *ObjectRecognitionManager* offers consist of three public functions. The first one returns the copy of a list of all recognized objects (read only). The second one is a registration to get a callback upon detection of a new object, and the third one is a registration to get a callback upon movement detected for a particular object. This allows other game components to query for the set of recognized objects or to register for events, which in turn inform them

when new objects get detected or if they are moved.

Since our solution for image based object recognition requires querying an external server, we divided the implementation of the *ObjectRecognitionManager* into two sub-modules: the first one performing spatial object localization and tracking and the second one doing actual image processing. The first sub-module takes a webcam image which it uses to query the second sub-module in order to acquire the information it needs about the objects that are present in the image. This allows us to implement the image processing sub-module as a man in the middle that queries an external server instead of performing the image processing itself. Such a network module needs to run asynchronously, in order to not block the Unity main thread and keep a good frame rate.

The HoloToolkit provides an asynchronous API called PhotoCapture, which allows access to images taken with the webcam. It is implemented using callbacks and provides functionality to initialize it and to take images. It returns a texture containing the webcam image along with metadata, such as the location of the camera at capture time. We chose to implement the image processing sub-module in a similar way and called it *PhotoLabeling*.

The interface it provides consists of two asynchronous public functions with callback, one for initialization and the second one for processing an image. Initialization is done by querying asynchronously the lookup server for the local IP address of the image processing server, connecting to it and calling back on the main thread. Processing an image consists of sending it to the server and receiving the responses, which are then again forwarded in a callback. This could later be replaced with a native solution for image processing, which may or may not be implemented asynchronously.

Our *ObjectRecognitionManager* internally accesses both HoloToolkit's PhotoCapture API and our own *PhotoLabeling* module to take images and process them. Given the resulting labels and two dimensional bounding boxes, it performs spatial object localization and tracks the found objects (see Figure 4.2).

4.3.2 Spatial Localization

Identifying objects in an image is one thing, but localizing them in the real world space is even more complex. The image processing server provides our main AR application with information about recognized objects and their bounding boxes within images. These two-dimensional coordinates of a recognized object in the image are then used to localize it in world space by casting a ray from the camera's position at capture time into the world direction that corresponds to the image coordinates of the recognized object.

Since the web-camera is on the user's forehead, we can not trace a ray from the rendering camera but instead we need to know the position and orientation of the webcam. Fortunately, the

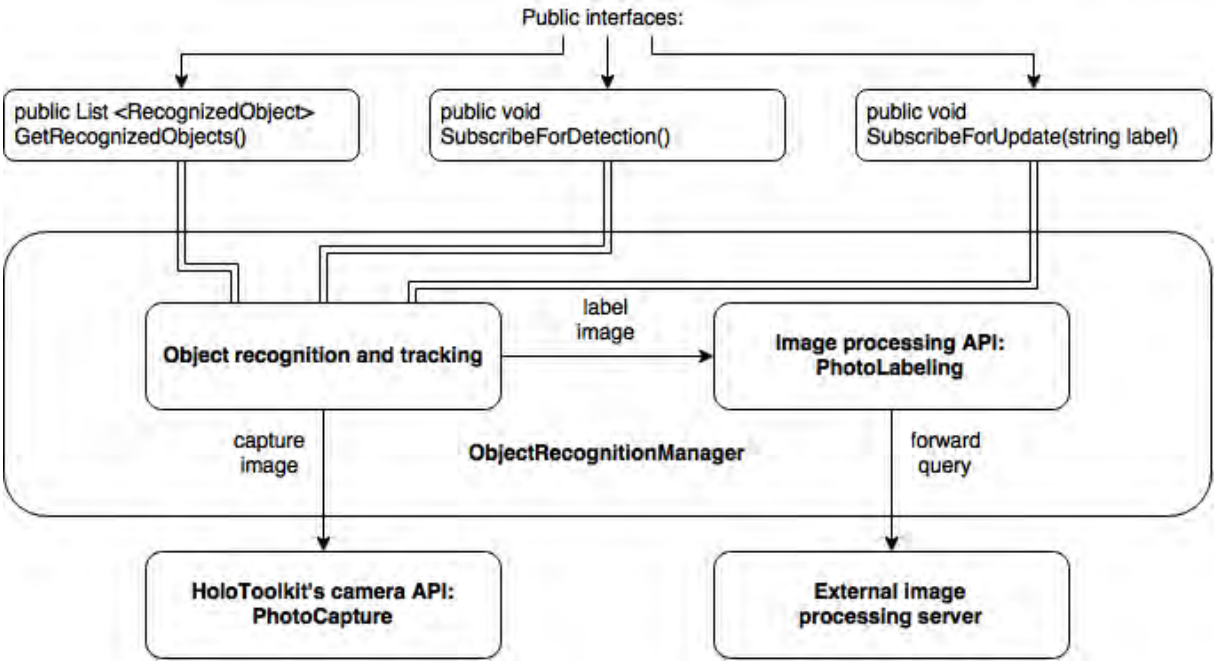


Figure 4.2: *Object Recognition Manager.*

PhotoCapture interface provided by the HoloToolkit also returns metadata like the projection matrix and world to camera matrix, along with the actual image. The world to camera matrix can be inverted to obtain a camera to world matrix.

When our main application takes an image with the webcam, it remembers the camera's world position at capture time. When it gets an answer to its query, it uses the remembered position, the viewport of the camera and the returned bounding boxes of the classified objects in the image to compute a ray that corresponds to the object's coordinates in the image. Given a 2D coordinate in the image, the projection matrix is used to compute a point in front of the camera that would be projected to this pixel coordinate in the image. This point is then placed in world space using the camera to world space matrix.

Finally, the ray used for raytracing starts at the camera position and goes through the computed point. The recognized object is then positioned at the location where the ray intersects the spatial mapping. The method we use has the advantage that it requires no preparations of the objects and is completely based on processing webcam images and the spatial mapping we are already provided with.

4.3.3 Achieving Pseudo Realtime Tracking

Our main application keeps track of the positions of previously classified objects and tries to use that information to match, move or remove them according to newer image processing results. The image processing is not truly real-time, but if its frequency is high enough, we can use it repeatedly to track objects. Tracking objects using previous knowledge and new information

presents a few challenges. For example, one needs to answer the question of what it means when an object is expected in an image but is no longer there, or when an object that was previously elsewhere is later encountered at a new location. It is not trivial in such a situation to distinguish whether or not it is the same object or another one of the same type.

Our current assumption is that there is no more than one instance of each object, so we create a new instance if the toy was not yet detected, and otherwise we just update the position. Hence every toy that has ever been detected is instantiated and its position is where it had last been recognized. This assumption is very strong and only chosen for practical reasons. A polished solution would have to find a way to reason about things in order to allow several instances of the same category.

Since we have a network latency, we could theoretically stream a window of queries to the server to maximize the frame rate of new information, by constantly sending another image when a new response arrives and keeping a constant number of requests in the network pipe. But if the window is too big, we delay responses even further by having outdated images in between, so the window size should not be so big as to cause any queuing at the server. This is basic queuing theory, but we did not have the time to analyze our system and since we plan to implement this feature locally in the future, this is not really relevant for us. We only have one pending request at a time. Nevertheless, while we wait for a response we already take a new picture used for the next request. Since both capturing an image and querying the server are asynchronous tasks with latency, this saves us some time.

4.4 Image Processing Server

The image processing server is meant as a temporary solution that facilitates implementation, because it can run on a different platform than the main AR application. It is currently programmed in python and runs on a Linux environment, but could later be replaced with a native solution to run on the same device as the rest of the system if the Hololens is capable of handling the workload.

The image processing server initializes itself by setting the IP stored in the lookup server and loading the files needed for image processing. It then runs in an infinite loop in which it accepts one connection at a time, and keeps answering queries until the connection is closed. Such a query consists of receiving an image, processing it and sending the processing results back. Notice that the interface for networking is programmed in C# on the main application, whereas the server is implemented in python, so defining the protocol is crucial. Receiving an image involves receiving its size first, which is stored in a fixed number of bytes and defines the number of bytes that follow and contain the content of the image. That image is then processed in order to detect and identify objects in the image. The results are sent back to the main application in a response which starts with the number of identified objects. Then for each object it contains the score, coordinates of the bounding box (center and dimensions), the size of the label string (in bytes) and finally the label as string (see Figure 4.3). As seen in Figure 4.4, we succeeded

in having the server process images and return the correct labels.

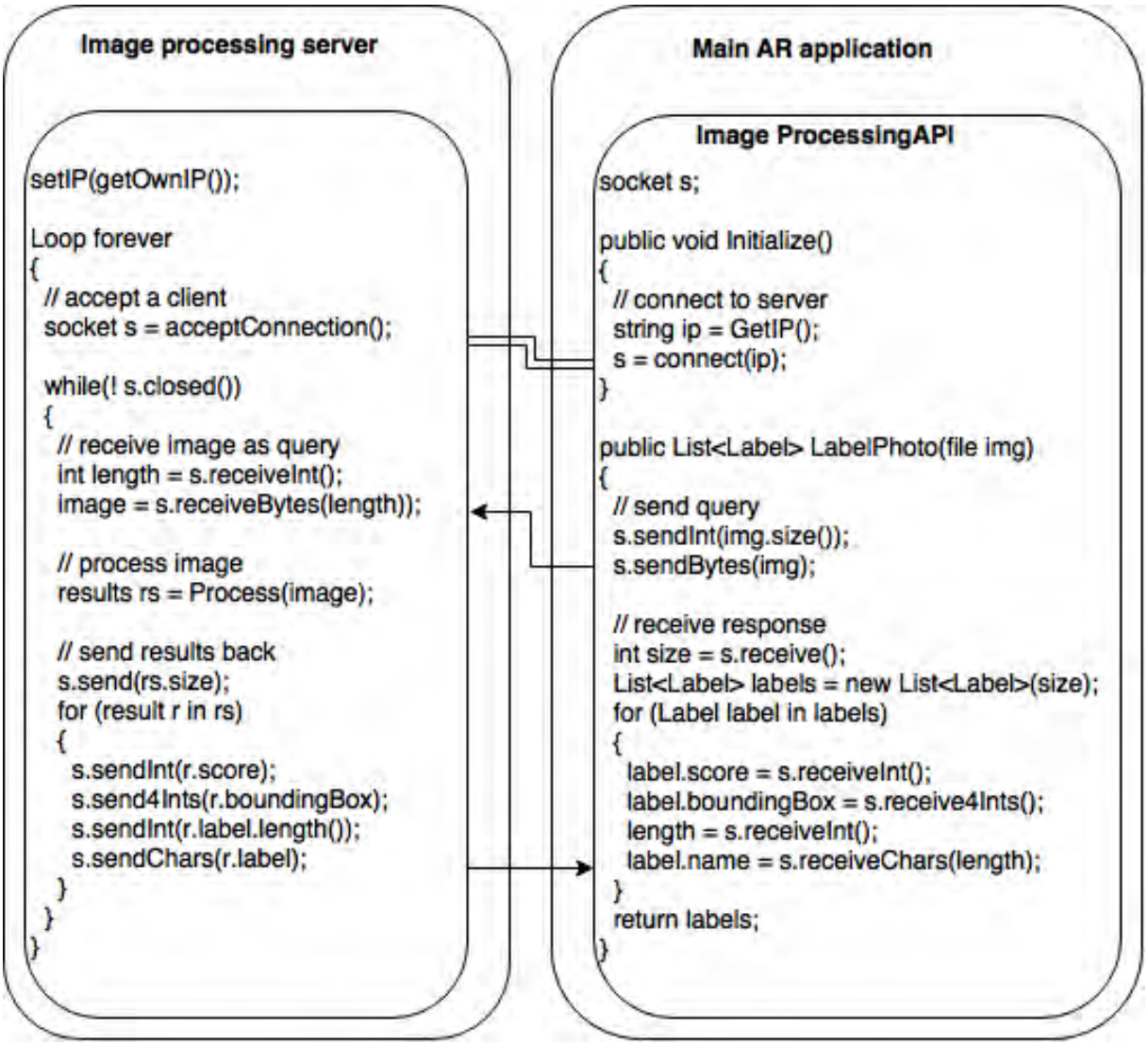
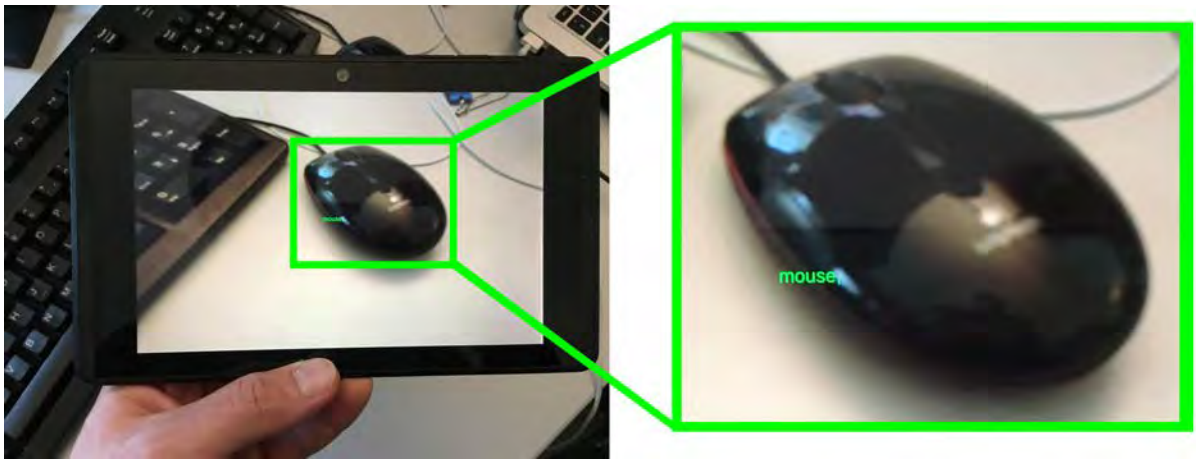


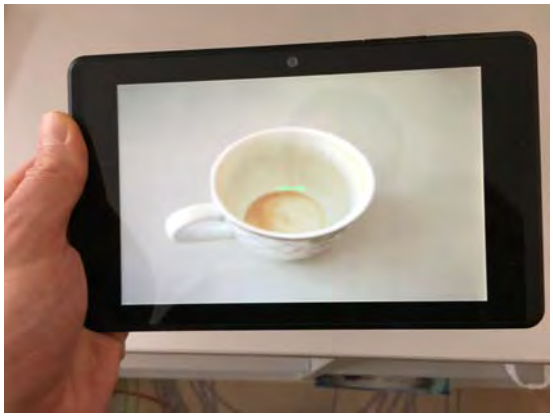
Figure 4.3: *Pseudo Code of the Image Processing Server and its Interface.*

Our first approach was to use the 3D scan of the room's geometry to take images with a virtual camera from the character's perspective. This approach worked for images in self created 3D scenes (see Figure 4.5), but it failed on 3D scans, because the reconstruction quality is not good enough for that purpose. Figure 4.6 depicts the loss of texture in 3D reconstructions using Tango. We gave up on this idea and decided to use web camera images instead.

Image based object identification is a well-known research area in Computer Vision. Algorithms evolve to get more and more powerful at solving the task, and recent progress in the area of deep learning has brought tremendous improvements. We explored two technologies for image based object identification: Tensorflow and Faster R-CNN.



(a) Tensorflow Identifying a Computer Mouse.



(b) Tensorflow Identifying a Coffe Mug.

(c) Tensorflow Identifying a Chair.

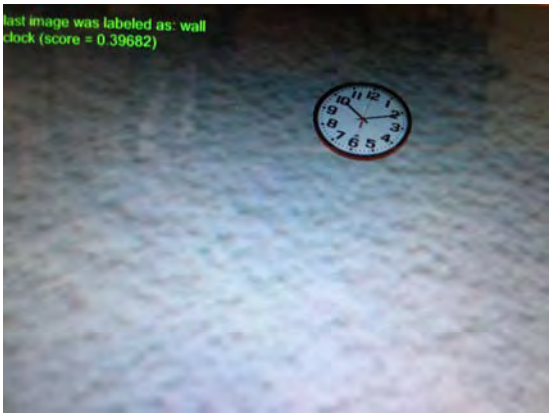
Figure 4.4: *Tensorflow Identifying Real World Objects.*

4.4.1 Tensorflow Solution

The first option we considered is the one stated in the project proposal: Tensorflow [AAB+15]. Google released its powerful Machine Learning API along with a pre-trained model called Inception v3 which classifies images across a thousand different categories. An image is given to the model and it returns a score for each of the categories it was trained on. Those scores add up to the value one, and the higher a score for a given category, the higher the probability, that the image corresponds to it. Testing this model on images found in the internet worked very well on various contents in photographs (see Figure 4.7 (a)). But it performed poorly on images of toys, because they are made out of plastic, the backgrounds are artificial, and the model is not trained on that type of images (see Figure 4.7 (b)).

Tensorflow allows you to train a model from scratch using labeled input images. But training such a model takes a lot of images and computing power. This option can be taken into consideration for best performance, but we chose a more practical solution. The cheaper alternative is to retrain an already existing model, in this case the Inception v3 model, on a distinct image classification task. This process is also called learning transfer. To finetune a model, the

(a) Tensorflow Identifying a Digital Soccer Ball.



(b) Tensorflow Identifying a Digital Clock.

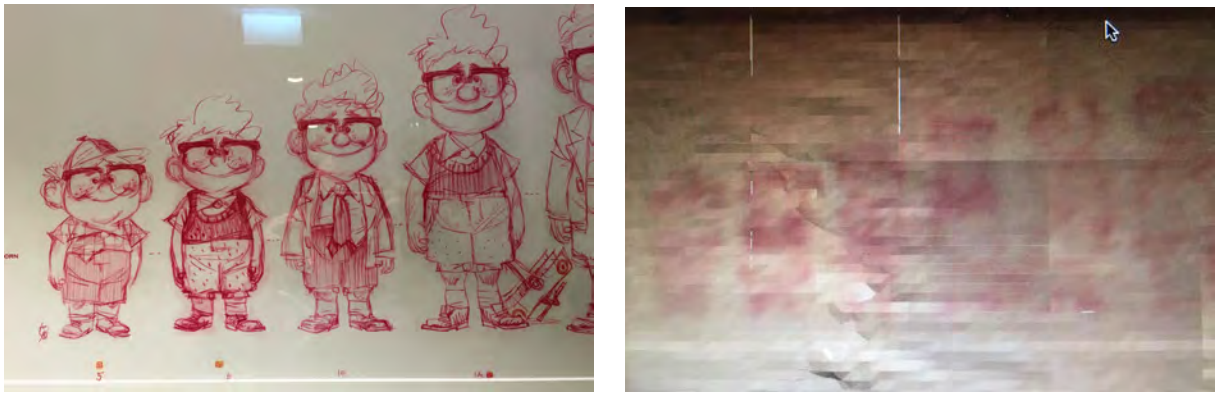
(c) Tensorflow Identifying a Digital Lion Poster.

Figure 4.5: *Tensorflow Identifying Digital Objects.*

pre-trained model is loaded, the topmost layer is removed, and a new one is added using our own image dataset for retraining. The resulting model successfully classified images into the different toy categories that we retrained it on (see Figure 4.7 (c)).

However, our application does not only encounter large scale images of single objects, but rather needs to recognize multiple objects within a single image and localize them. We use OpenCV to try to detect objects regardless of what they are. Our naive algorithm makes the strong assumption that toys are placed on uniformly colored backgrounds. The three steps performed are edge detection, edge dilation and contour finding (see Figure 4.8).

Edge detection is meant to find object edges (see Figure 4.8 (b)). This breaks down if there are edges that do not correspond to object boundaries.
Dilation is meant to merge multiple edges that belong to the same object so that we are left with one thick edge per object.
Contours is the final step in finding the bounding boxes of the individual objects. This functionality is provided by the OpenCV library. Notice that we ignore edges that touch the image



(a) Scanned Surface. (b) Scan Result.

Figure 4.6: *The Loss of Texture from a Scanned Surface (a) to the Reconstructed Mesh (Using Tango) (b).*

border bounding boxes with an area smaller than a given minimum value, since those edges would belong to objects that are not entirely in the image or are just caused by small, high contrast areas in the image.

Finally each bounding box is used to classify the corresponding sub-image using Tensorflow.

This task of finding regions in an image that correspond to distinct entities is an entire research topic in itself and other work has been done to solve it. Not only did our algorithm perform poorly, because of the many Tensorflow invocations per image, but it also made strong assumptions about the background behind objects for accurate object proposals. Very often images contain many uninteresting high contrast regions due to texture and color. The idea that edges are caused by object boundaries is not practical. This is why we decided to leave this approach behind and turned towards an alternative.

4.4.2 Faster R-CNN Solution

Caffe [JSD+14] is another well known API for deep learning and Faster R-CNN [RHGS15] implements object detection and classification using Caffe. It provides object proposals as well as labels which is ideal for our purposes. So we implemented a second version of the image processing server which uses Faster R-CNN for image based object recognition. It is implemented in a similar way as the one using Tensorflow and a lot of code is reusable since the only part of the code that needs to be replaced is the part that performs the image processing. The performance was greatly improved in comparison with the first implemented server, due to its efficient region proposals, but still has potential for even further improvements.

Similarly to Tensorflow, Faster R-CNN also allows to finetune existing models to your own dataset. Since it also generates object region proposals, training images need so-called annota-

tion files that specify all objects present in the image and their bounding boxes.

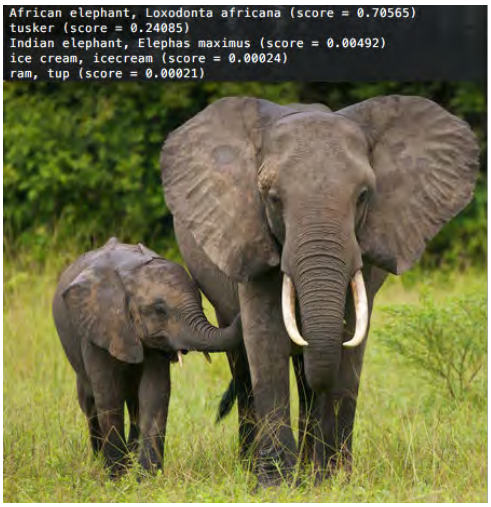
4.4.3 Gathering Training Image Datasets For Tensorflow

Training models can lead to artifacts if not done properly. The dataset should include images taken in front of different backgrounds at different locations and with different illuminations at various times of the day. Collecting datasets can be very time consuming. In order to facilitate the gathering of images, we wrote an application that runs on the same device as our end result, therefore taking images similar to the ones we are expecting. It sends the images directly to a server written in python, which stores them in a new directory for every new shooting session. This way individual photos don't have to be looked at for the sorting process, because entire directories can contain images of the same category.

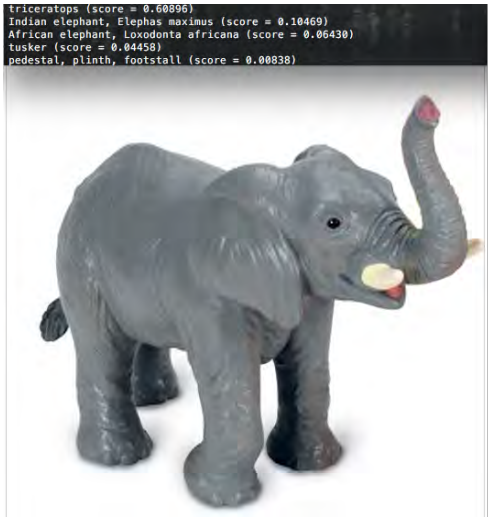
4.4.4 Generating Annotation Files to Finetune Faster R-CNN Models

To finetune an existing Faster R-CNN model for our own dataset, the correct way to do it is to take thousands of images in different conditions (background, lighting etc). Those images would need to display the particular objects to be recognized from different angles, at different scales and positions in the image. Each image would need to be annotated with an xml file that specifies the objects occurring in the image and their bounding boxes. Finally we would need to create text files containing a list of all the images and a list of all annotation files that should be used for training, validation and testing.

In order to automate this process, a common method is to create a 3D mesh of each object and to render them from different camera positions and angles and on different backgrounds. Since we did not have 3D reconstructions of the toys, we chose a hybrid solution: we took images from the toys in front of a green screen in order to segment an crop them. This is the equivalent of rendering the 3D model from different angles. We then designed an algorithm to generate training images, annotation files, and lists of files to be used for each finetuning step. To create the images we took the segmented and cropped green screen images and overlayed them onto different backgrounds with random positions and scales, which are used to generate the corresponding annotation files. This also allows for automatic generation of files containing the list of images and annotation files to be used for training, validation and testing.



(a) Tensorflow on Real Elephant (Using Inception V3).

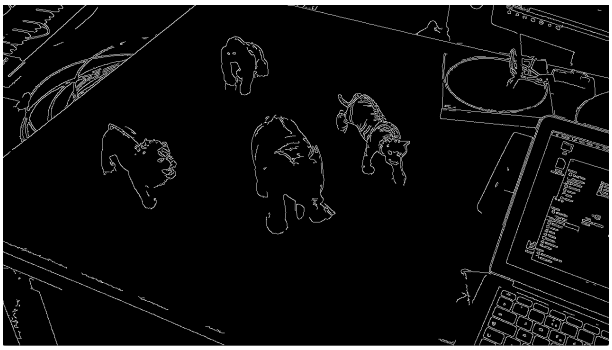


(b) Tensorflow on Toy Elephant (Using Inception V3).

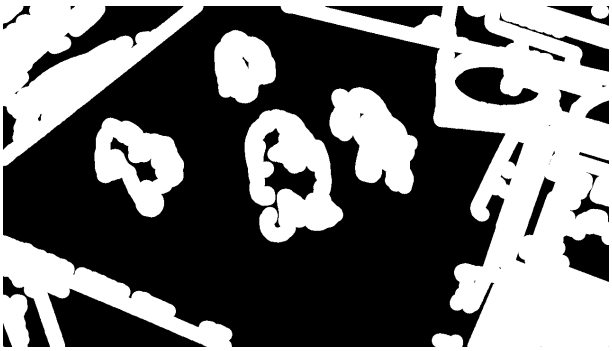
(c) Tensorflow on Toy Elephant (After Finetuning Inception V3).

Figure 4.7: *Tensorflow Performance on Real and Toy Elephant Before and After Fine Tuning the Inception V3 Model.*

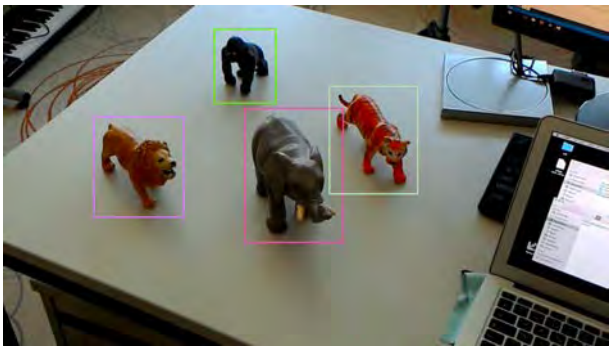
(a) Original Image.



(b) Edge Detection.



(c) Edge Dilation.



(d) Countour Finding.

Figure 4.8: *Naive Object Detection Algorithm Using OpenCV.*

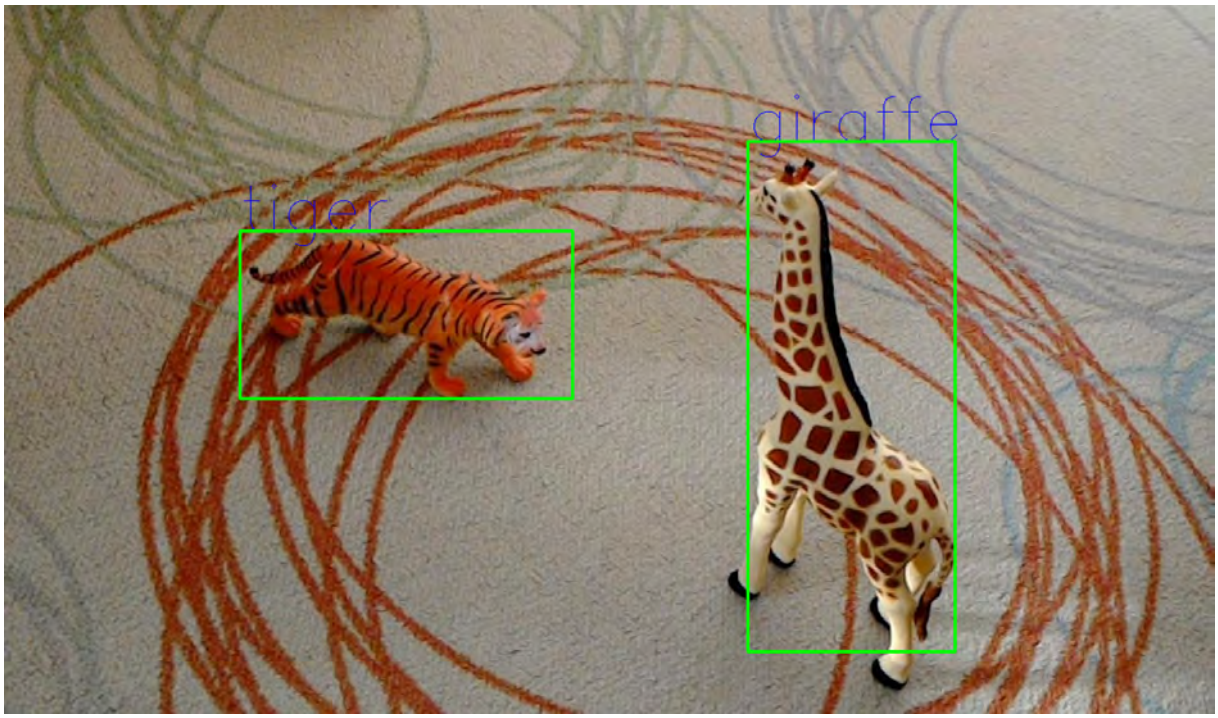


Figure 4.9: *Object Proposal and Identification Using Faster R-CNN.*

5

Interaction

Our third goal is interaction between the character, the environment and the child. For that we need to use knowledge about the environment and the toys to determine the behavior of the character in such a way that it fits well into the current context. Some of the questions we need to answer are what types of interactions we would like to achieve, how we can make the character autonomous and how the experience can be made interesting over time by allowing designers to author stories that can unfold in certain situations. We also need to implement speech conversation to make the interaction feel natural.

5.1 Forms of Interaction

There is a conflict when it comes to interaction between a digital character and a physical environment since the character cannot influence the latter directly. In this subsection we will discuss possible forms of interactions.

Passive Interaction. There are passive forms of interaction which do not require modifying the environment, like jumping around on the room's furniture or pointing at toys and saying what they are. This form of interaction demonstrates the character's awareness of the environment, meaning the physical space and recognized toys.

Reactive Interaction. The child can on the other hand actively modify the physical environment like moving a lion toy towards the character who can respond meaningfully for example by running away from it. This form of interaction requires the character to understand what the child is doing with the toys, but initiatives are taken by the child and the character is merely responding to them.

Active Interactions. If the character wants to actively change the environment he can either reach out to the child to ask for cooperation, or augment the environment with digital objects which he can modify. If they are in a close relation to physical objects, the child and the character can interact with each other in a meaningful way. Imagine a game of chess with half of the pawns being physical and the other half being digital, where both user and digital character can independently take actions which are meaningful to their counterpart on an abstract level because of the rules of chess. This concept can of course be generalized to more simple situations. The character can spawn digital objects that give purpose to physical toys with which the child can interact.

Through communication the child can instruct the character to do things, and similarly the character can also tell the child about possible actions that could be performed. For example, if there is a lion and a gorilla in the scene, the character can spawn a tree somewhere and go to the gorilla to defend it from the lion until the child puts the gorilla close to the tree, which would make the gorilla safe and free the character to do other things.

5.2 Character Control

Our application controls character animation and movement using a system called KADAPT which is an altered version of ADAPT [KFS+16a]. The character's behavior is defined by a behavior tree which is constructed using a library called Tree Sharp Plus.

5.2.1 Behavior Trees

A behavior tree formally defines a system's behavior and is executed in so called ticks, in which all the running nodes are executed. When a node is executed, it returns a *RunStatus*, which can be *RunStatus.Success* if the node's subtree is completed, *RunStatus.Failure* if something went wrong, or *RunStatus.Running* if its subtree should keep being executed in the next tick.

A behavior tree contains control nodes and leaf nodes. Control nodes can be loop nodes, sequence nodes, parallel nodes, selector nodes and more. For example, a sequence node will return *RunStatus.Running* as long as not all of its children have returned success and execute the currently running child. It will return *RunStatus.Failure* if a child returns failure and return *RunStatus.Success* once all its children have successfully completed.

Some needed control nodes are not implemented, but can be achieved by combining existing ones. For example, to achieve a parallel-first-success, one can use a parallel-first-failure, with a parent and all children being nodes that invert the returned status of their subtrees. In Figure 5.1, the combination returns success as soon as one of the children returns success, and if all children fail it returns failure as well, because none succeeded. Hence this construction is equivalent to a control node parallel-first-success.

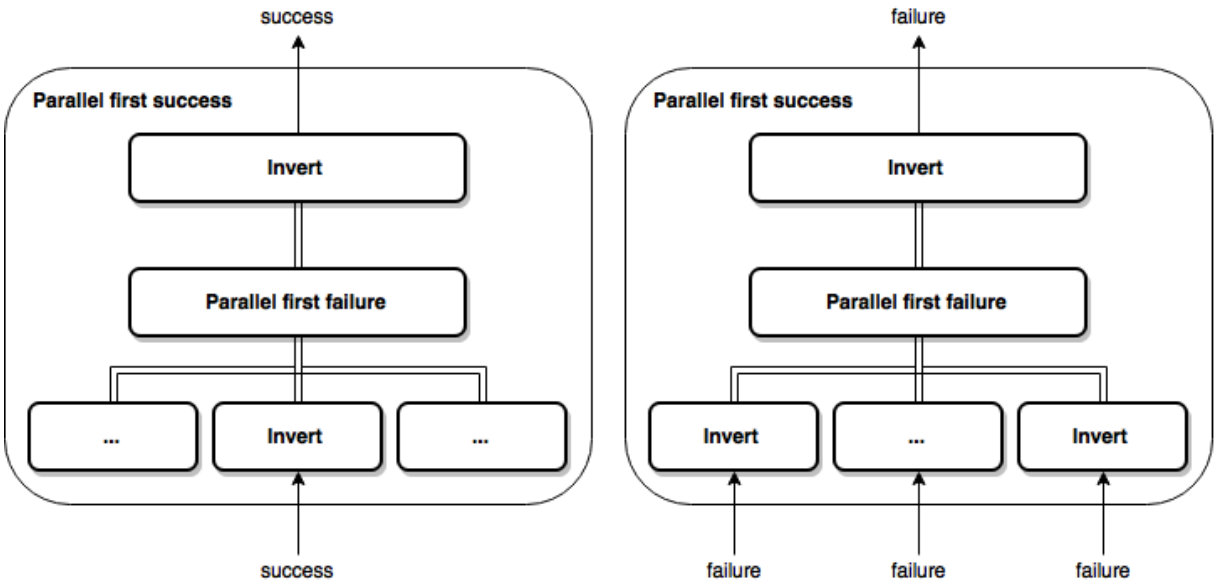


Figure 5.1: *Constructing A Parallel First Success using a Parallel First Failure and Inverts.*

Leaf nodes are either assert nodes, invoke nodes or wait nodes. Assert nodes are used to evaluate a boolean expression and return *RunStatus.Success* or *RunStatus.Failure* in order for the parenting control nodes to know which subtrees to execute in the next tick. A leaf invoke node is implemented to call a function that returns a run status, and returns that run status itself as a node. Therefore such a function typically executes some function provided by the system, and then checks for some condition determining whether or not the action is complete. If it is, it returns success, and otherwise its returns running which will cause it to be executed again in the next tick. If something goes wrong it can return failure. One could think of using the failure status to return a state in the tree, but it is strongly recommended to use a leaf assert node for such purposes, which can simply be added to the leaf invoke in a sequence. A leaf invoke node is used to carry out character actions called affordances, like setting a navigation target or setting an animation to play. Wait nodes are an easy to use interface of a leaf invoke node to wait for some time.

5.2.2 Altered IBT Design

One can easily design a behavior tree which executes a linear and unmodifiable story. But it is a challenge to incorporate user interaction as part of the story and enable the user to impact the story execution. IBT [KFZ+15] presents a behavior tree design that enables separation of the story arc execution and the monitoring of the system's state and the user's interaction. We were inspired by that design and implemented an IBT-like behavior tree containing two subtrees: one for story arc execution and one for state monitoring.

The first subtree contains all the possible story arcs and is organized in such a way that the selected story is executed until completion or stopped if a new story arc was chosen. The

second subtree is monitoring the system's state and the user's interaction and sets the chosen story arc to be executed. For example, if the user asks the character a question, the character should first respond to it before doing anything else (see Figure 5.2).

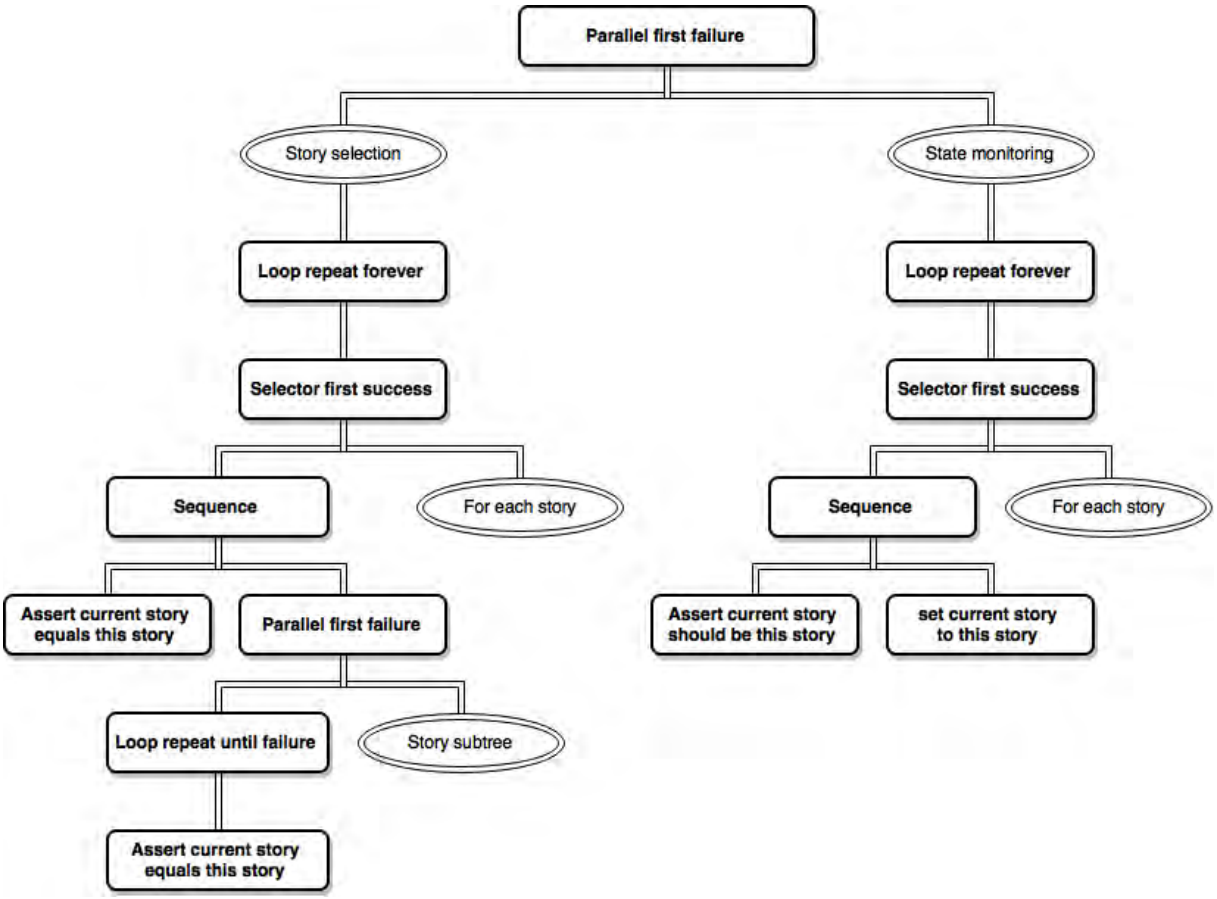


Figure 5.2: *An IBT-like Behavior Tree.*

In the IBT design, a global variable contains the value of the story arc that should be executed. This variable is of type *enum* and hence has a static value, meaning that each value it can contain corresponds to exactly one story arc subtree, and provides no other information about a particular story instance, such as for example the associated participants. The state monitoring subtree evaluates the states of smart characters and smart objects in order to set the current story through this variable.

In our design, we decided to create and store an instance of a class called *Story* which stores a story instantiation with parameters. For example, the story can contain a string with a text to be spoken, or the position to which the character should go to. The character has a queue of stories which it executes. When a story completes, it fetches the next story from the queue. This allows for two features that we discuss in the next two sections: interrupting and resuming stories, as well as choosing event participants at runtime.

Story Interruption and Resumption. We use our story class to store achieved checkpoints in

the story by incrementing a counter. If a story gets interrupted, it is put in the front of the queue and the interrupting story is executed instead. Using the checkpoints that are stored in a story, the second time a story gets executed, the already achieved checkpoints can be skipped and an optional recapitulation substory can be executed instead, for example to put the user back into context.

We demonstrate this functionality in a story where the character goes to a few animals and introduces them (one substory for each animal and a checkpoint at the end of each of those substories). If the user interrupts the story by placing the lion next to the character, this story is interrupted by a new story in which the character runs away from the lion. The interrupted story in which animals were shown is suspended. Once the lion is fare enough away, the character will resume introducing animals. But instead of starting all over again, the story execution will skip the ones already shown, and enumerate them instead to save some time, yet still put the user back into context (that is the optional recapitulation substory).

In order to make story authors use this feature, we implement a function called *CheckPointStory* which takes as arguments the checkpoint index, a substory, and an optional recapitulation substory, which will be executed in case the substory was already completed previously (see Figure 5.3).

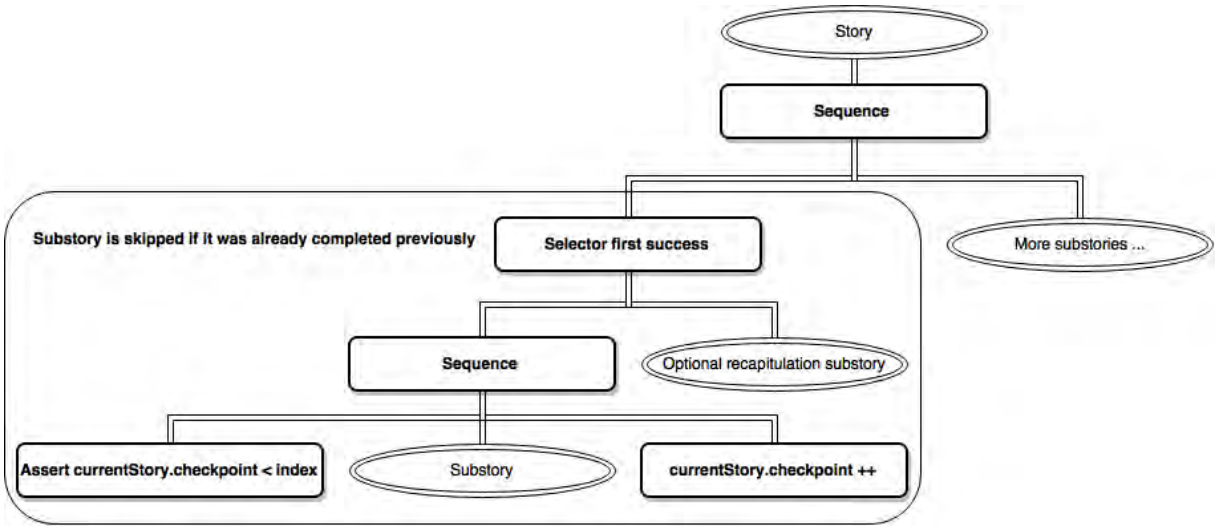


Figure 5.3: *Story Organized in Substories.*

Dynamically Chosen Event Participants. As mentioned in the related work about Emergent Narrative [Ayl99], there is a conflict between the designer who authors stories and has a well defined imagination of the situation in which it should be played out, and the user who has the freedom to look wherever he or she wants and can disrupt the story or lose important parts of it from its visual field.

As also mentioned in the related work chapter, current systems like CANVAS [KFS+16b] exist for authoring stories and they create compelling visual executions. In a normal scenario, when

authoring such a story, the designer can choose participants, events and locations a priori, but in our case the environment and the toys are only determined at runtime. Hence, one of the main challenges is to use those existing tools, but loosening some constraints in order to create stories in a more general context, keeping some freedom for the involved participants and the environment.

We do not provide a framework to author such stories, because it is up to the existing systems to incorporate this type of functionality. Nevertheless, we explore the possibilities of dynamically chosen participants, in order to understand how such a system ought to be designed. For our purposes it would be important to enable story authors to define sets of participants for which a story could be played out.

The first approach for choosing participants for a story arc, was to define them before we start an event. In this case we determine if enough and the right types of toys are available and set them as the participants. Otherwise we run an alternative event in which, for example, the character says that he cannot execute the event because too many toys are missing.

The second approach was to implement the choosing as an affordance within the behavior tree of the event, as well as the alternative action in case toys were absent. The advantage here is that if the user asks the character to show him three toys, the character can start the story arc even if only two of the toys are present. While the first events execute, the user can add a third toy to the scene and the story will unfold while it's being executed, by choosing the third participant only after the first two were involved. This applies generally to more complex story arcs where we may want the character to make decisions about participants on the fly while executing the events.

Implementing such decision making as an affordance presents some architectural problems. Since affordances are functions without arbitrary return functions and can hence not pass arguments between tree nodes, the decision made about the participants needs to be stored somewhere by the affordance that chooses it and read from there by the following affordance using this participant. This introduces lots of challenges when considering the possibility of leaf nodes that are executed in parallel in a complex behavior tree and possibly concurrently write to the same storage.

At tree construction time we need to define exactly which part of memory each node of the tree should write to and read from, in order to avoid any concurrency. Using the story class mentioned previously, the tree author can pass different indices to each parallel subtree for the intended participant to be written to and read from (see Figure 5.4).

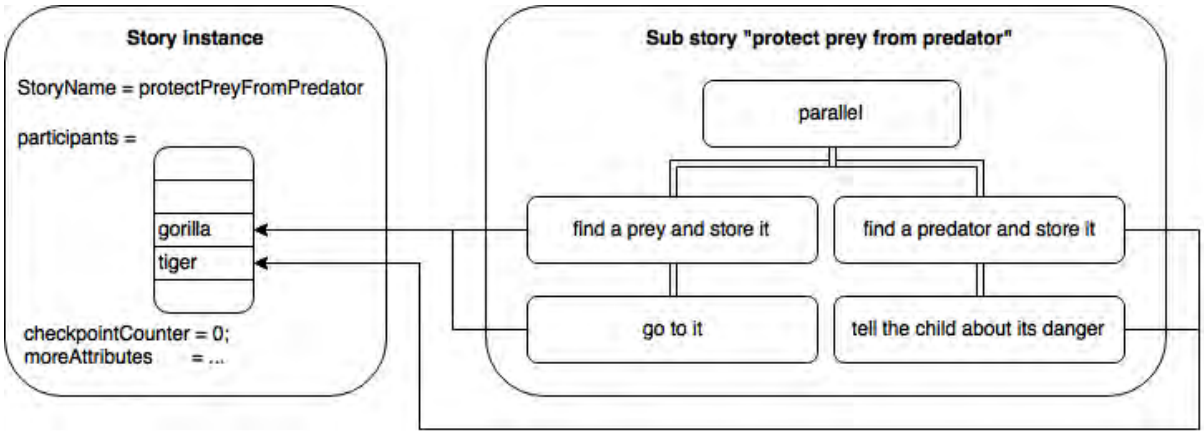


Figure 5.4: *Choosing Participants at Runtime, Storing them and Parallelism.*

5.3 Implementing New Affordances

Because of the architecture of such a behavior tree, KADAPT has three scripts that control character actions: *BehaviorMecanim*, *CharacterMecanim* and *BodyMecanim*. *BehaviorMecanim* contains the leaf invoke nodes that can be used in behavior trees. *CharacterMecanim* implements the functions executed in those leaf nodes and the condition checking mechanism returning a *RunStatus* for the node to know whether or not the action failed, is still running or has completed. *BodyMecanim* implements the functions interfacing any functionality provided by the rest of the system, such as playing an animation or setting a navigation target (see Figure 5.5).

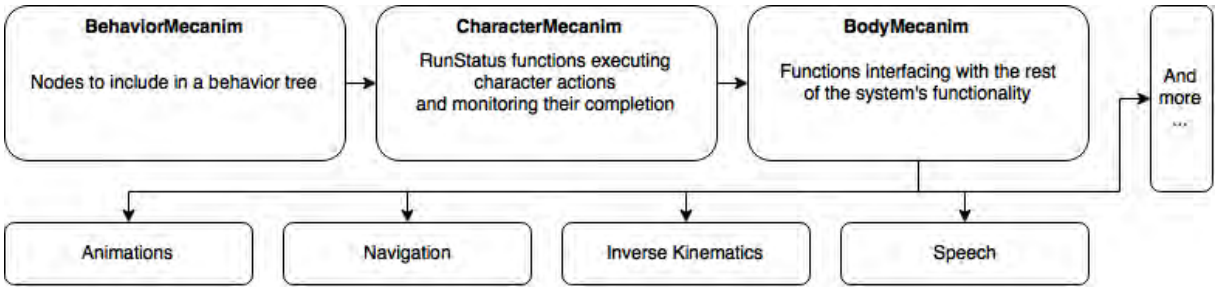


Figure 5.5: *Behavior, Character and Body Mecanims.*

5.3.1 Navigation

Since Unity's solution for pathfinding cannot be used on dynamic environment maps, we need to replace the navigation functionality of KADAPT. As mentioned in the chapter about environment awareness, in the section about pathfinding, we modified the existing *UnitySteeringController* into an *AStarSteeringController* and implemented a class *AStarAgent* which provides an interface similar to Unity's *NavMeshAgent*. Mapping the interfaces was the main challenge to

achieve this. But after that, coupling KADAPT to A* is a matter of changing the used steering controller in *BodyMecanim* to the *AStarSteeringController* instead of *UnitySteeringController*.

5.3.2 Speech Synthesis

We want to enable the character to talk. The HoloToolkit for Unity comes with a utility script called *TextToSpeechManager*, which can be added to any game object and associated to an audio source. It provides a public function for speaking text, which will cause an asynchronous synthesis of the speech and upon completion will automatically call back on the main thread to make the audio source play it. Since our character controller needs to play a talking animation on the character speech, the first naive attempt was to just play the animation whenever the audio source is playing. But for a nicer design the talking animation should be started and ended within the behavior tree, and the behavior tree should be able to have the character waiting for speech to complete before proceeding. Since the synthesis of the speech takes some time to complete, querying the state of the audio source doesn't work. We therefore added three functions *IsSynthesizing*, *IsSpeaking* and *Stop*, to query if speech is currently being synthesized or if speech is being played or to interrupt it. In order to control speech from the behavior tree, we integrated speech functionality in a similar way as the animation and navigation. Keeping the same architecture, we added a behavior node for speech, which initiates speech and then waits for the speech to complete. This Node can be included in a helper subtree that first starts a talking animation, then plays the speech audio and finally stops the talking animation again. The functions *SpeechStart*, *SpeechRunning* and *SpeechStop* are added in *BodyMecanim* and call functions in the character's attached *TextToSpeechManager* game component.

5.4 Speech Recognition

One natural form of interaction between humans is speech. Therefore we want to enable the character to understand what the user might tell him.

Our initial approach and the one mentioned in the project proposal was to use Google's speech-to-text solution.
Previously one could simply query the Google Translate web site, but they recently put in a redirect to a CAPTCHA in order to prevent requests by 'robots'. However, there is a Unity plugin for Android which accesses a native text to speech API. This API takes recorded user speech as input and returns a string with the recognized text.

In an early stage of our project we have also tested alternatives, such as the ones available on IBM's Watson Developer Cloud. It provides similar functionality for speech recognition, taking audio as input and returning a string with the recognized text. IBM Watson Developer Cloud also provides many other interesting functionalities, such as text to speech, language analysis or visual object recognition. Such a solution would however imply that our system needs to comply with a server-client architecture instead of having an integrated solution. Moreover,

this solution is not free of charge and since speech conversation was not required that early in our project, we postponed the decision.

Having delayed speech implementation until the end, once we deployed our system on the Hololens, the provided Speech API on that platform made it really easy to incorporate simple forms of communication into our application. The HoloToolkit provides an API to define a set of valid sentences using an XML-like SRGS grammar file, which allows for a very modular declaration of possible speech inputs. Our system is able to recognize simple phrases with nouns and verbs, if they are defined in this file. We are hoping for future progress in Natural Language Understanding to expand this module and make conversation more independent from predefined scenarios.

Our aim was to provide a practical interface between a possible future conversation module and the rest of the system in such a way that the character can execute actions as a result of conversations, if any can be matched to the current situation. Having the character wait for user speech could easily be implemented as an affordance which would start by loading the appropriate grammar file for valid speech input. From there a communication module could take over to converse with the user until the discussion leads to a meaningful action.

As mentioned in the related work about Kairi, a way to associate speech input to possible actions in the scene is to map nouns and verbs to game objects that are present in the scene and to action that the character can perform. Again, we do not attempt to solve the task of Natural Language Processing, but we are glad to use any framework that provides us easy to use tools in this direction.

5.5 Building Behavior Trees Using Speech

We explored building behavior trees using speech. This could enable designers to author complex stories using speech. For example, one could say: “if there is an elephant and there is a lion, then go to the elephant and then point at the lion”. Trying to implement this, we defined the following recursive syntax to represent trees, using operations, arguments and parenthesis: op(arg1)(arg2)(arg3). The operation can be *or*, *and*, or *ifThenElse*, in which case the arguments are other operations. The operation can also be *assert*, *do* or empty in which case the arguments can be a condition and a target (like: exists the lion) or an action and a target (like: go to the lion). The application parses the received string in order to create a Tree Sharp Plus behavior tree that represents the speech instruction and then executes it. Parsing is done by analysing the outer operation and finding beginnings and ends of the arguments by counting all the opening and closing parenthesis when scanning the string. The operation gets translated into a combination of behavior tree nodes, and the arguments are parsed themselves recursively to be added as children. This was tested and worked on hardcoded strings such as ifThenElse(and(assert(exists)(the elephant))(assert((exists)(the lion))))(and(do(go to)(the elephant))(do(point at)(the lion)))(). But we ran into problems when trying to have the SRGS grammar file construct such a tree when recognizing speech. In fact left-recursions are not al-

lowed, which means that instead of saying " go to the elephant and then point at the lion", the user would have to say "do two things: go to the elephant and then point at the lion". This makes it very teedious to author complex trees, since the speech gets very unnatural and too difficult.

5.6 Artificial Intelligence

Since our aim is to implement a playmate, we need our character to have some form of artificial intelligence and act autonomously. Actions should reflect the system's knowledge about the environment and understanding of the recognized toys. This part of the project allows for many future improvements, since the knowledge of the character really depends on the types of toys that can be encountered. In our system, the set of toys and actions are finite, yet expandable. Authors can add actions and toy names to the SRGS files, as well as animations in the animation controller. The intelligence our character has in this project is hard coded, but future work could use the same deep learning techniques that we use for object recognition in order to enable the character to learn how to respond to objects.

6

Results, Limitations, and Future Work

6.1 Results

In our final application, the following interactions are possible. The user can gaze at a position and tell the character to go where he or she is gazing at. The user can tell the character to point at animals, or to go to them. The user can also ask the character to tell jokes or to dance. The user can ask the character to introduce some of of the animals, in which case the character choses the animals at runtime, points at them, names them and goes to them to greet them. This story is split up into checkpoint events (one for each introduced animal) which can be interrupted at any time and resumed later on. For example, the user can scare the character away by moving the lion toward him. If this interrupts the previous story, the character will, once at a safe distance from the lion, remind the user about all the already introduces animals, instead of going to them all over again, and then resume the story.

6.2 Limitations

The localization of recognized objects in images is being performed with a raytrace on the spatial mapping. This presents some limitations, since the object's center can not be estimated, but rather just the surface on which the ray hit it. Furthermore, if the geometry of an object was not yet updated in the spatial mapping, this ray cast will not find a hit point at the correct position, but rather one behind the object, if any. It takes some time for the system to update the environment's reconstruction but we count on future improvements to the reconstruction updating latency which would solve this problem.

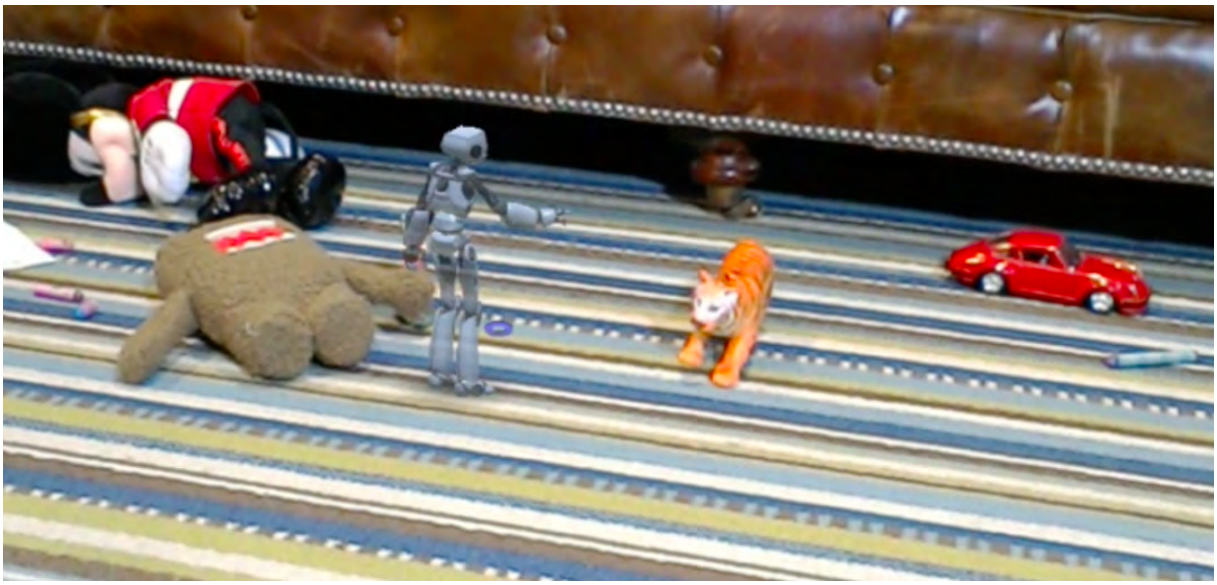


Figure 6.1: *Screenshot of the Final Application.*

Our objects detection is not entirely real time. Any future improvements in the runtime of object recognition will contribute to the performance of our system (see Table 6.1).

Part of the System	*Time*
Image capture	x ms
Network latency	y ms
Image labeling	z ms
Total processing time per image	t ms

Table 6.1: *Time Spent in Parts of the System for Object Recognition.*

6.3 Future Work

One important architecture change to consider is implementing a native solution for object recognition. This would allow for an offline application. Microsoft provides a framework called Computational Network ToolKit which could certainly be used for the same purpose. We do not know if the device could handle the workload, but it is worth a try.

In our project the character has an a priori knowledge about existing objects. To make the system more powerful, we could implement dynamic learning where the user can teach the digital character, telling him what the objects are or correcting him when he makes mistakes, in such a way that the digital character can learn new objects and improve the accuracy of recognition over time. Not only could the character learn to recognize objects, but he could also learn how to respond to it.

For now, the system recognizes speech if the user employs very specific sentences. We then perform a similar mapping as done in the system Kairai [GCM97], but in the future we could implement better conversations, for example using SyntaxNet [Syn].

The HoloToolkit for Unity provides a sharing functionality so that two users can see the same augmented reality. This functionality could also be incorporated into our application to allow interaction between several users with the system. Also, we did not yet explore the possibilities of having more than one digital character, which could make the entire experience more complex and appealing.

7

Conclusion

Existing AR applications use pattern recognition to augment flat surfaces on real world objects. Although the resulting user interactions can be very compelling, they are restricted to very specific two dimensional images. Systems like Vuforia [Mor15] allow using 3D objects, but require a careful 3D scan and the use of that very specific object, which it uses to track the camera. We draw inspiration from technologies used in robotic systems and Computer Vision, which not only allow recognition of very specific printed out patterns or objects for camera tracking, but rather recognition of any object of a given category. We present the concept of using object detection, identification and localization in AR applications in order to give a character knowledge about the environment and have him interact with it and the user accordingly.

To the best of our knowledge this thesis presents the first use of real-world three dimensional object recognition and localization in an AR application. Our contributions let us combine 3D reconstruction, object recognition, and state-of-the-art storytelling to obtain a digital character that is aware of the environment, understands it, and interacts with it and the user.

Even though our implementation, as well as the used technologies, could improve in robustness, this concept is very promising and offers many applications for future work, especially at Disney. Imagine an AR system that detects when the user is rubbing a lamp, spawns a genie who is able to augment reality with magic interactions, who can fly through the living room on a magic carpet, and display Jafar behind prison bars through any of the posters hanging on the wall. The possibilities are countless and we are planning to continue this work.

Bibliography

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2015.

[Ayl99] Ruth Aylett. Narrative in virtual environments - towards emergent narrative. In *Narrative Intelligence*, pages 83–86, 1999.

[GCM97] Stephen Grand, Dave Cliff, and Anil Malhotra. Creatures: Artificial life autonomous software agents for home entertainment. In *Proceedings of the First International Conference on Autonomous Agents*, AGENTS '97, pages 22–29, New York, NY, USA, 1997. ACM.

[Hol] Hololens.

[JSD+14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pages 675–678, New York, NY, USA, 2014. ACM.

[KFS+16a] Mubbasir Kapadia, Seth Frey, Alexander Shoulson, Robert W. Sumner, and Markus Gross. Canvas: Computer-assisted narrative animation synthesis. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Anima-*

tion, SCA '16, pages 199–209, Aire-la-Ville, Switzerland, Switzerland, 2016. Eurographics Association.

[KFS+16b] Mubbasir Kapadia, Seth Frey, Alexander Shoulson, Robert W. Sumner, and Markus Gross. CANVAS: Computer-Assisted Narrative Animation Synthesis. In Ladislav Kavan and Chris Wojtan, editors, *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*. The Eurographics Association, 2016.

[KFZ+15] Mubbasir Kapadia, Jessica Falk, Fabio Zünd, Marcel Marti, Robert W. Sumner, and Markus Gross. Computer-assisted authoring of interactive narratives. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games*, i3D '15, pages 85–92, New York, NY, USA, 2015. ACM.

[KXN+16] Mubbasir Kapadia, Xu Xianghao, Maurizio Nitti, Marcelo Kallmann, Stelian Coros, Robert W. Sumner, and Markus Gross. Precision: Precomputing environment semantics for contact-rich character animation. In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '16, pages 29–37, New York, NY, USA, 2016. ACM.

[LHFS12] Jürgen Leitner, Simon Harding, Mikhail Frank, and Jürgen Schmidhuber. Learning spatial object localization from vision on a humanoid robot. *International Journal of Advanced Robotic Systems*, 9, 2012.

[Mor15] C.R. Morales. *Developing Augmented Reality applications with Unity 3D and Vuforia:*. eAcademicBooks LLC, 2015.

[RAN11] Otmar Hilliges David Molyneaux David Kim Andrew J. Davison Pushmeet Kohli Jamie Shotton Steve Hodges Andrew Fitzgibbon Richard A. Newcombe, Shahram Izadi. Kinectfusion: Real-time dense surface mapping and tracking. In *IEEE ISMAR*. IEEE, October 2011.

[RHGS15] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[Syn] Syntaxnet.

[Tan] Tango.

[TTS04] Hozumi Tanaka, Takenobu Tokunaga, and Yusuke Shinyama. *Animated Agents Capable of Understanding Natural Language and Performing Actions*, pages 429–443. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[ZAD+16] N. Zioulis, D. Alexiadis, A. Doumanoglou, G. Louizis, K. Apostolakis, D. Zarpalas, and P. Daras. 3d tele-immersion platform for interactive immersive experiences between remote users. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 365–369, Sept 2016.

[ZRM+15] Fabio Zünd, Mattia Ryffel, Stéphane Magnenat, Alessia Marra, Maurizio Nitti, Mubbasir Kapadia, Gioacchino Noris, Kenny Mitchell, Markus Gross, and Robert W. Sumner. Augmented creativity: Bridging the real and virtual worlds to enhance creative play. In *SIGGRAPH Asia 2015 Mobile Graphics and Interac-*

tive Applications, SA '15, pages 21:1–21:7, New York, NY, USA, 2015. ACM.