Interacting with Intelligent Characters in AR

Gokcen Cimen[*] ETH Zurich
Ye Yuan Carnegie Mellon University
Robert W. Sumner Disney Research
Stelian Coros ETH Zurich
Martin Guay Disney Research

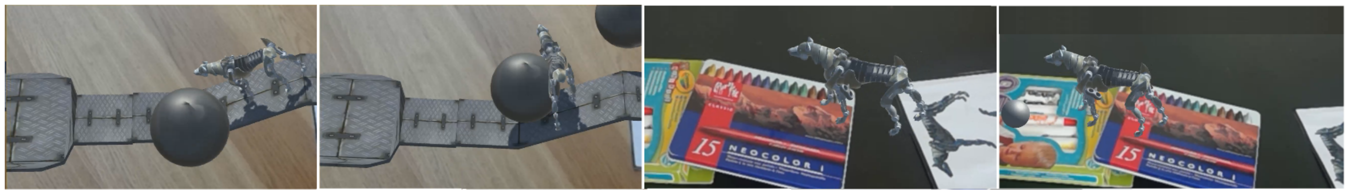


Figure 1: *Our intelligent virtual characters can navigate real world environments (right) and react to objects that collide with them (left).*

Abstract

In this paper, we explore interacting with virtual characters in AR along real-world environments. Our vision is that virtual characters will be able to understand the real-world environment and interact in an intelligent and realistic manner with it. For example, a character can walk around un-even stairs and slopes, or be pushed away by collisions with real-world objects like a ball. We describe how to automatically animate a new character, and imbue it's motion with adaption to environments and reactions to perturbations from the real world.

Keywords: Character animation, locomotion tracking, procedural character animation.

Concepts: •Computing methodologies → Animation; *Graphics systems and interfaces;*

1 Introduction

Augmenting our environment with intelligent virtual characters that can walk around and interact with our environment is an exciting and promising vision. However, achieving this idea represents several technical challenges. It remains a challenge to model the motion of a character, have it understand its environment and navigate the world in a natural way.

In this work, we take a first step in the direction of making a character intelligent, and able to interact in AR. We separate the problem into three main components. First is the modeling of the character's motion and its ability to move around. Given a character's skeleton, how should the joints move in order to go from point A to point B, including on un-even terrain. We describe a parametric model of quadruped locomotion, which we use to fill a blendtree that outputs motions conforming to control directions. The second problem is how to adapt the characters motion to un-even terrains, as well as collisions with objects (such as a ball). We full-fill this by layering on top of the blendtree, an inverse kinematics solver for terrain adaptation, and a physically simulated ragdoll for character-object collisions. The last problem is the character's ability to understand the environment and naviguate it. Online consumer-level scanning of 3D worlds remains inaccurate, and we describe our solution which cleverly combines pre-defined objects with off-the-shelf scanning solutions to provide high-resolution 3D reconstructions of the environment.

Given our intelligent character that can understand the real world and move around, we describe the types of AR interactions we support with real world objects, as well as the experiments we conducted using these interactions.

[*]goekcen.cimen@inf.ethz.ch

2 Related Work

AR: An early example of AR technology is the MagicBook [Billinghurst et al. 2001]. Large markers are integrated into a book's pages, which enable viewing virtual content through VR glasses (and later through mixed reality [Grasset et al. 2008]), based on which page of the book is open. They add various visual and auditory effects to an existing illustrated book to enhance the reader's immersion. The Haunted Book [Scherrer et al. 2008] is a prime example of well-integrated AR content. The camera is mounted on a lamp on the table and the augmented book is viewed through a computer screen. Their focus lies on interaction between the virtual content and the physical book.

Seeing a virtual character walking on different terrains is ordinary in a game environment, but to see it walking on your desk and interacting with the physical objects on the table is not as common. A few games coming with the Hololens ([Hololens 2017]), have a character navigating the real world, using the device's built 3D scanning technology. However, the accuracy of the interaction is quite low as the scans are of low resolution. From the idea of mixing physical objects with virtual 2D character, Kim et al. developed an interactive interface where a virtual character jumps over the physical blocks which the user can change their position in real-time [Kim et al. 2014]. It doesn't require any special equipment, such as wearable glasses, but a depth sensor and a projector.

Character Motion: The current practice for real-time interactive characters is to manually craft motion clips by key-framing in software such as Maya. The motions are then fed into a blendtree that blends the motion [Kovar et al. 2002; Arikan et al. 2003]. The controllers are then given a layer of inverse kinematics to adapt to different terrains, and use short-lived ragdoll dynamics for the character to react to perturbations and body collisions. We follow a similar path, which we describe in detail, but are different in one regard: when a new character comes in, the animator has to craft new motion clips which is a time consuming process that hinders the ability to scale the applications to many different characters. Hence in our paper we describe an automatic way of synthesizing the motion clips, similar to [Kokkevis et al. 1995], [Torkos and van de Panne 1998], [Megaro et al. 2015], [Bhatti et al. 2015].

3 Character Motion

Our animation model provides a virtual quadruped character the ability to navigate real world environments, and react to objects in it in real-time—given only the character's skeleton as input. The motion model is composed of motion clips (walk straight, left, right, backward, etc) that are blended in real-time. Then an inverse kinematics and short-lived ragdoll retargeting method are layered on top to adapt the motion to terrains and perturbations. We start by describing how we generate motion clips from a skeleton.

3.1 Parametric Locomotion Model

We use mechanical simulation, together with characterizations of quadruped motion, to generate locomotion for characters of different shapes and sizes. Internally, our parameterized motion generation system is based on constrained multi-objective optimization. The parameters are what we call the motion plan: a gait pattern (which foot falls at which time), foot height, center-of-mass velocity and rotational velocity. We optimize to match these values together with various regularizers ensuring smooth transitions between clips. Some constraints are implicit, or by construction. To support a wide range of characters, we constrain the skeleton to a known simplified template (see Fig. 2) that has only hinge joints and pre-defined masses. The final stage consists in upscaling the motion from the simplified template to the higher-resolution template (see Figure 2)

3.1.1 Parameterization

We use a parametric model of quadruped that are composed of articulated chain like structures, in particular, of serially connected and actuated links. The design parameters $\mathbf{s}$ is used to specify the quadruped morphology, which is given by

$$\mathbf{s} = (l_1, \ldots, l_g, \mathbf{a}_1, \ldots, \mathbf{a}_n, b_w, b_l) \ , \tag{1}$$

where g is the number of links, $l_i \in \mathbb{R}$ is the length of each link, n is the number of actuators, and $\mathbf{a}_i \in \mathbb{R}^3$ is the actuator parameters. For linear actuators, $\mathbf{a}_i$ defines the 3D attachment points, while for rotary actuators, it corresponds to orientation of axis of rotation. Apart from these parameters that represent the kinematic tree morphology of the quadruped, we use two additional parameters b_w and b_l to represent the physical dimensions of the quadruped body (width and length respectively).

Likewise, the motion parameters $\mathbf{m} = (\mathbf{P}_1, \ldots, \mathbf{P}_T)$ are defined by a time-indexed sequence of vectors $\mathbf{P}_i$, where T denotes the time for each motion cycle. $\mathbf{P}_i$ is defined as:

$$\mathbf{P}_i = \left(\mathbf{q}_i, \mathbf{x}_i, \mathbf{e}_i^1, \ldots, \mathbf{e}_i^k, \mathbf{f}_i^1, \ldots, \mathbf{f}_i^k, c_i^1, \ldots, c_i^k, \right) \ , \tag{2}$$

where $\mathbf{q}_i$ defines the pose of the quadruped, i.e., the position, and orientation of the root as well as joint information such as angle values, $\mathbf{x}_i \in \mathbb{R}^3$ is the position of the quadruped's center of mass (COM), and k is the number of end-effectors. For each end-effector j, we use $\mathbf{e}_i^j \in \mathbb{R}^3$ to represent its position and $\mathbf{f}_i^j \in \mathbb{R}^3$ to denote the ground reaction force acting on it. We also use a contact flag c_i^j to indicate whether it should be grounded ($c_i^j = 1$) or not ($c_i^j = 0$).

3.1.2 Motion Optimization

The purpose of motion optimization is to take a quadruped design $\mathbf{s}$ and optmize its motion for user specified task while satisfying certain constraints. We used a cost function $F(\mathbf{s}, \mathbf{m})$ to encode the task specifications. We now describe how $F(\mathbf{s}, \mathbf{m})$ is constructed. To this end, we use a set of objectives that capture users' requirements, and constraints that ensure task feasibility.

Objectives We allow the users to define various high-level goals to be achieved by their quadruped designs such as moving in desired direction with specific speeds, different motion styles, etc. To capture the desired direction and speed of motion, we define the following objectives:

$$E_{\text{Travel}} = \frac{1}{2}||\mathbf{x}_T - \mathbf{x}_1 - \mathbf{d}^D||^2 \ ,$$
$$E_{\text{Turn}} = \frac{1}{2}||\tau(\mathbf{q}_T) - \tau(\mathbf{q}_1) - \tau^D||^2 \ , \tag{3}$$

where $\mathbf{x_i}$ is the quadruped's COM as defined in eq. 2, $\tau(\mathbf{q_i})$ is the turning angle computed from pose $\mathbf{q_i}$, while $\mathbf{d}^D$ and τ^D are desired travel distance and turning angles respectively. E_{Travel} ensures that the quadruped travels a specific distance in desired time, while E_{Turn} can be used to make a quadruped move on arbitrary shaped paths.

Motion style is highly effected by gait or foot-fall patterns that define the order and timings of individual limbs of a quadruped. We internally define various foot-fall patterns for different motion styles such as trotting, pacing, and galloping. When users select a specific motion style, our system automatically loads the necessary foot-fall patterns, thereby allowing novice users to create many expressive quadruped motions. Motion style is also effected by quadruped poses. For expert users, we allow the capability to specify and achieve desired poses, if needed, using the following objectives:

$$E_{\text{StyleCOM}} = \frac{1}{2}\sum_i^T ||\mathbf{x}_i - \mathbf{x}_i^D||^2 \ ,$$
$$E_{\text{StyleEE}} = \frac{1}{2}\sum_i^T \sum_j^k ||\mathbf{e}_i^j - \mathbf{e}_i^{j\,D}||^2 \ , \tag{4}$$

where k is the number of end-effectors, $\mathbf{x}_i^D$ and $\mathbf{e}_i^D$ represent desired quadruped COM, and end-effector positions respectively. Apart from these, motion smoothness is often desired by the users, which is encoded by the following objective:

$$E_{\text{Smooth}} = \frac{1}{2}\sum_{i=2}^{T-1} ||\mathbf{q}_{i-1} - 2\mathbf{q}_i + \mathbf{q}_{i+1}||^2 \ . \tag{5}$$

Constraints We next define various constraints to ensure that the generated motion is stable.

Kinematic constraints: The first set of constraints ask the position of COM, and end-effectors to match with the pose of the quadruped. For every time step i, and end-effector j:

$$\varphi_{COM}(\mathbf{q}_i) - \mathbf{x}_i = 0 \, ,$$
$$\varphi_{EE}(\mathbf{q}_i)^j - \mathbf{e}_i^j = 0 \, , \quad \forall j, \tag{6}$$

where φ_{COM} and φ_{EE} are forward kinematics functions outputting the position of COM and end-effectors respectively.

We also have a set of constraints that relate the net force and torque to the acceleration and angular acceleration of the quadruped's COM:

$$\sum_{j=1}^k c_i^j \mathbf{f}_i^j = M\ddot{\mathbf{x}}_i \ ,$$
$$\sum_{j=1}^k c_i^j (\mathbf{e}_i^j - \mathbf{x}_i^j) \times \mathbf{f}_i^j = \mathbf{I}\ddot{\mathbf{o}}_i \ , \tag{7}$$

where M is the total mass of the quadruped, and $\mathbf{I}$ is the moment of inertia tensor. The acceleration $\ddot{\mathbf{x}}_i$ can be evaluated using finite differences: $\ddot{\mathbf{x}}_i = (\mathbf{x}_{i-1} - 2\mathbf{x}_i + \mathbf{x}_{i+1})/h^2$, where h is the time step. Similarly, the angular acceleration $\ddot{\mathbf{o}}_i$ can be expressed as $\ddot{\mathbf{o}}_i = (\mathbf{o}_{i-1} - 2\mathbf{o}_i + \mathbf{o}_{i+1})/h^2$. We note that the orientation of the root $\mathbf{o}_i$ is part of the pose $\mathbf{q_i}$, and it uses axis-angle representation.

Friction constraints: To avoid foot-slipping, we also have the following constraints for each end-effector j:

$$c_i^j(\mathbf{e}_{i-1}^j - \mathbf{e}_i^j) = 0,\; c_i^j(\mathbf{e}_i^j - \mathbf{e}_{i+1}^j) = 0\,, \tag{8}$$

for all $2 \leq i \leq T-1$, which implies that the end-effectors are only allowed to move when they are not in contact with the ground. Further, to account for different ground surfaces, we enforce the friction cone constraints:

$$f_{i\,\parallel}^j \leq \mu f_{i\,\perp}^j\,, \tag{9}$$

where $f_{i\,\parallel}^j$ and $f_{i\,\perp}^j$ denote the tangential and normal component of $\mathbf{f}_i^j$ respectively, and μ is the coefficient of friction of the ground surface.

Limb collisions: For physical feasibility, we propose a collision constraint that ensures a safe minimum distance between the limb segments of quadruped over the entire duration of the motion.

$$d(\mathbf{V}_i^{k_1}, \mathbf{V}_i^{k_2}) \leq \delta\,, \tag{10}$$

where $\mathbf{V}_i^k$ represents a 3D segment representing the position and orientation of k^{th} limb, $d(\cdot)$ computes the distance between k_1 and k_2 limbs, and δ is the threshold distance beyond which collisions may happen.

Motion periodicity: If the users prefer a periodic motion, we can add an additional constraint that relates the start pose $\mathbf{q}_1$ and the end pose $\mathbf{q}_T$ of the quadruped:

$$\mathbf{J}(\mathbf{q}_T) - \mathbf{J}(\mathbf{q}_1) = 0\,, \tag{11}$$

where $\mathbf{J}(\mathbf{q_i})$ extract the orientation of the root and joint parameters from pose $\mathbf{q_i}$.

3.1.3 High-Resolution Motion

The motion planning algorithm described above mainly cares about the root and the end-effectors of the skeleton, and it does not optimize for the motion style of the limbs. Thus, for motion planning, we choose to use a reduced version of the modeled skeleton which only has two joints for each limb (Figure 2(a)). After the motion is generated, we do IK post-processing to match all joints (except intermediate limb joints) and end-effectors between the original high-resolution skeleton and the reduced one (Figure 2(c)).

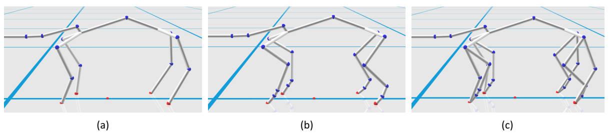


Figure 2: *(a) Low-resolution skeleton. (b) High-resolution skeleton. (c) Joint correspondence between low-resolution and high-resolution skeletons.*

IK post-processing We will just use front limb for the discussion. Since motion planning only produces the positions of the shoulder and the end-effector for each limb and we have two additional joints, *i.e.* wrist and finger, we need to add two parameters to constrain the limb and provide stylizing interface for the user. As illustrated in Figure 3(a), L is the distance from the elbow to the end-effector, which can help determine the elbow's position. θ is the angle between the finger and the upright direction, which infers the positions of the finger and the wrist. Additionally, Figure 3(b-c) tells us that there are two solutions for the elbow, and similarly for the wrist. Thus, we need two binary parameters choose which way we want the elbow and the wrist to bend. If we inspect the motion of real animals, we will find that their joint angles keep changing during a motion cycle. To mimic such behavior, we use a different set of L and θ when the limb is in full swing, and linearly interpolate these two sets of parameters for other motion phases.

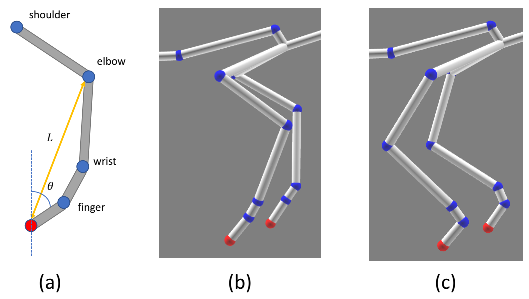


Figure 3: *(a) Illustration of the front limb. Two parameters L and θ are used to constrain the joints and stylize the limb motion. (b-c) Since their are two analytical solutions for the elbow position, a binary parameter is used to select one of them.*

3.2 Kinematic Controller and Adaptation

The real-time motions is produced with an animation controller which transitions and blends between motion clips based on two input parameters: the *speed* and *direction* of the character's root. The controller is a state machine holding idle, walk forward, and walk backward states. The walking states (forward and backward) each blend between 3 motion clips: left, straight and right. The parameters for blending between clips or transitioning between states are detailed in Figure 5, and are automatically created from the generated motion clips, which is described in the previous section.

This motion controller performs only motions over flat terrain, and cannot react naturally to pushes and perturbations. To walk over different terrains, we adapt the current frame of the animation using inverse kinematics (IK), based on the terrain height. The ground height is computed by raycasting from the ground foot position, as shown in Figure 4.

Finally, to have the character react realistically to physical perturbations, such as being pushed or hit, we added a simulated character (ragdoll) layer on top. For this, we used PuppetMaster ([PupperMaster]) which is a character's physics tool for automatically generating ragdolls for bipeds. It enables creating active ragdolls that can follow kinematic reference motions. We extended its ragdoll layer for quadruped characters, and used it for simulating reactions.

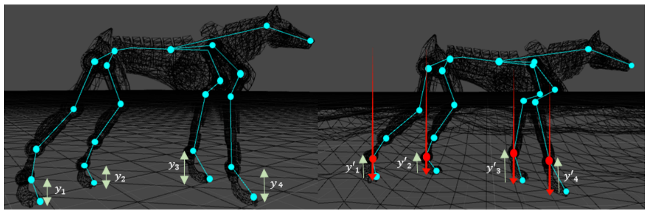
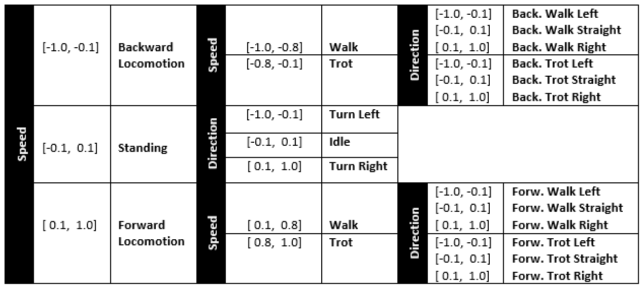


Figure 4: *Terrain adaptation is maintained by the estimation of the ground height at the position of the each feet by casting a ray and adding the feet offsets at the current animation frame.*



Speed	[-1.0, -0.1]	Backward Locomotion	Speed	[-1.0, -0.8]	Walk	Direction	[-1.0, -0.1]	Back. Walk Left
							[-0.1, 0.1]	Back. Walk Straight
							[0.1, 1.0]	Back. Walk Right
				[-0.8, -0.1]	Trot	Direction	[-1.0, -0.1]	Back. Trot Left
							[-0.1, 0.1]	Back. Trot Straight
							[0.1, 1.0]	Back. Trot Right
	[-0.1, 0.1]	Standing	Direction	[-1.0, -0.1]	Turn Left			
				[-0.1, 0.1]	Idle			
				[0.1, 1.0]	Turn Right			
	[0.1, 1.0]	Forward Locomotion	Speed	[0.1, 0.8]	Walk	Direction	[-1.0, -0.1]	Forw. Walk Left
							[-0.1, 0.1]	Forw. Walk Straight
							[0.1, 1.0]	Forw. Walk Right
				[0.8, 1.0]	Trot	Direction	[-1.0, -0.1]	Forw. Trot Left
							[-0.1, 0.1]	Forw. Trot Straight
							[0.1, 1.0]	Forw. Trot Right

Figure 5: *A blending diagram is automatically created from the generated motion clips that controls the motion transition using parametric inputs- speed and direction (of root).*

4 3D reconstruction

We describe our approach to understanding the environment for AR purposes. Because current consumer level hardware devices such as the Hololens only offer coarse reconstructions online, we cannot use them for having characters walk over as they appear to be floating in air.

Hence, we developed pre-defined objects that are recognized and localized in space using feature-based technology (Vuforia Engine [Vuforia 2017]). For each real world object, we define a corresponding 3D digital geomtric counter-part that matches in shape and size. Then, we scan the real world object from all directions using an RGB camera to obtain a data-base of image-based features and transformation pairs. At runtime, Vuforia searches for matching features and returns the id of the object, together with its transformation that we apply to the 3D object in the scene.

We encountered a few issues recognizing objects with Vuforia. One problem is when the objects are transparent, or have plain textures. In this case, the lack of features causes the recognition to fail. Similarly to object recognition, objects with shiny and reflective properties do not give successful image recognition and tracking. Hence for some objects, we add a rich texture on top to make them distinguishable, as shown in our figures bellow.

5 Interactions

We take our animated character together with its ability to navigate the real world environment, and design AR interactions in 3D. In particular, we provide ways for the user to specify where the character should go, ways to have 3D virtual and real objects collide with the character, as well as ways to configure different terrains.

Specifying trajectories via touch on the screen. The quadruped character can be directed through any arbitrary paths in the physical environment, over different terrains. The paths are generated by *projecting* onto the environment, the user's fingertip when drawing on the touchscreen, as shown in Figure 6.

Walking over different real-world slopes. The virtual character's behavior depends on the purpose of the interaction. Therefore we label objects either as terrain or non-terrain in order for the motion model to behave in the correct manner. We label the terrain automatically by defining objects as terrain if their height is bellow a certain threshold, that corresponds to the maximum height the character can climb.

Different slopes and platforms can be formed with different arrangements of the objects as shown in Figure 7. While the character's motion model will only employ the inverse kinematics for adapting to objects labeled as terrain, it should react differently for the other objects, as described next.

Pushing characters with real-world objects. Non-terrain objects can be used for interactions like colliding with or pushing the quadruped, as shown in Figure 8. For animating the reactions, a ragdoll simulation (un-controlled passive dynamics) is activated for a short period of time, letting the character react to the perturbation. After the short period of time, the state of the simulated (ragdoll) character is blended back into the animation state over another small window of time. Completely switching to a ragdoll simulation causes to the character to fall. Hence, above a certain force threshold, we do not blend back to the animation and simply let the character flow.

Interacting with virtual objects. We also experimented the interactions between the character and virtual objects. We designed a simple platformer game (which is shown in our accompanying video), where the user can use various *props* to carry the virtual character from the beginning to the end of the platform puzzle, while trying to prevent him from falling. The character only moves forward, and its moving direction can only be changed if it hits a *wall prop*. When the character meets an *elevator platform* which goes up and down, the user needs to use a *fan prop* (shown in Figure 10) to stop the character such that it can wait. For an increased challenge, we added an *enemy cannon* which shoots 3D balls at the character, possibly causing it to fall from the platform, as shown in Figure 9.

6 Discussion and Conclusion

While we did a first step in the direction of having intelligent character we can interact with in AR, characters that can understand their environments and navigate them, our system has a few limitations. First we can only interact slowly with objects, as the tracking is remains at low frequency. We use pre-defined 3D objects instead of scanning the world. We believe that both of these issues will improve with the evolution of hardware.

In the future, we plan on investigating chameleon technology with in-painting, allowing real-world characters to "come-to-life", by replacing their background and animating their virtual counter-parts. We also plan on integrating additional components to our character's intelligence, such the ability to talk and reason about the objects in the world.

References

ARIKAN, O., FORSYTH, D. A., AND O'BRIEN, J. F. 2003. Motion synthesis from annotations. *ACM Transasctions on Graphics 22*, 3, 402–408.

BHATTI, Z., SHAH, A., WAQAS, A., AND KARBASI, M. 2015. Automated animation of quadrupeds using procedural program-

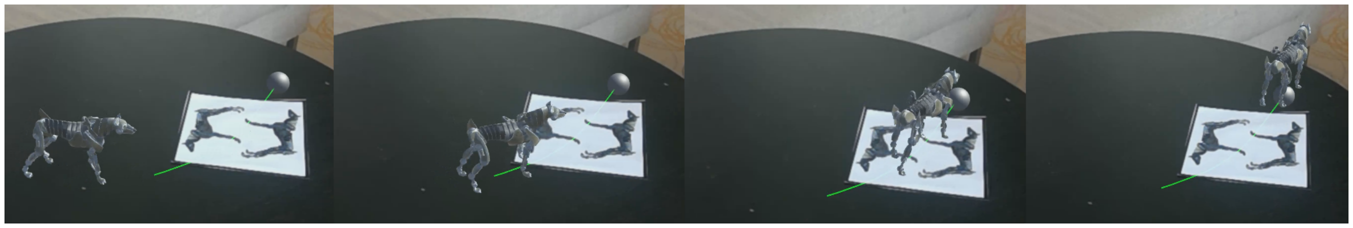


Figure 6: *Path drawing with touch is used to direct the character in the physical environment.*

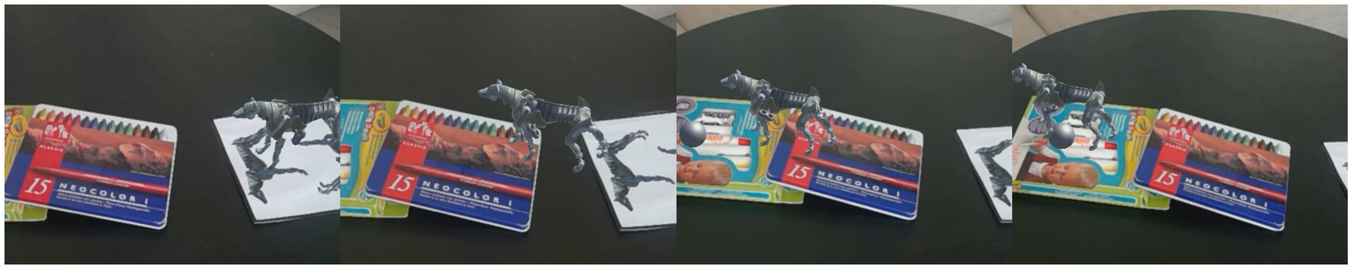


Figure 7: *Different arrangements of predefined physical objects creates different slopes for the character to walk on. For the details of the 3D object reconstruction, we kindly refer to Section 4.*

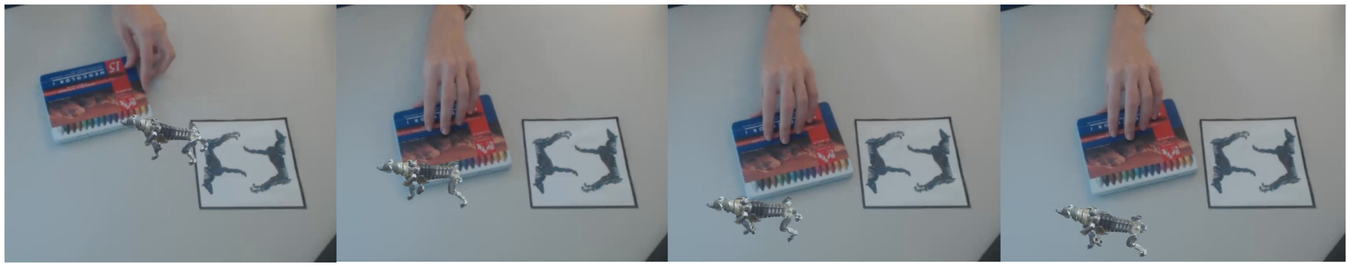


Figure 8: *The character reacts to the pushes by real objects.*

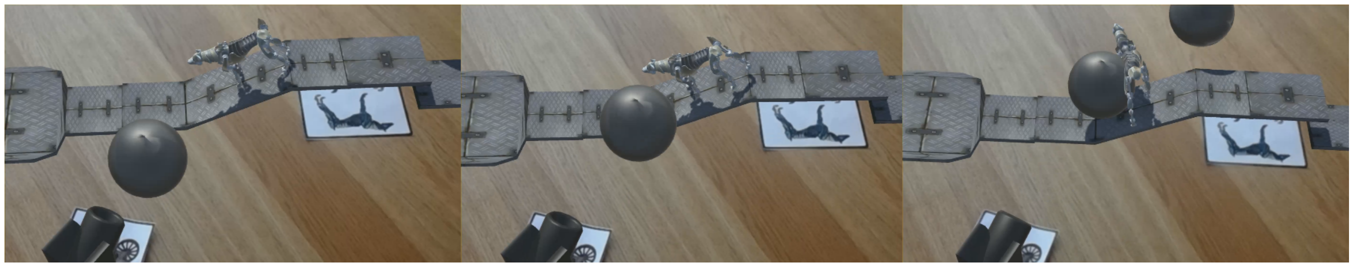


Figure 9: *During the platformer game, the character can be hit by a virtual cannon ball and fall down.*

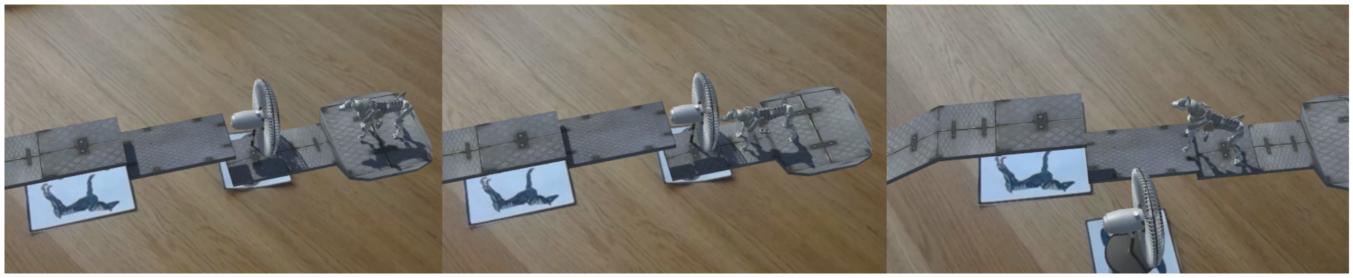


Figure 10: *A virtual fan can be used to stop the character which creates virtual forces.*

ming technique. In *Asian Journal of Scientific Research*, vol. 8, 165–181.

BILLINGHURST, M., KATO, H., AND POUPYREV, I. 2001. The magicbook - moving seamlessly between reality and virtuality. *Computer Graphics and Applications, IEEE 21*, 3 (May), 6–8.

GRASSET, R., DÜNSER, A., AND BILLINGHURST, M. 2008. Edutainment with a mixed reality book: A visually augmented illustrative childrens' book. In *Proceedings of the 2008 International*

Conference on Advances in Computer Entertainment Technology, ACM, New York, NY, USA, ACE '08, 292–295.

HOLOLENS, 2017. Microsoft, https://www.microsoft.com/en-us/hololens/.

KIM, H., TAKAHASHI, I., YAMAMOTO, H., MAEKAWA, S., AND NAEMURA, T. 2014. Mario: Mid-air augmented reality interaction with objects. *Entertainment Computing 5*, 4, 233 – 241.

KOKKEVIS, E., METAXAS, D., AND BADIER, N. I. 1995. Autonomous animation, control of four-legged animals. In *Proceedings of Graphics Interface '95*, GI '95, 10–17.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '02, 473–482.

MEGARO, V., THOMASZEWSKI, B., NITTI, M., HILLIGES, O., GROSS, M., AND COROS, S. 2015. Interactive design of 3d-printable robotic creatures. *ACM Trans. Graph. 34*, 6 (Oct.), 216:1–216:9.

PUPPERMASTER. Rootmotion, http://root-motion.com/.

SCHERRER, C., PILET, J., FUA, P., AND LEPETIT, V. 2008. The haunted book. In *ISMAR*, IEEE Computer Society, 163–164.

TORKOS, N., AND VAN DE PANNE, M. 1998. Footprint–based quadruped motion synthesis. In *Proceedings of the Graphics Interface 1998 Conference, June 18-20, 1998, Vancouver, BC, Canada*, 151–160.

VUFORIA, 2017. Qualcomm, http://www.qualcomm.com/vuforia.