The Robot Engine - Making The Unity 3D Game Engine Work For HRI

Christoph Bartneck, Marius Soucy, Kevin Fleuret, Eduardo B. Sandoval[1]

Abstract— **HRI is a multi-disciplinary research field and integrating the range of expertise into a single project can be challenging. Enabling experts on human behavior to design fluent animations and behaviors for advanced robots is problematic, since the tools available for such robots are often in their prototype stage. We have built The Robot Engine (TRE) based on the Unity 3D Game Engine to control robots with Unity 3D. Unity 3D allows non-programmers to use a set of powerful animation and interaction design tools to visually program and animate robots. We review several animation techniques that are common in computer games and that could make the movements of robots more natural and convincing. We demonstrate the use of TRE with two different Arduino based robot platforms and believe that it can easily be extended for use with other robots. We further believe that this unconventional integration of technologies has the potential to fully bring the expertise of interaction designers into the process of advanced human-robot interaction projects.**

I. Introduction

The field of Human-Robot Interaction (HRI) is inherently multidisciplinary [1]. Researchers and developers from the areas of engineering, computer science, psychology and design need to work together to make HRI projects a success. The HRI Conference has itself acknowledged its multidisciplinary nature by introducing "Themes" and their associated committees. Working in multidisciplinary teams can be challenging and one of the biggest hurdles is that experts on human behavior and interaction design often do not have the technical skills to program the advanced robots the engineers are constructing. When working at the cutting edge of engineering and computer science there are often only prototype hardware and software packages available. These more often than not require classical programming in order to control the interaction with humans. Once a platform matures and becomes more widely available the associated tools also become more sophisticated and easier for human behavior experts and interaction designers to use.

An example of mature hardware and software is the NAO platform from Aldebaran. Its Choregraphe software [2] enables users to control the behavior of the robot without writing a single line of code. Choregraphe is currently the golden standard for easily animating and programming robots, but it works with only a single robot: NAO. It is therefore not suitable for HRI projects that develop their own robotic hardware or that use robotic hardware from other manufacturers. The LEGO Mindstorms platform is even easier to use and is targeted at children and young adults. This ease of use is required to enable interaction designers to unfold their full potential in HRI projects. Currently, these experts are forced to use tools that are as difficult to them as it would be for a computer scientist to program robots using assembly language.

In this paper we introduce a new approach to animating robots and programming the interaction with users: The Robot Engine (TRE). The design principle of TRE follows the First Rule of Design: "Do not design what you can copy". We therefore use the powerful animation and programming tools of a game engine to give interaction designers the tools they need. TRE then connects the movements in the virtual world to the motors and sensors of the real world. This approach is so simple and powerful that we ourselves are surprised that it has not yet already been widely used. We believe that TRE has the potential to bridge the gap between engineers and computer scientists on the one side and psychologists and designers on the other.

First we will review existing animation and interaction design software that is being used in HRI. Second, we will describe the TRE system and present two case studies. Last, we will discuss the limitations of TRE and its future development.

II. HRI Software

There are several middle-ware solutions for robots, such as the Robot Operating System (ROS) [3] or Microsoft's Robotics Developers Studio (MRDS) [4]. These frameworks offer libraries and tools to help developers create their own robot control applications. To use these frameworks developers need to know programming languages, such as C++ or Python. They therefore do not overcome the fundamental gap described above.

A. Programming Languages

There are many different programming languages that have evolved over time, though an argument can be made that the most dramatic inventions were made in the seventies. Without going into the history and classification of programming languages (the interested reader may wish to consult [5]), we would like to share our observation that the languages being used in HRI vary in their abstraction levels. Abstraction levels are typically correlated to the ease of use for non-programmers, meaning that the more low- level languages, such as C, are harder for interaction designers to use than higher level languages, such as Java. A popular solution for robots is the combination of Processing, a simplified

*This work was not supported by any organization

[1]HIT Lab NZ, University of Canterbury, Private Bag 4800, 8140 Christchurch, New Zealand. christoph.bartneck@canterbury.ac.nz

version of Java, with the Arduino micro controllers. A large community of "Makers" uses this powerful combination to develop do-it-yourself projects, and textbooks are even available [6]. Visual Programming Languages (VPL), such as LabView [7], Scratch [8], and MAX [9] go one step further by using graphical elements on the screen to capture the common elements of programming languages [10]. These VPLs are easy to learn and enable children to program robots such as LEGO Mindstorms.

However, the animation of robots escapes the boundaries of programming languages completely. The human mind is highly trained to perceive biological movement. This sensitivity challenges even the most sophisticated computer animation tools and today's computer characters are still being animated using motion tracking. A robot does not necessarily need to exhibit natural biological movements, but the fact remains that social gestures and body language are very difficult to program directly using mathematical functions such as sine waves.

Following the First Rule of Design it appears useful to investigate what tools are being used to animate and control the robots' next of kin: computer game characters. One could argue that social robots are essentially computer game characters on wheels. These characters also need to be animated and have to interact with the user. Computer game characters are in many ways easier to control since they operate in a virtual environment in which the noise of sensors plays little to no role at all.

B. Game Engines

Years ago, computer game developers faced the same problem that HRI researchers face today: how to enable the experts to make a game fun, and how to quickly develop games without having to program directly in low-level languages. Their solution was to develop Game Engines that include graphical editors for not only creating the visual artwork, but also animating it and scripting the interaction with the users. Similar to Choregraphe, these game engines used to be specific to hardware platforms and not open source. Several Game Engines, such as the Unreal Engine, CryEngine and Unity 3D became available and possibly due to the increased competition, the Game Engines became multi-platform, Open Source, and freely available for education and research. There is hope that the HRI software will follow a similar trajectory.

Game Engines are not only useful for developing games; they have been successfully used in scientific research [11]. They are already in use for robotics, in particular for simulations [12] and visualizations, such as MORSE [13]. This comes as no surprise, since the RoboCup competition even features a "Soccer Simulation League". The USARSim can be used to simulate and visualize robots and their environment for such competitions [14]. Ohashi et al used a Game Engine to remotely control a robot through the internet [15]. It is important to note that already today Game Engines include modules that allow the game to process input not only from the mouse and keyboard, but also from cameras and microphones. Games can be programmed to be controlled using gestures and speech. Controlling hardware, however, is far less developed. It has therefore been the main focus for the development of TRE.

III. The Robot Engine

Following the First Rule of Design we built TRE on top of a Game Engine. There were several Game Engines that could potentially be used, such as Unreal Engine, CryEngine and Unity 3D. These Game Engines largely compete on features that are not necessarily relevant to HRI, such as their ability to quickly draw polygons on the screen, but for the purposes of HRI the following features are of importance:

1) An easy-to-use graphical user interface for animating objects and controlling interaction
2) The ability to communicate with external hardware
3) The ability to process multimedia sensory data
4) Being freely available for research and education
5) The ability to offer support for multiple operating systems

We decided to use Unity 3D for TRE because it not only fulfilled all these requirements; there is a strong developer community supporting its progress. Unity 3D has a plug-in architecture that allows these developers to extend the core functionality, although this is restricted to the "Pro" version. It should be noted that there are already plug-ins available to recognize speech, understand the users' gestures using Microsoft's Kinect, and use advanced computer vision methods by integrating OpenCV. We do not claim to have made the best possible choice, but during the development of TRE we did not encounter any major problems and hence feel confident that the choice was good enough.

Our main technical contribution was to develop a plug-in to communicate with sensors and actuators from within Unity 3D. In addition we integrated several available plug-ins, such as a speech recognizer and a computer vision library to enable our robots to process sensory information. We thereby made a proof of concept for the power of using Game Engines for controlling the interaction between humans and robots.

A. System Architecture

In this paper we cannot provide a full introduction to the structure and operation of Unity 3D. We have to constrain ourselves to provide a short overview of TRE and its associated components. TRE consists of the MainScript, ModelScript, SerialCommunicationScript and the AnimatorScript (see Figure 1).

The MainScript has two functions. First, it manages all other scripts, modules and graphical user interface elements. It (de)activates modules, such as the speech recognition, text-to-speech (TTS) module and a computer vision module. The speech recognition module is based on Miscrosoft Kinect and requires the Kinect SDK to be installed. The TTS system is based on Microsoft Windows Voice and the computer vision module is based on OpenCV. Optional modules include and interface to Leap Motion, a device that can track

Bartneck, C., Soucy, M., Fleuret, K., & Sandoval, E. B. (2015). The Robot Engine - Making The Unity 3D Game Engine Work For HRI. Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN2015), Kobe pp. 431 - 437. DOI: 10.1109/ROMAN.2015.7333561

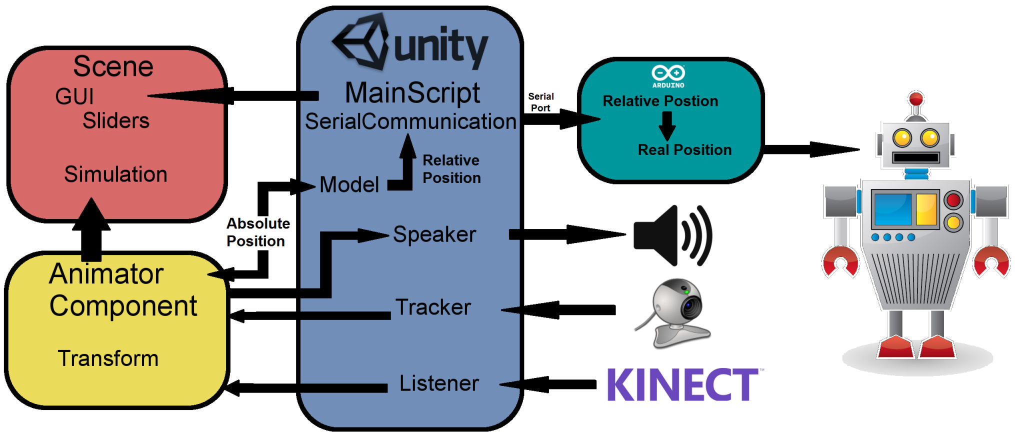


Fig. 1. Flow diagram

hand gestures. The Unity 3D assets store has many more modules that can be integrated. Second, MainScript connects the scripts and modules with the GameObjects. In Unity 3D, GameObjects are all the elements of a game, such as characters, scenes and props [16]. In the case of TRE, every part of the robot in the 3D model is a GameObject with particular characteristics, such as eyes, head, arms and torso.

The ModelScript manages all the mechanical parts of the robot associated with the GameObjects in the 3D model. In the case of the InMoov robot, there are 26 GameObjects matching the 26 main parts of the robot, such as the torso or head. The main function of the ModelScript is to convert the absolute positions of the GameObjects into the relative positions needed for the robot. The head, for example, will have an absolute position and orientation in the game world, but what the robot needs is the relative angle between the head and the torso. From this the desired angle for the servo motors can be calculated. Furthermore, the ModelScript translates the constraints that the physical robot has into the virtual robot. The head, for example, may only be able to rotate 180 degrees. This constraint must also be introduced into the virtual robot so that it does not perform movements that would be impossible for the physical robot to execute. The ModelSCript sends the appropriate angles to the SerialCommunicationScript to the Arduino board, which is attached to the computer through a USB port. The SerialCommunicationScript also handles the mapping of the GameObjects to the servo PINs on the Arduino board. It determines, for example, that the head GameObject is mapped to the servo attached to PIN number one on the Arduino board. A small software script on the Arduino simply directs the servos to the desired angle.

The AnimatorScript manages the Animator component that is part of Unity 3D's Mecanim Animation System. It manages the animations of the GameObjects and sends the positions and angles of the GameObjects to the ModelScript. It also catches keyboard presses and maps them to the graphical user interface elements, such as sliders.

TRE also includes two scripts that need to be executed on the Arduino boards. The debugging script is used for testing the wiring and mapping of the servo motors, and the control script then takes the commands from the SerialCommunicationScript and moves the servo motors into the desired position.

IV. PROCESS

The first step is to build your physical robot. You may consider 3D printing one of the openly available robots, but you may also have your own ideas for an HRI project. For the two robots described in the case studies below we used Arduino micro controllers that are able to control up to 30 servo motors. Figure 2 shows a typical wiring of Arduino controllers, a battery and several servo motors. When more servo motors are connected it may become necessary to power the Arduino boards with a 5V power supply.

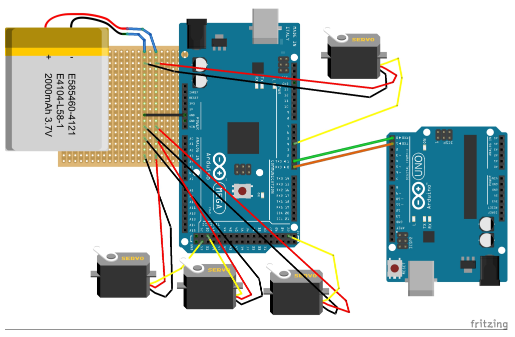


Fig. 2. Example of Arduino wiring

For the development of your robot you will be likely to have used 3D modelling software to prepare the manufacturing of all the required parts. Figure 3 shows the construction of a LEGO Robot in the 3D modelling software Blender. This free software is a popular tool for 3D modelling and animation but it is not a Game Engine in itself. Unity 3D is able to import a large variety of 3D file types and can then serve as a starting point for constructing the humanoid avatar

Bartneck, C., Soucy, M., Fleuret, K., & Sandoval, E. B. (2015). The Robot Engine - Making The Unity 3D Game Engine Work For HRI.
Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN2015), Kobe pp. 431 - 437.
DOI: 10.1109/ROMAN.2015.7333561

in Mecanim. Note that humanoid does not necessarily mean that the robot must have a human shape; it simply means a torso and four extremities. Alternative body shapes can also be animated using inverse kinematics, but this requires additional work.

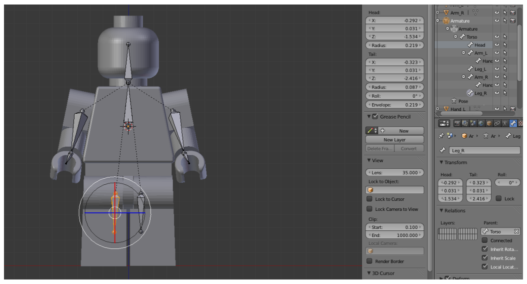
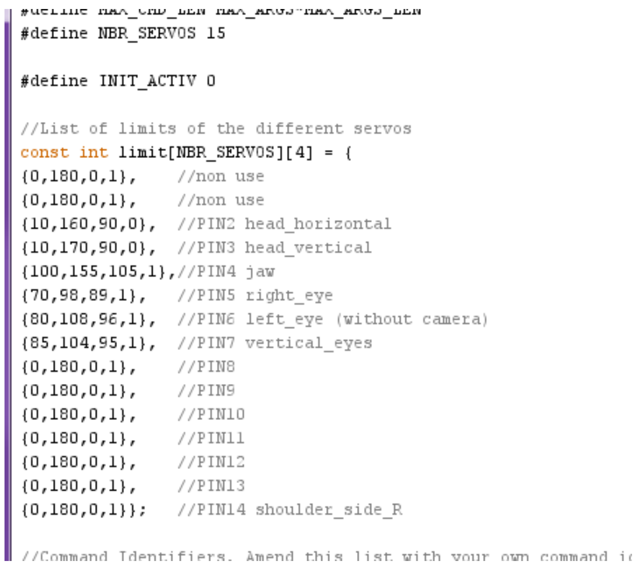


Fig. 3. 3D modeling with Blender

The second step is to create a new project in Unity 3D and to import the TRE package. Next, you can either import the already existing 3D geometry files or create the 3D geometry in Unity 3D. Unity 3D has a good set of tools for free modeling and it is matter of personal preference whether you want to model the geometry in Unity 3D or another 3D program. The next step is to place all the 3D models into the scene, which automatically turns them into GameObjects. You need to define the parameters in the ModelScript to describe the mechanical constraints of your physical robot, and then the scripts need to be associated to the appropriate GameObjects (see Figure 4.

The third step is to run a debugging script on the Arduino to test the wiring and its connectivity to Unity 3D. You can then use the confirmed configuration to configure the Arduino control script. The next step is to define the PIN numbers in the Arduino script. Figure 5 shows an example of a configuration. Each line describes one servo motor and consists of a pin number followed by four values. The first value is the minimum angle of the articulation, the second is the maximum, the third is the initial position, and the last is the option to reverse the rotational direction.

The TRE is now ready for use and you can use Unity 3D's powerful animation tools to create convincing expressions. You only need to create an Animator controller and attach it to your GameObject. You may also want to add additional modules to the scene by connecting the modules to the appropriate GameObjects in the scene. TRE already provides access to text-to-speech, speech recognition and computer vision. Once you are finished with the design of your animations, behaviors and interactions with users you can compile your Unity 3D project into an executable file that can be played on many different platforms, but note that due to the dependencies of some plug-ins on certain libraries such as the Kinnect SDK, not all platforms may be suitable for the distribution of your HRI project. Unity also has a build in collision control that enables the programmer to trigger events in case an animation would result in a situation in which the robot makes moves that could harm itself.


```
#define MAX_CMD_LEN MAX_ARGS*MAX_ARGS_LEN
#define NBR_SERVOS 15

#define INIT_ACTIV 0

//List of limits of the different servos
const int limit[NBR_SERVOS][4] = {
{0,180,0,1},    //non use
{0,180,0,1},    //non use
{10,160,90,0},  //PIN2 head_horizontal
{10,170,90,0},  //PIN3 head_vertical
{100,155,105,1},//PIN4 jaw
{70,98,89,1},   //PIN5 right_eye
{80,108,96,1},  //PIN6 left_eye (without camera)
{85,104,95,1},  //PIN7 vertical_eyes
{0,180,0,1},    //PIN8
{0,180,0,1},    //PIN9
{0,180,0,1},    //PIN10
{0,180,0,1},    //PIN11
{0,180,0,1},    //PIN12
{0,180,0,1},    //PIN13
{0,180,0,1}};   //PIN14 shoulder_side_R

//Command Identifiers. Amend this list with your own command i
```
Fig. 5. Placement of the motors limits

A. Case Study 1: LEGO Fireman

The first robot we controlled using TRE was Fireman. We bought a LEGO LED torch in the shape of a Fireman (see Figure 6). We removed all the internal components, such as battery compartment, switch and LEDs. We placed servo motors in the torso, head, arms and legs, resulting in six degrees of freedom. In addition we placed a USB webcam with a microphone in the head. The LEGO Fireman was mounted on a base, which also enclosed an Arduino micro-controller and a speaker. This robot is a typical example of a simple robot that can be used as a conversational agent.

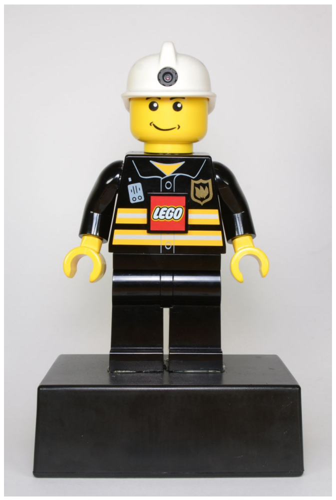
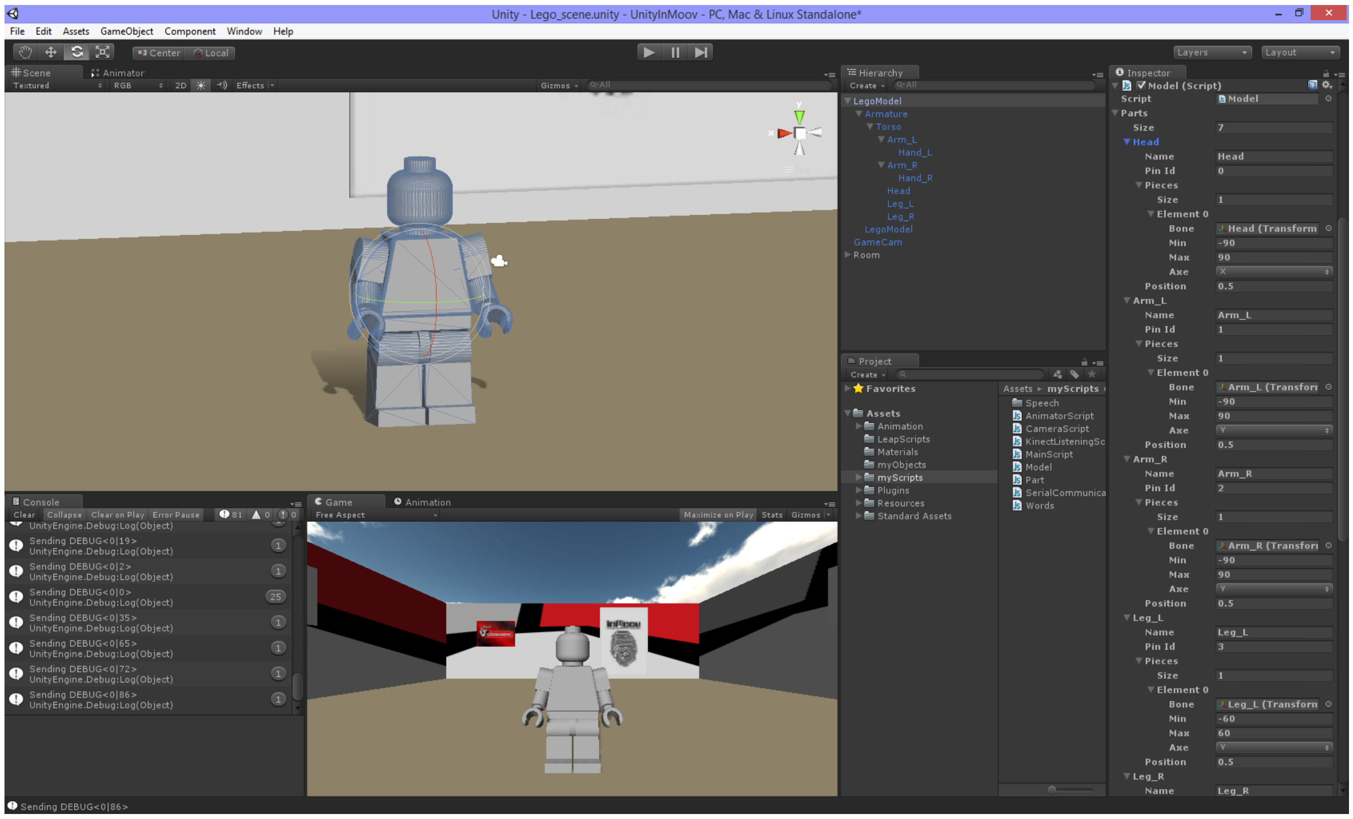


Fig. 6. The LEGO Fireman Robot

We created a 3D model of the LEGO Fireman using Blender and imported it into Unity 3D (see figure 7. After configuring the TRE modules we could animate the robot using the virtual robot. We implemented a simple face tracking behavior in which the robot used its head and hip rotation to follow the face of a user. The full documentation

Bartneck, C., Soucy, M., Fleuret, K., & Sandoval, E. B. (2015). The Robot Engine - Making The Unity 3D Game Engine Work For HRI. Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN2015), Kobe pp. 431 - 437. DOI: 10.1109/ROMAN.2015.7333561



```
Console
Clear  Collapse  Clear on Play  Error Pause   81  0  0
UnityEngine.Debug:Log(Object)
Sending DEBUG<0|19>
UnityEngine.Debug:Log(Object)                1
Sending DEBUG<0|2>
UnityEngine.Debug:Log(Object)                1
Sending DEBUG<0|0>
UnityEngine.Debug:Log(Object)               25
Sending DEBUG<0|35>
UnityEngine.Debug:Log(Object)                1
Sending DEBUG<0|65>
UnityEngine.Debug:Log(Object)                1
Sending DEBUG<0|72>
UnityEngine.Debug:Log(Object)                1
Sending DEBUG<0|86>
UnityEngine.Debug:Log(Object)                1

Sending DEBUG<0|86>
```

```
Model (Script)
Script            Model
Parts
  Size            7
  Head
    Name          Head
    Pin Id        0
    Pieces
      Size        1
      Element 0
        Bone      Head (Transform
        Min       -90
        Max       90
        Axe       X
    Position      0.5
  Arm_L
    Name          Arm_L
    Pin Id        1
    Pieces
      Size        1
      Element 0
        Bone      Arm_L (Transforı
        Min       -90
        Max       90
        Axe       Y
    Position      0.5
  Arm_R
    Name          Arm_R
    Pin Id        2
    Pieces
      Size        1
      Element 0
        Bone      Arm_R (Transforı
        Min       -90
        Max       90
        Axe       Y
    Position      0.5
  Leg_L
    Name          Leg_L
    Pin Id        3
    Pieces
      Size        1
      Element 0
        Bone      Leg_L (Transforn
        Min       -60
        Max       60
        Axe       Y
    Position      0.5
  Leg_R
    Name          Leg_R
```

Fig. 4. Unity project settings

of this project is available [1].

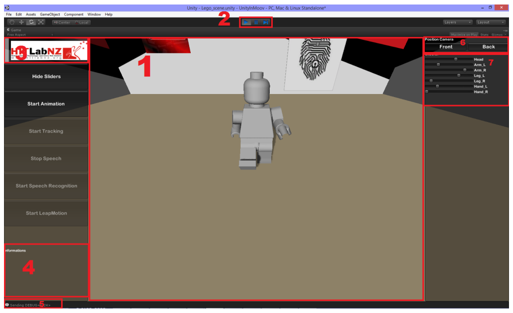


Fig. 7. Lego Robot in Unity

B. *Case Study 2: InMoov*

The original development of robotic hardware typically required expensive machine tools and expert knowledge. The arrival of 3D printers in combination with the development of easy to use micro controllers, such as the Arduino boards, enabled enthusiastic amateurs to build their own robots. Similar to open source software we are now encountering the first open source hardware projects. The Intel Company recently announced its own 3D printable robot called Jimmy and more companies are expected to come forward with their own robotic hardware soon. The InMoov robot [17] is another example of such an open source hardware robot. Anybody can download the 3D files and blueprints and start printing their own InMoov robot.

We decided to engage in this open source hardware robotic development and printed an InMoov robot (see figure 8). Printing all the required parts was a considerable task and we even decided to print some of the moving parts on a professional 3D printer to reduce friction. Our InMoov has 25 degrees of freedom and includes two cameras, a Microsoft Kinect and a speaker.

We imported the 3D files provided for the InMoov robot into the Blender software. From there we exported it to a format compatible with Unity 3D (see figure 9). Similar to the LEGO fireman we configured the TRE models to fit this specific hardware configuration. Afterwards we were able to fully control the InMoov robot using Unity 3D. We also implemented face tracking using the two degrees of freedom of the head.

V. Evaluation

We have not been able to run a fully controlled user study for TRE but we did develop a set of tutorials that enables users familiar with Unity 3D to quickly install and use it. We asked two members of our lab who had not previously been involved in the project to follow these instructions, and based on their feedback were able to identify several issues with the manual and with TRE. We have made the necessary changes and are now of the opinion that they are sufficiently

[1] `http://bartneck.de/publications/2015/unity/index.html`

Bartneck, C., Soucy, M., Fleuret, K., & Sandoval, E. B. (2015). The Robot Engine - Making The Unity 3D Game Engine Work For HRI. Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN2015), Kobe pp. 431 - 437. DOI: 10.1109/ROMAN.2015.7333561

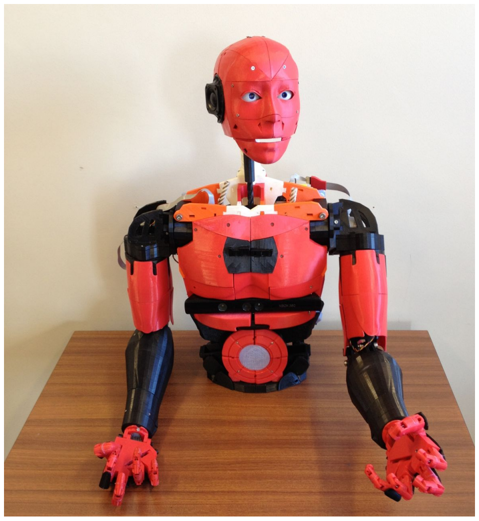


Fig. 8. InMoov Robot

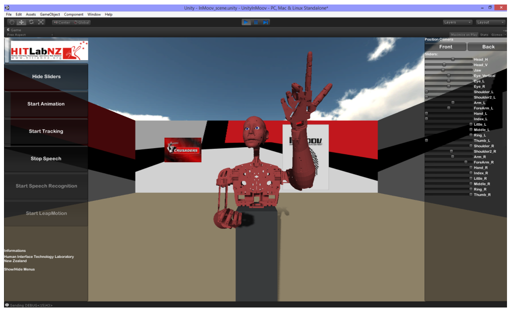


Fig. 9. InMoov robot in Unity

useful for others to follow. This does not of course replace a full usability evaluation or a controlled experiment, but for this stage of the project the evaluation was most informative [18]. At a later stage in the project other forms of evaluation will become the preferred option.

VI. Conclusions

We have developed The Robot Engine (TRE), a plug-in for the Unity 3D Game Engine. TRE enables users with little to no programming experience to animate robots and to control the interaction with humans. The Unity 3D Game Engine has a powerful set of animation tools that go beyond what Choregraphe has to offer. The tools that are being used to develop the most advanced computer games are now at the fingertips of HRI researchers. The Graphical User Interface of Unity 3D is also designed with creative people in mind; it is an excellent tool for interaction designers to take control of human-robot interaction. Like many other Game Engines, Unity 3D is for free for research and educational purposes and it is able to compile executable programs for different hardware platforms. It also features a large number of plug-ins for human-computer interaction that are also applicable to human-robot interaction, such as text-to-speech, speech recognition and computer vision. TRE does seem to be an ideal tool for the democratization of the robot development. Many "Makers" are already using Arduino and 3D printers to develop interactive social robots. With TRE they now also have an easy to use tool for designing their interaction.

One advantage of using a Game Engine as the control software is that it is not only possible to develop stand alone software; it is also possible to easily build graphical user interfaces to directly control the robot in real time. Given the large number of Wizard of Oz types of studies in the field of HRI, this might be a considerable advantage.

Another advantage of using a Game Engine as the animation tool for robots is that it already contains many tools that have been developed specifically for animating biological life forms. Unity 3D's Mecanim animation system, for example, has a dedicated Humanoid Avatar option. Using a Humanoid Avatar allows you to map your specific robot to a more general humanoid body shape. Once this mapping is completed, a huge library of readymade motions and behaviors can be applied to the robot. There are, for example, several idle and pointing animations. Moreover, Mecanim includes an Animation State Machine, which allows the animator to define the transitions between animation states. A robot might have restrictions on the next animation state it can go to rather than being able to switch immediately from any state to any other. The states and transitions of a state machine can be represented using a graph diagram within Unity 3D, where the nodes represent the states and the arcs represent the transitions (see Figure 10).

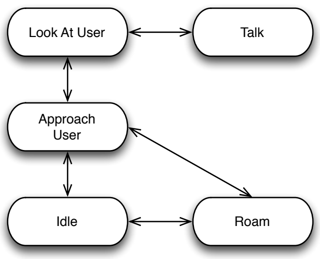


Fig. 10. Animation State Graph

A common task in game animation that is also highly relevant for robot animation is to blend two or more similar motions. Perhaps the best known example is the blending of walking and running animations according to the character's speed. Another example is a character expressing a blend of different emotions through its face. The Mecanim system allows animations to be blended into each other.

Another game character animation technique that is useful for animating robots is Inverse Kinematics. The "bones" of the virtual robot are connected and constrained thought the Animator module. It is then possible, for example, to position the hand of the robot in space with the lower and upper arm following accordingly, thereby making it sufficient to animate only the hand instead of all parts of the arm. All of these animation techniques are commonly used in computer games and with TER have now also become available for animating robots. We believe that these interaction techniques can make a powerful contribution towards more natural and pleasing robotic movements.

While Game Engines have been used for robotics before, they have not been widely used for the animation of robots and the design of their interaction with humans. We are almost embarrassed to propose such an obviously useful integration of technologies for fear that we might have failed to understand why this would be a bad idea. But our two case studies have convinced us that there are no clear reasons for not using a Game Engine for controlling social robots. The technical challenges of developing TRE were modest, but we believe that our unconventional integration of technologies can empower non-programmers to fully integrate their expertise in interaction design and human behavior into the development process of advanced HRI projects.

A. Limitations and future work

At this point, TRE only communicates to the Arduino micro-controller, but it would be simple to extend it to talk to other micro-controllers using the serial port. We intend to enable more than one Arduino to receive commands from Unity 3D. It will also be useful in the future to be able to receive sensory information through sensors attached to the Arduino board, rather than as right now, where the sensors such as cameras and microphones are being attached to the computer directly. We also intend to connect other equipment to TRE, such as Leap Motion module and joysticks, and it would additionally be desirable to be able to connect open source speech software, such as Sphinx and Festival.

Moreover, it might be desireable for TRE to connect the Robot Operating System (ROS) middleware so that more robots can take advantage of the animation tools of unity. ROS also offers the option to access motion planners, inverse kinematics solvers and collision checker modules.

While Unity 3D is a powerful animation and interaction design tool, its complexity might challenge users at the beginning. Unity 3D is, however, one of the best documented Game Engines and there are many tutorial videos and community support forums. But with its complexity also comes an enormous opportunity of designing beautiful movements. We must also acknowledge that once Choregraphe becomes available for other robotic platforms then TRE might lose its competitive edge.

References

[1] C. Bartneck, "The end of the beginning - a reflection on the first five years of the hri conference," *Scientometrics*, vol. 86, no. 2, pp. 487–504, 2011.

[2] E. Pot, J. Monceaux, R. Gelin, and B. Maisonnier, "Choregraphe: a graphical tool for humanoid robot programming," in *RO-MAN 2009 - The 18th IEEE International Symposium on Robot and Human Interactive Communication*. IEEE, Sept. 2009, pp. 46–51.

[3] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5. [Online]. Available: http://pub1.willowgarage.com/~konolige/cs225B/docs/quigley-icra2009-ros.pdf

[4] J. Jackson, "Microsoft robotics studio: A technical introduction," *Robotics & Automation Magazine, IEEE*, vol. 14, no. 4, pp. 82–87, 2007.

[5] G. O'Regan, "History of programming languages," in *A Brief History of Computing*. Springer London, 2012, pp. 121–144.

[6] J. Noble, *Programming Interactivity: A Designer's Guide to Processing, Arduino, and Openframeworks*. O'Reilly Media, Inc., 2009.

[7] N. Ertugrul, "Towards Virtual Laboratories : a Survey of LabVIEW-based Teaching / Learning Tools and Future Trends," vol. 16, no. 3, pp. 171–180, 2000. [Online]. Available: http://www.ijee.ie/articles/Vol16-3/ijee1116.pdf

[8] A. Ruthmann, J. M. Heines, G. R. Greher, P. Laidler, and C. Saulters, "Teaching computational thinking through musical live coding in scratch," in *Proceedings of the 41st ACM technical symposium on Computer science education - SIGCSE '10*. New York, New York, USA: ACM Press, 2010, pp. 351–355.

[9] C. 74, "Max is a visual programming language for media." [Online]. Available: http://cycling74.com/products/max

[10] P. T. Cox, "Visual programming languages," in *Wiley Encyclopedia of Computer Science and Engineering*, B. W. Wah, Ed. Hoboken: John Wiley, 2007, pp. 1–10.

[11] M. Lewis and J. Jacobson, "Game engines," *Communications of the ACM*, vol. 45, no. 1, pp. 27–31, 2002.

[12] A. Kirsch and Y. Chen, "A testbed for adaptive human-robot collaboration," in *KI 2010: Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, R. Dillmann, J. Beyerer, U. Hanebeck, and T. Schultz, Eds. Springer Berlin Heidelberg, 2010, vol. 6359, pp. 58–65. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-16111-7_6

[13] G. Milliez, E. Ferreira, M. Fiore, R. Alami, and F. Lefvre, "Simulating human-robot interactions for dialogue strategy learning," in *Simulation, Modeling, and Programming for Autonomous Robots*, ser. Lecture Notes in Computer Science, D. Brugali, J. Broenink, T. Kroeger, and B. MacDonald, Eds. Springer International Publishing, 2014, vol. 8810, pp. 62–73. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-11900-7_6

[14] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "Usarsim: a robot simulator for research and education," in *Robotics and Automation, 2007 IEEE International Conference on*, April 2007, pp. 1400–1405.

[15] O. Ohashi, E. Ochiai, and Y. Kato, "A remote control method for mobile robots using game engines," in *Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on*, May 2014, pp. 79–84.

[16] "Unity - Manual: GameObject." [Online]. Available: http://docs.unity3d.com/Manual/class-GameObject.html

[17] "InMoov Project," 2014. [Online]. Available: http://www.inmoov.fr/project/

[18] S. Greenberg and B. Buxton, "Usability evaluation considered harmful (some of the time)," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '08. New York, NY, USA: ACM, 2008, pp. 111–120.