Using a Cognitive Architecture to Control the Behaviour of Virtual Robots

Paul R. Smart (ps02v@ecs.soton.ac.uk)
Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ UK

Katia Sycara (katia@cs.cmu.edu)
Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA

Abstract

In order to support the use of cognitive architectures in computer simulation studies involving virtual environments, an integration solution is proposed that enables ACT-R cognitive models to communicate with the Unity game engine. The integration solution is tested using virtual robots that are able to move around a 3D virtual environment and interact with various objects. The robots are equipped with visual, tactile and proprioceptive sensor systems, which yield information about the current sensory state of the robot. This information is communicated to an ACT-R model that subsequently controls the behaviour of the robot by making decisions about motor output. The performance of the model and the integrity of the integration solution are evaluated using a task that requires a virtual robot to retrieve target objects and deposit them in a specified goal location.

Keywords: virtual robotics; situated cognition; virtual environment; game engine; cognitive architecture

Introduction

Cognitive architectures are computational frameworks that have been used to model human cognitive processes in a wide variety of task settings (Thagard, 2012). Typically, cognitive architectures are used to test hypotheses concerning the cognitive mechanisms underlying aspects of task performance; however, they have also been used in a range of contexts where the aim is not so much the emulation of specific human cognitive processes as the more general control of adaptive or intelligent behaviour. One example, here, concerns the use of cognitive architectures to control the behaviour of real-world robotic systems as part of studies into what is known as cognitive robotics (see Kurup & Lebiere, 2012). Although the use of cognitive architectures, in these situations, is not based around the modeling of human cognitive processes, the support that cognitive architectures provide for (e.g.) learning, memory, reasoning, and so on, is often useful in producing psychologically-interesting patterns of behaviour.

The aim of the current work is to combine the use of cognitive architectures with virtual environments in order to study the situated behavior of virtual robots – autonomous mobile agents that are equipped with simulated sensors and motor control systems. Our particular focus of attention concerns the integration of the ACT-R cognitive architecture (Anderson et al., 2004) (which is one of the most widely used cognitive architectures) with the Unity[1] game creation system (which is one of the most widely used game development environments). By integrating these two systems, we hope to show how ACT-R models can be effectively embedded in virtual environments.

[1] http://unity3d.com/

The main features of our work, which serve to distinguish it from previous attempts to integrate ACT-R with virtual environments (e.g., Best & Lebiere, 2006), include the following:

1. Firstly, we aim to create a generic framework for integrating ACT-R with Unity-based virtual environments in order to study a variety of different behaviors. The current work is centered on the use of ACT-R models to control virtual robotic systems in a contrived environment; however, the same framework could also be used to control the behaviour of humanoid characters in more realistic settings. The generality of our approach is also evidenced by the fact that the same approach could be used with both 3D and 2D virtual environments.

2. Secondly, we aim to achieve a tight coupling between ACT-R models and the perceptual and motor processing mechanisms of an environmentally-embedded agent. In particular, we require ACT-R models to process real-time information feeds from simulated sensor systems, and we also require them to generate real-time behaviour using a set of motor control parameters.

3. Thirdly, we aim to make use of a broader array of sensor systems than is typically seen in the case of other work. The current ACT-R models thus make use of visual, tactile and proprioceptive information in order to make decisions regarding motor output.

4. Finally, in the spirit of work that goes under the heading of embodied, situated and extended cognitive science (Clark, 2008; Shapiro, 2011), we aim to approach the creation of behaviour-producing mechanisms from a distributed perspective. In other words, we treat aspects of a robot's body as well as features of the external (in this case, virtual) environment as resources that could potentially constitute part of the mechanistic substrate that realizes intelligent behaviour.

In order to demonstrate progress against these objectives we first describe the effort to construct virtual robotic platforms that are controlled by ACT-R models. The robots are simple wheeled vehicles – referred to as 'virtual vehicles' – that are equipped with simulated sensors and motors. We then present a network-based integration solution that supports bidirectional modes of communication between ACT-R and the Unity game engine. Using this solution, it is possible to run simulations in which ACT-R models control the behaviour of virtual robots in specific task contexts. As a

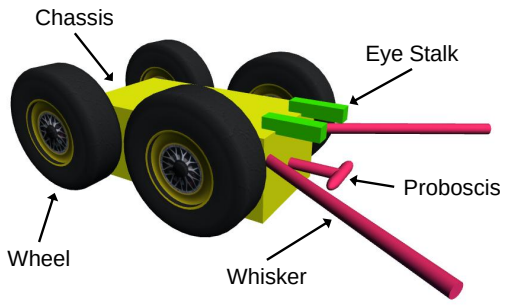

Figure 1: A virtual vehicle.

means of testing the integrity of the ACT-R/Unity integration solution we present the results of a small-scale simulation in which a virtual vehicle is tasked with the collection of target objects within a 3D virtual landscape.

Virtual Vehicles

The virtual vehicles used for the current work are implemented as simple 3D models consisting of a number of primitive geometric shapes. Each vehicle has a simple body plan consisting of a main chassis, to which are attached two eye stalks, four wheels, two whiskers and a proboscis (see Figure 1).

A `Rigidbody` component is attached to the top-level game object in the hierarchy of objects that make up the virtual vehicle's body. This Unity component enables the vehicle to participate in the physics calculations made by Unity's physics engine. The addition of a `Rigidbody` component enables some of the physical properties of a vehicle to be specified, such as its mass and drag. It also enables the vehicle to respond to physical forces and engage in physical interactions with other objects in the environment. A set of `Colliders` are attached to each element of the vehicle's body in order to define the collision geometry of the vehicle. This geometry establishes the 'physical presence' of the vehicle within the virtual environment. In other words, it defines the surface at which collisions with other physical objects in the environment take place. The wheels of the vehicle are associated with a set of special colliders, called `Wheel Colliders`. These are specifically designed for wheeled vehicles within game environments. They feature built-in collision detection, wheel physics, and a slip-based tire friction model.

Sensors

Virtual vehicles come equipped with a variety of virtual sensors. These provide information about the current state of the vehicle's sensory environment, and ACT-R models rely on this information in order to make decisions regarding motor output. The state of the sensors is polled as part of a 'sensor processing cycle' that is periodically executed throughout a simulation.

Vision Vehicles come equipped with two Unity `Cameras` situated at the end of the eye stalks. These function as the vehicle's eyes. By default, the eye cameras point directly ahead (i.e., zero degrees rotation with respect to the Y or 'up' axis) and thus provide overlapping fields of view. This can, however, be modified during the design phase, or at runtime.

As with all `Camera` components, the eye cameras render a specific view of the 3D scene based on the properties of the camera's view frustum. For our purposes, the eye cameras render to a specific type of game asset, known as a `Render Texture`. This is a special type of 2D image asset that can be created and updated at runtime. Once a camera's view of the scene has been rendered to the `Render Texture`, it is possible to use standard 2D graphic processing routines to retrieve information about the red, green and blue (RGB) values of each pixel in the `Render Texture` assets. This provides the raw data for further visual processing. For example, it is possible to compute average luminance levels across the three RGB color channels in order to get an estimate of the average light intensity recorded by each eye camera. It is also possible to deal with individual color channels in order to process color information.

The kind of image processing techniques employed in the context of the current work are relatively simple: they involve the computation of average or maximum luminance levels taken across the entire image, or regions thereof. Each `Render Texture` is divided into 5 vertically-oriented regions that are numbered from 1 to 5 with a value of 1 representing the most medially-situated region and a value of 5 representing the most laterally-situated region. We refer to the `Render Texture` as the vehicle's 'retina' and the subregions as 'retinal patches' (see Figure 2).

Within each sensor processing cycle the image rendered from each eye camera is processed to obtain 1) the average luminance level within each color channel across the entire retina, 2) the average luminance level within each color channel within each retinal patch, 3) the maximum luminance level within each color channel across the entire retina, and 4) the maximum luminance level within each color channel within each retinal patch. This body of visual information (from each eye) is ultimately made available to ACT-R models in order to support the implementation of phototactic behavioural responses.

Proprioception In addition to visual information, virtual vehicles are also able to supply ACT-R with proprioceptive information. This includes the angular orientation of the vehicle's eyes, which can be moved independently during the course of a simulation via the vehicle's motor system. It also includes the current state of the vehicle's proboscis, which can be extended in an effort to dislodge awkwardly placed target objects or to deposit target objects in a goal location. Finally, information about the current steering angle of the vehicles front and reer wheels is made available to ACT-R models during each sensor processing cycle. This keeps ACT-R models informed about the status of ongoing steering opera-

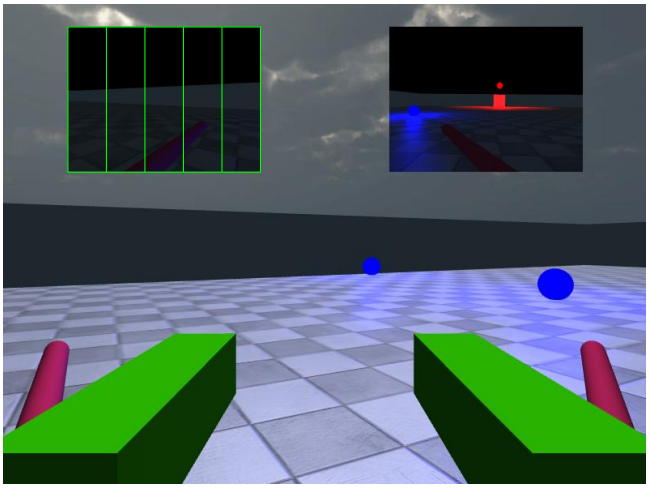

Figure 2: View of a scene from the vehicle's perspective, with additional views of what the vehicle 'sees' through each eye camera (see insets at top right and top left of the image). The image generated by each eye camera is divided into a number of 'retinal patches'. These are indicated by the green grid that is superimposed on the left eye. Data from the retinal patches can be processed to extract spatially localized visual information from the 3D scene.

tions.

Taction A primitive tactile sense is implemented by recording the collisions that take place between parts of the vehicle's body and objects in the environment. Collisions are detected on each sensor processing cycle by both the vehicle's whiskers and the proboscis. Collisions with the vehicle's whiskers provide an important form of feedback regarding obstacles in the environment. For example, if the whiskers report simultaneous contact with another physical object, then the vehicle's subsequent forward motion is likely to be impeded.

In the case of the proboscis, any collisions that occur with the proboscis will result in the colliding object being physically associated with the proboscis until they are actively released by the vehicle. This process is referred to as 'capturing'. Capturing occurs because the proboscis is intended to be sticky and capable of trapping objects that it comes into contact with. The capturing process occurs automatically and is not under the control of an ACT-R model; ACT-R models are, however, made aware of any objects that are 'stuck to' the proboscis.

Motors

The motor system of a virtual vehicle controls its behaviour in the virtual environment. Most of the control values for this system are based on instructions that originate from an ACT-R model, and in the absence of input from an ACT-R model a vehicle is behaviourally quiescent.

The locomotor behaviour of the vehicle is controlled by three parameters: motor torque, steering speed and steering angle. The motor torque parameter specifies the amount of torque that is applied to the wheels of the vehicle. In most cases, the torque is applied to the rear wheels and is positive. This results in the vehicle moving forward in the environment (providing the applied torque is sufficient to overcome any inertial forces acting on the vehicle). The torque can also be inverted to reverse the rotation of the wheels and thus reverse the movement of the vehicle. The steering angle parameter specifies the orientation of the vehicle's front or rear wheels. It can be used to steer the vehicle towards or away from particular targets. The rate at which a vehicle will turn its wheels to align them with the steering angle is determined by the steering speed parameter. This parameter specifies the rate (in degrees per second) at which the vehicle will orient its wheels to the target steering angle.

The motor system also consists of virtual actuators that control the orientation of the left and right eye cameras. This allows a vehicle to visually scan a scene, even when it is stationary.

Finally, vehicles are able to extend and retract their proboscis. This behavior, referred to as 'protrusion', typically occurs in response to specific sensory contingencies and is used by the vehicle to retrieve awkwardly situated target objects or to deposit captured target objects in a goal location.

Environment

For the purposes of this work, we created a simple virtual environment using a set of basic 3D model assets. The movement of the vehicles is constrained to the surface of a flat rectangular-shaped tiled floor plane that is textured with a blue-tinted material. The plane is enclosed on all sides with vertical walls; however, a number of small openings are situated along the walls. These openings constitute 'goal locations'. During the course of a simulation, the openings are illuminated with red glowing spheres, and the objective of the vehicle is to deposit a number of target objects through these openings. The target objects, in this case, are blue glowing spheres that are randomly distributed in the environment. The task is deemed to be complete when all the blue spheres have been removed from the enclosed space and deposited in the goal locations. Both red and blue glowing spheres are implemented using `Point Light` components attached to spheroid meshes. This simulates the appearance of translucent glass spheres that emit light of a specific color in all directions from a single point source.

The main source of illumination in the scene is a `Directional Light` component. This lighting source is only intended for observation purposes, and it is not included in the rendering operations of the vehicle's eye cameras (the light is disabled prior to the rendering process and then re-enabled once the rendering is complete).

There is also a low level of ambient lighting in the scene. This ambient lighting is, unlike the directional lighting, included in the eye camera rendering process, and it thus adds to

the actual level of scene illumination that the vehicle responds to. This is important in terms of establishing a minimum level of locomotor activity. For example, when vehicles are attempting to retrieve the blue spheres, the blue ambient light emanating from the surface of the floor plane produces sufficient photic stimulation to yield a base level of 'exploratory behaviour'. This is the case even if no blue spheres are visible within the vehicle's field of view.

JSON Network Interface

In order for ACT-R models to control the behaviour of vehicles within the virtual environment, there needs to be a bidirectional exchange of information between ACT-R and the Unity game engine. Our approach to this integration challenge is based on the solution outlined by Hope et al (2014). Hope et al (2014) advocate the use of a network-based approach to integration in which the external system (Unity, in our case) plays the role of a server and individual ACT-R models play the role of clients. Communication between the two systems is then established via a series of client-server interactions mediated by TCP/IP socket connections. The messages exchanged by the two systems are formatted using the JavaScript Object Notation (JSON) data interchange format – a format that is commonly used for the transmission of data objects in the context of Web-based communications. These messages typically contain the name of specific functions to run in the context of the client or server environment, along with any relevant data.

Hope et al's (2014) solution is encapsulated in a custom ACT-R module – the `json-network-interface` module. This can be installed alongside the core modules that form part of the default ACT-R architecture (see Anderson et al., 2004). Each `json-network-interface` module implements the functionality to establish connections and interact with an external environment. In addition, the module enables Lisp functions to be registered as Lisp events that can be triggered by commands sent from the external environment.

The parameters of the `json-network-interface` module include the IP address of the external environment as well as the TCP port that the external environment is listening on. This obviously supports the ability of the ACT-R model to connect to the environment at the beginning of a simulation. In addition, the commands that are posted to the external environment include the name of the ACT-R model that posted the command. This enables the external environment to identify the specific entity that is being controlled by an ACT-R model.

ACT-R Unity Interface

The solution outlined by Hope et al (2014) provides the basis for ACT-R to be integrated with a variety of external environments. However, additional effort needs to be undertaken in order to enable specific environments to inter-operate with ACT-R. In order to enable ACT-R to be integrated with the Unity game engine, we have developed a framework called the ACT-R Unity Interface (ACT-R UI). The ACT-R UI consists of a number of components, all of which are implemented as Unity-compatible C# scripts:

- **ACTRNetworkInterface:** The main function of the ACTRNetworkInterface component is to handle connection requests from ACT-R models. It also implements the functionality to post messages to ACT-R clients on the network. The component relies on the native support that Unity provides for .NET socket connections.
- **ACTRCommand:** The ACTRCommand class is the base class for all the commands that are recognized by ACT-R and Unity in the context of a specific simulation. Each ACTRCommand has properties that hold data relevant to the execution of the command in the context of the target environment (either Unity or ACT-R). Subclasses of the ACTRCommand class can be created to extend the range of messages that ACT-R and Unity are able to process.
- **ACTRMessageInterface:** This is a component that inherits from the Unity `MonoBehaviour` class. Its primary function is to engage in the preliminary processing of ACT-R messages. It processes the raw JSON content of the message and attempts to create appropriate instances of the ACTRCommand class based on the message content. Once created, these commands are posted to the relevant ACTRAgent component (see below) based on the name of the originating ACT-R model. The ACTRMessageInterface component contains a user-editable field (visible in Unity's Inspector pane) that specifies the port number that ACT-R clients should use to connect to Unity.
- **ACTRAgent:** This `MonoBehaviour` component represents the entity in the virtual environment that is controlled by a particular ACT-R model. It is typically attached to a game object representing a non-player character, such as a virtual vehicle or humanoid avatar. The main function of the component is to engage in sensor processing and implement the motor instructions received from ACT-R.

In addition, to these components, the ACT-R UI relies on the ability to serialize and deserialize JSON-formatted messages. In the case of the current work, this functionality is provided by the Json.NET framework[2].

ACT-R Extensions: The Vehicle Module

ACT-R has a modular structure in which different modules are intended to implement specific cognitive functions. Each of these modules is associated with one or more buffers that process requests relating to the function performed by the module. By default, ACT-R has eight modules; however, new modules (and buffers) can be added as needed in order to extend ACT-R's functionality in specific areas.

In the context of the current work, a new ACT-R module was implemented to support the processing of sensor

[2] `http://www.newtonsoft.com/json`

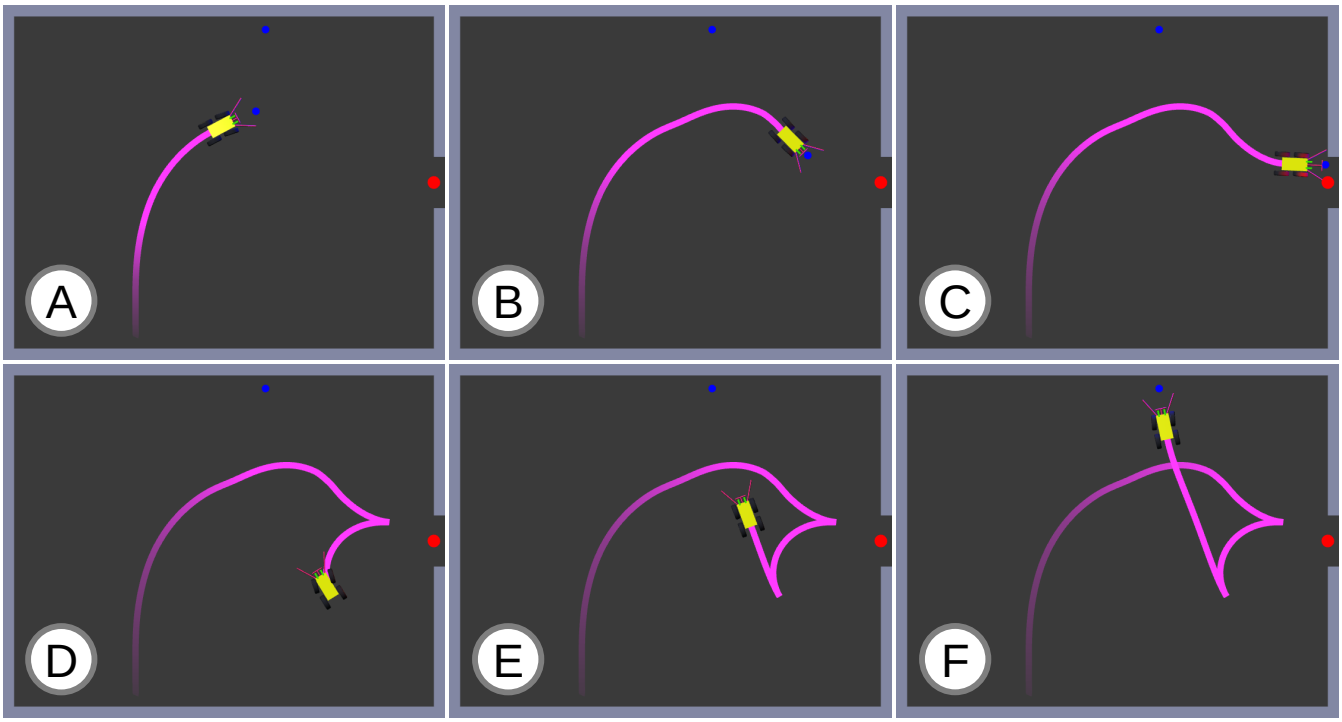

Figure 3: Trajectory of a vehicle as it first approaches a target object (A), manoeuvres the target towards a goal location (B), and then deposits the target at the goal location (C). Following target deposition, the vehicle engages in a reversing behaviour (D) and then begins to move in the direction of a second target (E and F). The progress of the vehicle is indicated by the magenta trail.

information and the generation of motor commands. This module is referred to as the `vehicle` module. It has two buffers: `vehicle-sensor` and `vehicle-motor`. The `vehicle-sensor` buffer contains information about the current sensory state of the vehicle within the virtual environment (if no sensory information is available, the buffer is empty). The content of the buffer is populated automatically based on the messages received via the ACT-R UI. When Unity posts a message containing updated sensory information, the message triggers the execution of a Lisp function that creates an ACT-R chunk containing all the sensor information contained in the message. This chunk is subsequently placed in the `vehicle-sensor` buffer.

The `vehicle-motor` buffer processes requests relating to the behaviour of the vehicle. It accepts information (in the form of ACT-R chunks) about the desired state of a vehicle's motor system. When a chunk is placed into the buffer (as a result of rule execution), the `vehicle` module processes the chunk to extract the relevant motor information. It then dispatches a JSON-formatted message containing the motor information to the `json-network-interface` module for transmission to Unity. Once this message is received by Unity, it is handled by the ACT-R UI: the relevant vehicle is identified based on the name of the ACT-R model that posted the message, and the message is then sent to the relevant ACTRAgent component for further processing. During the course of each update cycle of the game engine, the ACTRAgent will attempt to implement the motor instructions it receives from ACT-R.

Integration Testing

In order to test the integrity of the ACT-R/Unity integration solution, a simple ACT-R model was developed to control the behaviour of a virtual vehicle. As described above, the vehicle was situated in a simple 3D environment containing a number of blue glowing spheres. The objective of the vehicle was to retrieve the spheres and deposit them in one or more goal locations (indicated by a red light). The ACT-R model developed to realize the target behaviour consisted of a total of 30 productions. These controlled all aspects of the vehicle's behaviour. The productions can be organized into a number of categories based on the kind of behaviour they control. For example, target retrieval rules control the initial retrieval of blue spheres within the environment; target removal rules, in contrast, control the transfer of retrieved blue spheres to goal locations. Both of these behaviors are forms of phototactic response that are governed by the photic stimulation provided by either blue spheres (target retrieval behaviors) or red spheres (target removal behaviors). Other types of rules rely on tactile information. These include rules controlling obstacle avoidance behaviors (obstacle avoidance rules) and rules controlling the deposition of targets in the

goal location (target deposition rules).

In the testing phase, a total of 5 simulations were run with environments containing 3 randomly placed targets (i.e., blue spheres) and 1 goal location (i.e., a single red-illuminated opening). The sensor processing cycle was set to run at a rate of once per second. On average it took 295 seconds (standard deviation: 87.78) for a single vehicle to remove all the target objects from the environment. In none of the simulations did the vehicle fail to collect all of the target objects.

Figure 3 shows the behaviour of a vehicle as it first approaches a target, manoeuvres the target towards the goal location and finally deposits the target in the goal location. The magenta line, in this case, is generated by a Unity `Trail Renderer` component that is situated at the rear of the vehicle. It provides a visual trace of the vehicle's movements across time. A video showing the behaviour of a vehicle during the course of one particular simulation is available for viewing from the YouTube website[3].

Conclusion

The main aim of the current paper is to describe the approach taken with respect to the integration of the ACT-R cognitive architecture with the Unity game engine. The availability of this solution lays the foundation for a number of strands of work that seek to understand the mechanisms underlying the emergence of psychologically-interesting patterns of behaviour in environmentally-embedded agents. In particular, we suggest that this work can be seen as part of a larger effort to undertake studies into what might be called computational embodied, situated or extended cognition. The aim here is to understand the way in which a variety of forces and factors that are distributed across internal control systems, bodily structures and the external environment work together to yield intelligent behavior. With respect to the current work on virtual robotic systems, we can imagine running a variety of simulation experiments that systematically explore the effect of a range of factors on aspects of task performance (e.g., the time taken to remove a specified number of target objects from the environment). Such experiments could include the use of different environments (e.g., environments with different spatial layouts and sensory stimuli), bodily designs (e.g., bodies with different morphologies and sensor placements), sensor functionalities (e.g., visual systems with more sophisticated image processing capabilities or tactile sensors with more refined spatial resolutions) and behavior control systems (e.g., ACT-R models with more complex productions or different module parameter settings). It is also possible to imagine how the current work could be extended by exploiting more of ACT-R's built-in cognitive capabilities, such as the capacity for learning and mnemonic processing. This raises the possibility of introducing plasticity in a vehicle's behaviour, enabling vehicles to adapt their behaviour based on previous experience.

It is also important to recognize that nothing prevents the current integration solution being applied to situations that seek to model human cognitive processes. In this case, the use of the ACT-R architecture can be used to create cognitively-plausible models based on the prior observation of human performance in some task setting. This can then be tested in the context of a virtual environment that mimics at least some of the perceptual and motor features of the original task. Furthermore, by interfacing ACT-R models to humanoid virtual characters, we can seek to control character behaviour in a way that is sensitive to the sensory environment of the virtual world. This potentially serves as the basis for computer simulations that target the kinds of phenomena commonly encountered in the context of situated, embodied and extended cognitive science research (Clark, 2008; Shapiro, 2011).

[3] `http://youtu.be/pPZq2MgsOQE`

Acknowledgments

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*(4), 1036-1060.

Best, B. J., & Lebiere, C. (2006). Cognitive agents interacting in real and virtual worlds. In R. Sun (Ed.), *Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Interaction.* New York, New York, USA: Cambridge University Press.

Clark, A. (2008). *Supersizing the Mind: Embodiment, Action, and Cognitive Extension*. New York, New York, USA: Oxford University Press.

Hope, R. M., Schoelles, M. J., & Gray, W. D. (2014). Simplifying the interaction between cognitive models and task environments with the JSON network interface. *Behavior Research Methods*, *46*(4), 1007-1012.

Kurup, U., & Lebiere, C. (2012). What can cognitive architectures do for robotics? *Biologically Inspired Cognitive Architectures*, *2*, 88-99.

Shapiro, L. A. (2011). *Embodied Cognition*. Abingdon, Oxfordshire, UK: Routledge.

Thagard, P. (2012). Cognitive architectures. In K. Frankish & W. M. Ramsey (Eds.), *The Cambridge Handbook of Cognitive Science* (p. 50-70). Cambridge, UK: Cambridge University Press.