International Journal on Interactive Design and Manufacturing (IJIDeM) (2019) 13:121–136
https://doi.org/10.1007/s12008-018-0470-z

ORIGINAL PAPER

Machine tool setup instructions in the smart factory using augmented reality: a system construction perspective

Evangelos Tzimas[1] · **George-Christopher Vosniakos**[1] · **Elias Matsas**[1]

Received: 21 January 2018 / Accepted: 14 March 2018 / Published online: 22 March 2018

Abstract
The concept of smart factory includes collection and distribution of information in real time to help human operators in their work. Machine setup instructions are notable examples of such information. This paper advocates the use of augmented reality (AR) to present such information addressing in an intuitive manner the demonstration of setup tasks on the variety of machine tools that are available in a factory and for the multitude of parts that are produced. Turning and machining centres are used as typical machine tools, for which setup instruction applications were created on a widely used computer game development platform enhanced by standard AR functionality based on target markers present on the scene. This AR demonstrator has been primarily created as a proof of concept that highlights the feasibility of such low-cost, efficient and user friendly AR applications of industrial guidance and training. The emphasis of the work is on technical aspects of AR application development, which is lacking in literature. In particular, interfacing of distance sensors that impart intelligence, scenario structuring in terms of states and state threads, as well as techniques ensuring expansibility, flexibility and ease of authoring of such applications are demonstrated.

Keywords Augmented reality · Manufacturing execution · Machine tool · Setup · Smart factory · Industry 4.0

1 Introduction

Augmented Reality (AR) refers to mapping in real time the real environment, elements of which are augmented by presentation media created by computer. Thus, in AR user cognition is enhanced, by contrast to Virtual Reality (VR), in which the real environment is totally replaced by a virtual one [1]. Typically, using computer-vision techniques related to object recognition, the information directed to the user becomes interactive and digitally manipulable. AR technology is dynamically expanding in a variety of application domains. For instance, in archaeology, buildings, landscapes and even human characters from the past can be revoked, in architecture, improved visualisation of future projects can be achieved and in medicine, ultrasound scanning results can be visualised by 3D models superimposed on the patient's body [2]. Despite such well-established applications, AR in manufacturing practice has not been as widespread [3]. To some extent, this is due to lack of lightweight, non-intrusive, yet practical systems that can be seamlessly integrated in everyday workflows, a pitfall which is gradually being remedied [4]. In addition, there have been generally mediocre results in relevant pilot projects. For instance, Boeing's pioneering work in the 1990's regarding assembly of different wire harnesses on various aircraft was not succesful, although, in a noteworthy comeback, remarkable results employing AR glasses have been recently reported [5].

In the manufacturing domain, information collection from the factory and interactive presentation in real time to personnel involved through AR techniques has been suggested [6], which in some cases reach out to context-sensitive knowledge sharing [7]. By far the domains that attracted most AR applications are assembly and maintenance and related inspection [3]. Nevertheless, augmentation of process information focusing on best practice and on-line guidance has also been supported, e.g. for metal welding [8].

✉ George-Christopher Vosniakos
vosniak@central.ntua.gr

Evangelos Tzimas
mc06024@central.ntua.gr

Elias Matsas
imatsas@central.ntua.gr

[1] School of Mechanical Engineering, National Technical University of Athens, HeroonPolytehneiou 9, 15780 Athens, Greece

AR hardware includes visualisation devices, sensors connecting the AR application to the real environment and software processing sensor output and providing visualisation input [4]. Smartphones and tablets are popular hardware platforms because they are equipped with cameras, sensors (gyroscope, accelerometers, GPS etc.), with an operating system enabling application addition and execution whilst possessing energy autonomy. Head Mounted Displays (HMD) are also popular in AR for visualisation purposes allowing the user to free his/her hands [9]; AR goggles, although controversial so far, in terms of commercial success, have been reportedly used in pilot projects in industry [5]. However, plain computer monitors or even projectors [10] are sometimes used as visualisation devices, too, especially when mobility is not required, in which case the real environment is captured by camera. AR software involves computer vision to perform image recognition of some sort. Normally, some points or targets of interest are first recognised to serve as positioning and scaling reference and are tracked during the execution of the application [1]. Applications differ depending on the number and quality of the sensors required, tracking and virtual object positioning and orientation accuracy, speed of response of scene and object recognition and overall realism of the result [4].

General AR issues under research concern registration, tracking, especially markerless [11], and 3D workspace capture [12]. Beside those, the central issue is presentation of complex information to the human operator, in a most intuitive manner, alleviating inherent cognitive load [13] whilst at the same time allowing for reconfigurability of the overall task that is supported by instructions [14]. This points further to the overall structuring of such assistive systems [15] and, in particular, to knowledge representation [12] and formal structuring of instructions, e.g. semantics-based, to accommodate variability of products and processes [16], or even intelligent structuring varying with context [17,18]. It also points to the type and configuration of the user interface for conveying the instructions and, more generally, to the interaction between human and machine in a cyber-physical world. The user interface is mostly visual, relying on overlaying graphics, patterns and text displayed on the real image in the form of virtual sticky notes [19]. Moreover, context sensitivity is an interesting and desirable property of such interfaces, see for instance a pertinent implementation by Gorecky et al. in a prototype smart assembly station [20].

Typical examples of assembly instructions through AR have been reported in literature: manual assembly of business card holders [21], picking, positioning, screwing, tightening, press fitting simple components of an artificial product [22], control of an assembly line in terms of information on products being processed and processes being carried out [23], identification of spot welds in assembly inspection tasks [24] and manual assembly of prototype vehicles [25]. Examples of AR supported instructions of maintenance tasks include motorbike engine maintenance exploiting marker-based recognition [10], removal and installation of pitch trimmers in aircraft landing gear [18], maintenance of gasoline engines [26] and of flapper pump valves [27]. Design rules for laying out such systems are being researched, too [28]. However, generating annotations for AR support of large and diverse procedural tasks may be impractical, hence automatic authoring methods are also under investigation [29,30].

As a means to supporting assembly, maintenance, inspection or machine preparation instructions, AR has the potential to reduce users' body movements, cognitive effort and attention switching [31], and therefore it could lead to faster execution times [10]; in fact, it has been reported to be superior compared to instructions given in traditional handbooks or manuals [10], but also on video [32]. Note that such assistive systems can easily be used in training tasks, too, e.g. [33].

Apart from the potential user friendliness, immediacy and efficiency in conveying instructions by AR, this is also most interesting due to its potential functioning within a smart factory or production digitalisation context, commonly known as Industry 4.0. The pertinent palette of tools and methods encompasses VR/AR, cyberphysical systems, as well as data storage and retrieval for 'big data' [34]. In spite of the rich literature on providing AR-based instructions in a manufacturing execution context, few have put it in this larger context, see for instance reference [6].

In this work, instructions to machine tool operators concerning the setting up steps for various parts to be processed are focused on. Setup instructions do not differ much from assembly or maintenance instructions, except that they are typically less complex. Focus is on the technical aspects of implementing an AR-based assistive system for such tasks and not on evaluating it. In addition, the need for data management and communication infrastructure necessary for supporting such a system in an industrial environment is recognised but the pertinent technical solution is left aside for the time being.

This paper is structured as follows. In Sect. 2 the problem dealt with is described and the solution advocated is outlined. In Sect. 3 the functionality of the AR development platform is presented in general terms. Section 4 describes the application developed to provide setup instructions for turning centers, whilst Sect. 5 presents its counterpart for machining centers. Note that in contrast to other similar publications, sufficient technical detail is given for the reader to be able to appreciate the nature of the application and its development. Section 6 discusses the issues that arose during application development and use, whilst Sect. 7 summarises the pertinent conclusions.

2 The problem and the solution concept

Machining processes require clamping of the part to be processed on suitable devices depending on the machine tool. For instance, in turning centres the part is usually clamped on one end in three or four-jaw chucks and, if required, held at the other end on the tailstock to avoid vibration and excess bending displacement. Simple cylindrical workpieces may be held between centres, long workpieces may be held through the spindle, sometimes in conjunction with automatic bar feeders. Furthermore, depending on the complexity of the shape to be turned, two or more setups may be necessary, some of them involving gripping the part at an intermediate diameter and not at its end, see Fig. 1a. In machining centres, where part shapes are much more complex compared to turning, holding devices may be: vice, chucks, jigs build on-purpose. Most often, the table of the machine is used as positioning surface and modular clamping systems consisting of positioning and clamping modules are employed. Alternatively, pallets may replace the machine table to allow for external setups, the pallet itself being clamped on the machine table in an easy and standard way. Setup on machining centres may need to be changed several times for a complex part, see Fig. 1b.

Setup planning is part of the overall process planning in machining processes. The plan is usually communicated to the machine operator using CAD drawings or models, but these usually restrict themselves to depicting the orientation of the part and the points where positioning and clamping devices should be applied without depicting the devices themselves, let alone the procedure to use them. This is partly due to the fact that those devices are not known beforehand to the process planner, since their selection is a matter left to the machine operator.

When the same part is machined repeatedly in large batches, the first few times of setting it up on the machine tools will certainly take longer but setup duration will eventually reduce with repetition. However, in job shops where parts are produced in small batches or even in one-off fashion set up time may become comparable to actual processing time of the part due to the learning effect.

This paper advocates that setup guidance is very effectively provided by an AR system because it takes into account in real time the actual machine tool on which manufacturing process will take place as well as the variety of holding devices available. The machine tools are captured from real world in real time, whilst the holding devices are 3D virtual models that are superimposed to the real machines in order to achieve as realistic setup instructions as possible. In addition, supplementary textual and iconic information is advocated to ensure more accurate procedure description regarding setup implementation. Correct positioning of the virtual with respect to the real objects is ensured through computer vision, which is more or less standard in AR applications, but accuracy is enhanced through suitable sensors. The variety of parts, setup plans and holding devices is dealt with through a database where their particulars including 3D shape are registered. The above mentioned ingredients of the approach make for its universality irrespective of the particular machines to be setup and the part, jigs and fixtures involved. The system concept is shown in Fig. 2.

3 AR application development platform

The software platform on which the setup instructing system was developed consists of Unity3D™, together with the AR development plug-in (Software Development Kit—SDK) Vuforia™. Virtual objects were created using Solidworks™ and 3DSMax™ systems. In terms of hardware, Arduino IDE, Arduino Uno microprocessor were employed as programmed in C language. In addition, standard web cameras were necessary irrespective of the particular application, whereas ultrasonic sensors were indicatively used for distance measurement.

3.1 Unity 3D

This is the most popular game development engine in industry, supporting more than 15 platforms from PCs to smart phones and the internet since its emergence in 2005. It comprises extensive documentation and a large and active user community, which provides substantial help with technical matters. Unity can import virtual objects/models constructed on other software in .fbx and .obj formats. There are also plug-ins that support addition and programming of various peripheral devices. Of the different possible IDEs (Integrated Development Environments), the default was used, named MonoDevelop, whereas of the different programing languages, C# was used enabling software development in .Net Framework.

Unity3D supports creation of *Projects*, each comprising a number of *scene*s. Each scene contains a number of *GameObjects* which, in turn comprise of *Components. Components are instances of classes giving each GameObject desirable behaviour, when they are attached to it.* There is a large variety of ready-made *Components* as well as the possibility for the developer to define custom components to be attached to a Game Object. Those are known as script components. All game objects of a scene are displayed in the hierarchy window of the user interface, see Fig. 3, structured in a hierarchical parent-child tree. This depiction is very important as children game objects of the same tree have a relationship of inheritance with the parent object, so that features such as visibility and position can be shared among them. In the central window (scene view) the user can navigate in the

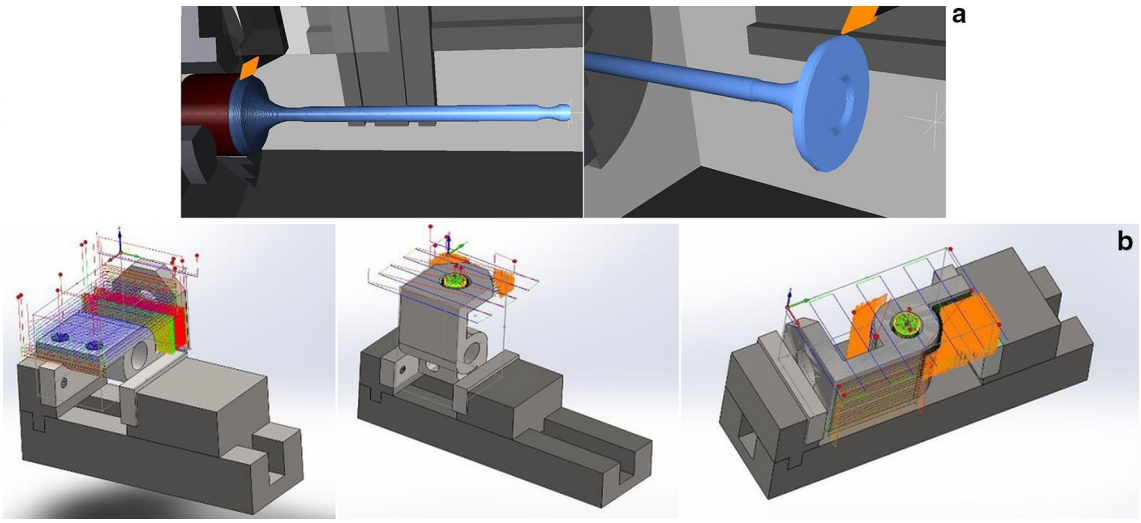

Fig. 1 Part setup examples **a** turning in two setups on a chuck **b** machining in three setups on a vice

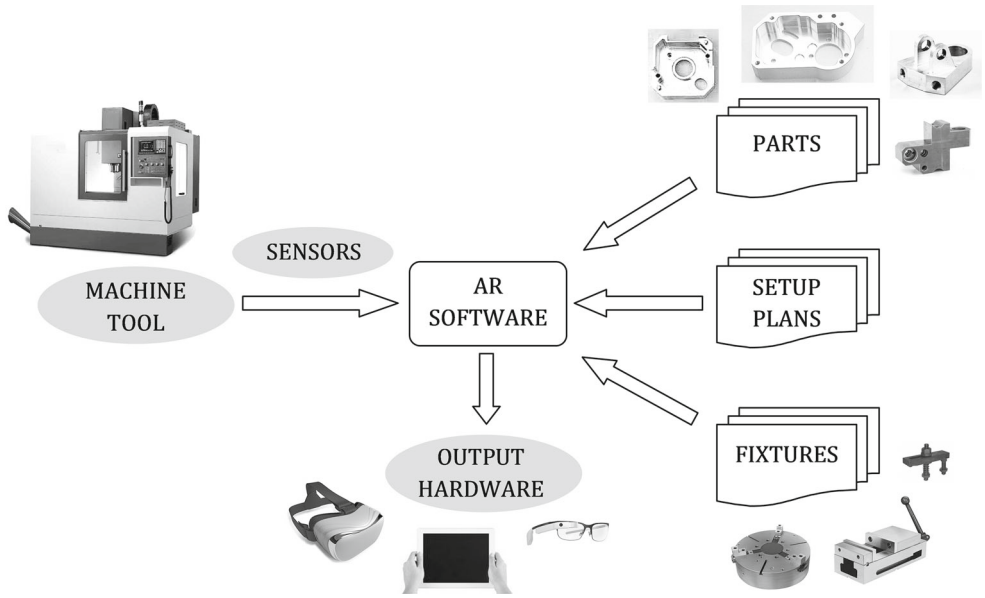

Fig. 2 Setup instructing system concept

virtual scene and also modify objects through the tools line. The inspector window displays all Components attached to a selected GameObject along with each modifiable parameter. Those are the public variables of the Component class. Project window, at the bottom part of the interface contains all files and folders comprising the project.

Basic game object shapes provided by Unity3d are: cube, sphere, capsule, cylinder, and plane. Objects with more complex geometry developed for the applications presented, are created in SolidWorks^{TM}, and stored as an .iges format file. Afterwards they are processed by software such as 3dsMax or Rhino3D, to parametrise each object's scale, mesh, orientation and position with respect to global coordinate axes and stored as .fbx files. The respective mesh is imported by Unity and appropriate materials and textures are added.

Light-related game objects are: *Directional, Point, Spot, and Area Light*. An important game object is the *Camera*, providing the user's field of view in the scene. There are also game objects without components, which group together other game objects into composite ones with inheritance relations.

Some components of standard use are: *Transform* which defines the position, orientation and size of the object, *Mesh Filter*, which links the object with a mesh denoting its shape and surfaces, *BoxCollider*, which identifies neighbouring

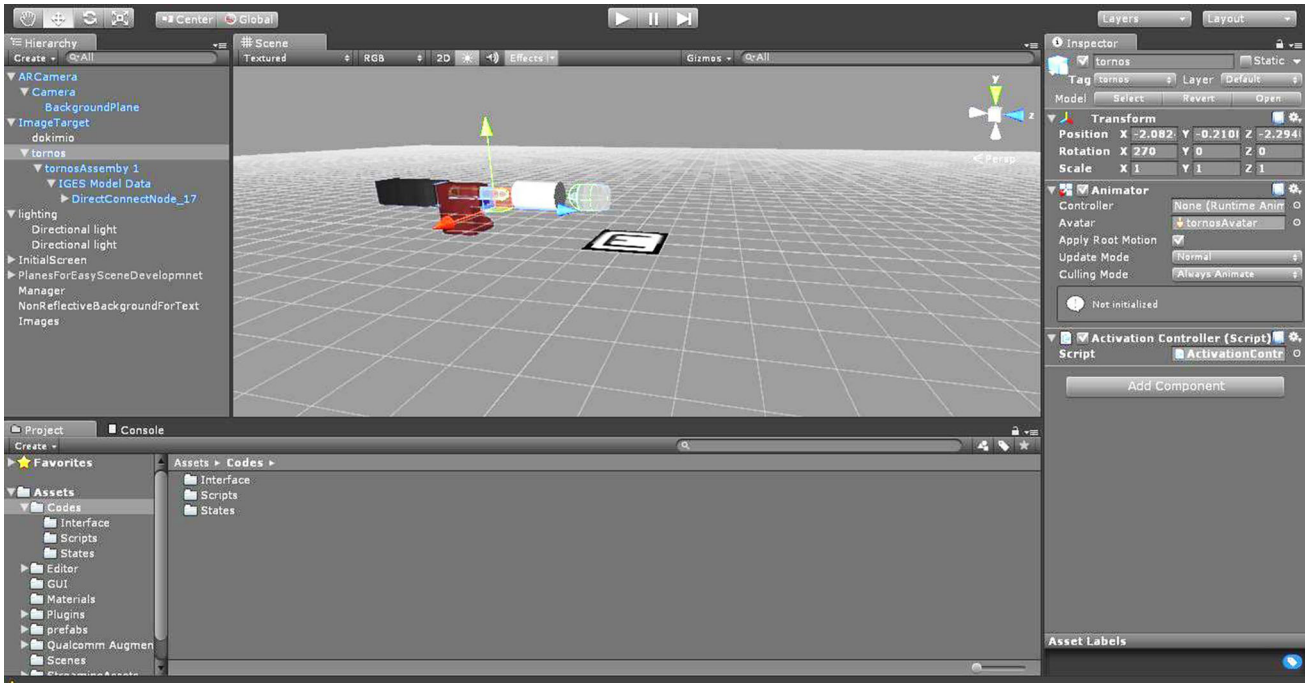

Fig. 3 Unity user interface

objects within a control cube surrounding the object and *Mesh Renderer*, which is responsible for visualising the object in a realistic way. There are also additional Components such as *Rigid Body*, which enables attachment of mechanical properties, calculation of acceleration as a result of forces etc.

Interactive behaviour of objects is also created by script components. Those can be children classes of the *MonoBehaviour base* class of Unity, thus inheriting methods such as *Start()* and *Awake()*used in component initialisation, *Update()* executed in each frame, FixedUpdate() executed in-synchronisation with the physics engine, OnGUI(). Another category of inherited methods is event handling functions, which anticipate a specific type of event in order to be executed, e.g. *OnMouseDown(), On TriggerEnter(), OnCollisionEnter()*etc. In addition, Coroutines (methods that can be interrupted before their normal termination and resumed from the interruption point) can be defined and executed. Script components can also gain access to other components' public methods and variables so that they can be manipulated when called during execution of the application. Unity3D's classes are organised in UnityEngine namespace.

Everything that is necessary in order to create a project in Unity3D is termed an *Asset*. *Assets* can be imported from external sources or from other projects, which makes for fast application development. In the same sense, 3D objects designed in CAD software can be imported in .fbx or .obj format.

3.2 Sensor interfacing for intelligent augmented reality

In advanced application of AR, interaction with the environment or just information that is not based on vision and image recognition may be beneficial or even necessary. For instance, thermometers, pyrometers or thermocouples may be needed to measure machine temperature and overlay this information on the scene captured via a normal camera of a tablet and projected on its screen. The signal from such sensors is typically transmitted or sent to a microcontroller, such as ATmega328P used in the Arduino UNO platform, where it is suitably processed before being transfered to the USB port of the PC on which Unity3D is being executed.

3.2.1 Interface hardware

Arduino Uno™ is an open source board with a microcontroller and accompanying electronics. It is used for developing digital devices and interactive objects that can capture information and control objects in physical space. There are both analog and digital input pins, but outputs may only be digital (0/5 V) or pulse width modulated (PWM). At least one serial port is available, connected to the usb port and the digital pins 0 (RX) and 1 (TX) thus data transfer is available according to UART protocol. There are additional communication capabilities too, with pertinent hardware such as wi-fi shields, Ethernet shields etc. A program for any Arduino™ platform can be created in the Arduino IDE

(Integrated Development Environment) using C/C++ programming language with the use of relevant libraries. The program can be uploaded to the board via usb cable.

Arduino's most basic functions are briefly presented in this paragraph. The *void setup()function* is executed once at program start (initialisation) to define pins to be used as well as serial communication parameters etc. *Void loop ()* function contains the main part of the program which is executed as long as there is power supplied to the Arduino™ board. Analog input signals are converted by a 10bit analog to digital converter to an integer from 0 to 1023 and are accessed using the analogRead() function, whilst analog output is simulated by a PWM signal, for which the preferred duty cycle can be set using the analogWrite function(). Serial and digital read/write is also supported as well as functions recording time elapsed from program start and delay in program execution.

3.2.2 Interface software

In order to interface a sensor to Unity3D™ serial port is used and API Compatibility Level is assigned the value .NET 2.0 so as to include System.IO. Ports namespace (it contains classes for controlling serial ports) in the creation of the script component that is going to listen to the serial port the Arduino board is connected to, and depending on the value captured to act accordingly. A delay larger than 50msec is applied after each new value capture by the microcontroller to safely allow for synchronisation with Unity3D™. A similar constraint applies to the *Update()*method that is executed once per frame.

4 Setup instructions for a turning centre

The application that was developed aims at supporting the turning center operator in clamping rotational parts. The main user requirements of the application were: (a) clarity of instructions (b) extensibility (c) modifiability. In addition, the application was considered as a testing platform for the software, camera, sensors and visualisation equipment used, in order to improve on them in the second application.

The camera employed was a conventional low cost one, i.e. *LogitechHDwebcamC525*.

4.1 Target recognition

A predefined marker was employed as a target for recognition and tracking. This constitutes, in fact, a position reference in 3D space and a dimension reference for all virtual objects to be input, see Fig. 4a. Target recognition is performed using Vuforia SDK, which creates two 'prefabs', namely *ARCamera* and *ImageTarget*, whilst the default gameobject *MainCamera* is withdrawn.

Component *ImageTargetBehaviour* dictates that the particular target is sought in every frame provided by the web camera when the application is running. Furthermore, the game objects which are overlaid on the physical space view captured by the camera are defined as children of *ImageTarget*, so that their positioning, motion direction and dimensions are executed with respect to the corresponding elements (fields of the *Transform* component) of *ImageTarget*.

ARCamera is parent to the *Camera* object, which in turn is parent to the *Backplane* object. The latter functions as projection surface of the image captured from the web camera. *Camera* object is the main object that is responsible for the view provided to the user during execution of the application. The parent of both objects, namely *ARCamera*, is mapped to the Licence Key via its component *QCARBehaviour*. Its role is to recognize and track the target image and to modify its coordinates in real time, so that the relative position of the web camera and the real target in physical space is transferred in the same, proportionate, manner to the scene between object *ImageTarget* and *Camera*. This can be achieved through the hierarchical relation of objects *ARCamera*, *Camera* and *BackgroundPlane* that dictates transfer of the variation of the parameters of the *Transform* component of the object *ARCamera* to its children objects, whilst keeping the relative distances of the three objects unchanged. Classes provided by Vuforia SDK are extensively documented in [35].

4.2 Length measurement sensor

The part to be setup has a specific length which the system has to be informed of. Asking the user for such information would be too intrusive, therefore a distance sensor, together with a microcontroller undertake automation of this task, see Fig. 4b.

The sensor, Cytron Technologies model HC—SR04, has an active range of 2–400 cm and an accuracy of 3 mm. The output signal is configured in the control circuit whilst a led is connected between Vcc and GND pins to indicate activation of the sensor. Measurement onset is triggered by a signal 5V fed to pin Trig, see Fig. 4c, for at least 10 μs, thereby activating the process of sending 8 ultrasonic pulses at 40kHz and pin Echo goes high. When the reflected pulse is received, pin Echo goes low. By measuring the duration of pin Echo's pulse the time that the signal has travelled can be obtained. Distance is then calculated as: $sound\ velocity * time/2 \approx time(\mu s)/58$

This process is programmed in the microcontroller's language, so that the signal from the sensor is led to the serial port of the PC and ultimately to the application.

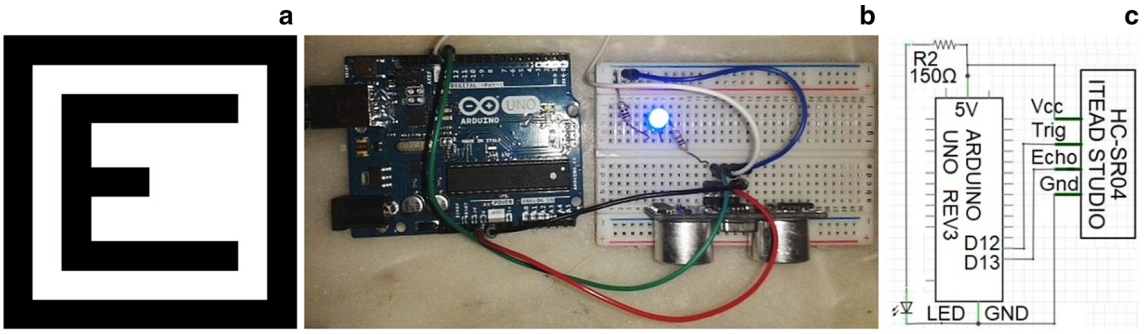

Fig. 4 **a** Recognition and tracking marker **b** ultrasonic sensor and microcontroller **c** sensor connection

To this aim, Component 'ArduinoSensor' was created in Unity, in order to acquire the digital information through serial port COM3 at 9600 baud rate. It is attached to object 'BackgroundPlane', and sets serial port COM3 to be monitored by the application. Using command ReadLine() it reads the string sent by the microcontroller and converts it to an integer, assigned to globally accessible variable int height which is actually the length of the part being set up in cm.

4.3 Game objects

All game objects are present in the scene from the beginning. The behaviour of some of them is dynamically modified by activating or deactivating pertinent components attached to them. The object InitialScreen has script component 'GUIManager' that changes the value of variable 'targetRecognized' to 'true' when recognition has been achieved and enables effective start of the application. Assembly object is parent of the objects 'Part' and 'Lathe' and child of the ImageTarget object and possesses only the Transform component. Thus, its purpose is to allow changes in positioning of its children objects but keeping their relative distance.

Object 'Part' is the virtual cylinder that simulates the respective real part to be setup, see Fig. 5. 'ScalePart' component, see Fig. 5, posseses public methods *void GetSensorValue()* and *void ManuallySetHeight(int height)*. Upon being called, these methods change the height of the cylinder (y-coordinate of the localScale vector of the part). The first method obtains the value of the height as measured by the microcontroller, whereas the second one obtains the same value from its argument, if this is desired. Note that when the height value changes, the respective dimension of the part's collider is updated.

'MotionDokimio' component possesses the method v*oid MoveFromAnotherScript()*, which upon being called starts upCoroutine*MoveToSpindle()*, resulting in the part being moved to the right with constant speed until it reaches the spindle'sCollider.

The clamping subsystem, i.e. chuck and tailstock, of the lathe is simulated by the object 'Lathe', see Fig. 6. Object 'Lathe' possesses method *void ChangeVisibilityOfLathe (),*toggling visibility states. Its children objects 'Chuck-DC_Shell' κ α ι 'Tailstock-DC_Shell' have component CapsuleCollider attached, in order to identify through method *OnTriggerEnter* contact with 'Part'. 'Tailstock-DC_Shell' possesses method *void MoveTailstockFromExternalScript()*which moves the tailstock towards the part until it reached the latter's Collider, by analogy to method *void MoveFromAnotherScript()* described above.

Object 'Manager' is of special importance since it is responsible for the interaction with the Graphical User Interface, as well as the management of the objects in the scene by calling the methods available in their components. The component script that gives Manager this behaviour is MainManager. MainManager's variable activeState accepts instances of classes implementing interface 'IState'. Each class will have methods *void Action(), void Reset()* and *void GUIImage()* implemented and publically accessible. These classes are all called *States*. Note that *States* do not have any inheritance relation to class *MonoBehaviour* and thus they are not script components, hence they cannot employ methods *Start(), Update()*etc. MainManager game object transfers part of the execution of its methods *Update()*and *OnGUI()* to methods possessed by objects of the IState interface pointed by *activeState* variable. ActiveState variable changes its value during execution by pointing to a different IState object. Thus, the application functions dynamically depending on the methods of *State* that are active in component *MainManager, the succession of States though is set in MainMnager's Start() method, where they are also instantiated*. The advantage of this approach is that the application can be extended or modified by adding a *State* or altering the order of *States* in the pertinent array, respectively.

4.4 Functioning of the application

The *InitialState* is the introductory scene. The *ScaleState* displays the sensor measurement requirement. *ScaleStateTwo* represents the capture of part height by the sensor or, alternatively, its override by manual input of this information,

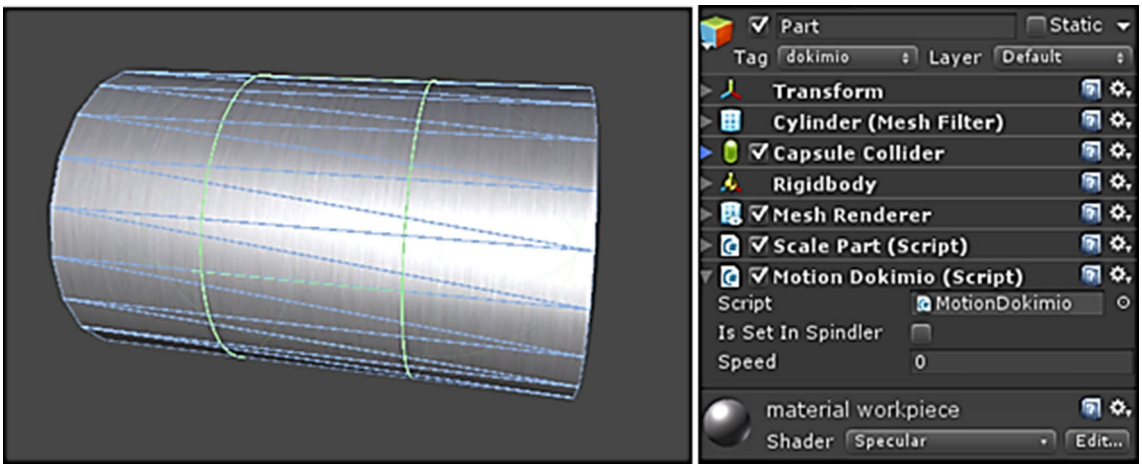

Fig. 5 'Part' object definition

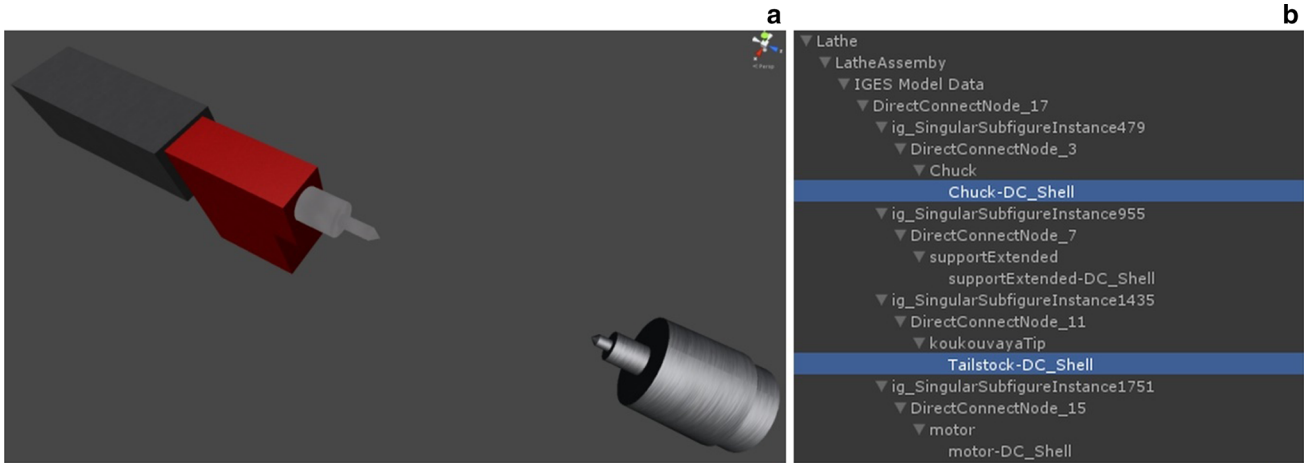

Fig. 6 Lathe object **a** graphics **b** structure

see Fig. 7a, b. *SetInSpindleState* makes the virtual clamping devices visible, see Fig. 7c and, then moves the virtual part to the desired clamping position, see Fig. 7d, allowing for repetition of the motion and displaying an appropriate message to the user. *SetInTailstockState* instructs the desired position of the virtual tailstock for proper clamping of the virtual part, see Fig. 7e.

5 Setup instructions for a machining centre

This application provides real-time instructions for clamping of parts on a machining centre. The same standard marker to be recognised was employed as in the first application, see Fig. 4a, its dimensions being 20×20 cm, for seamless target recognition.

5.1 Game objects

Game object *Manager* possesses script component *MainManager*, which materialises the following functions.

It can accept an object of a class that implements interface *IState, through variable activeState*. Such classes will have methods *void Action()* and void GUIImage() implemented and publically accessible. This allows them to be called without knowing beforehand the class from which their implementation will be effected. In this way, implementation of methods *void Update() void OnGUI()* which are inherited from class *MonoBehaviour* is transferred to objects of other classes. This practice is followed in order to allow the behaviour of game object *Manager* to be dynamically modified during application execution. The difference to the similar logic of the first application is that component *MainManager* is not responsible for the object to which variable activeState will point during execution of the application, except for objects of class *InitialScreenState*, which is initialised by method *voidStart()*.

It provides public method *void SwitchState (IState nextState).* When this is called from another object, it enables change of State pointed to by variable activeS-

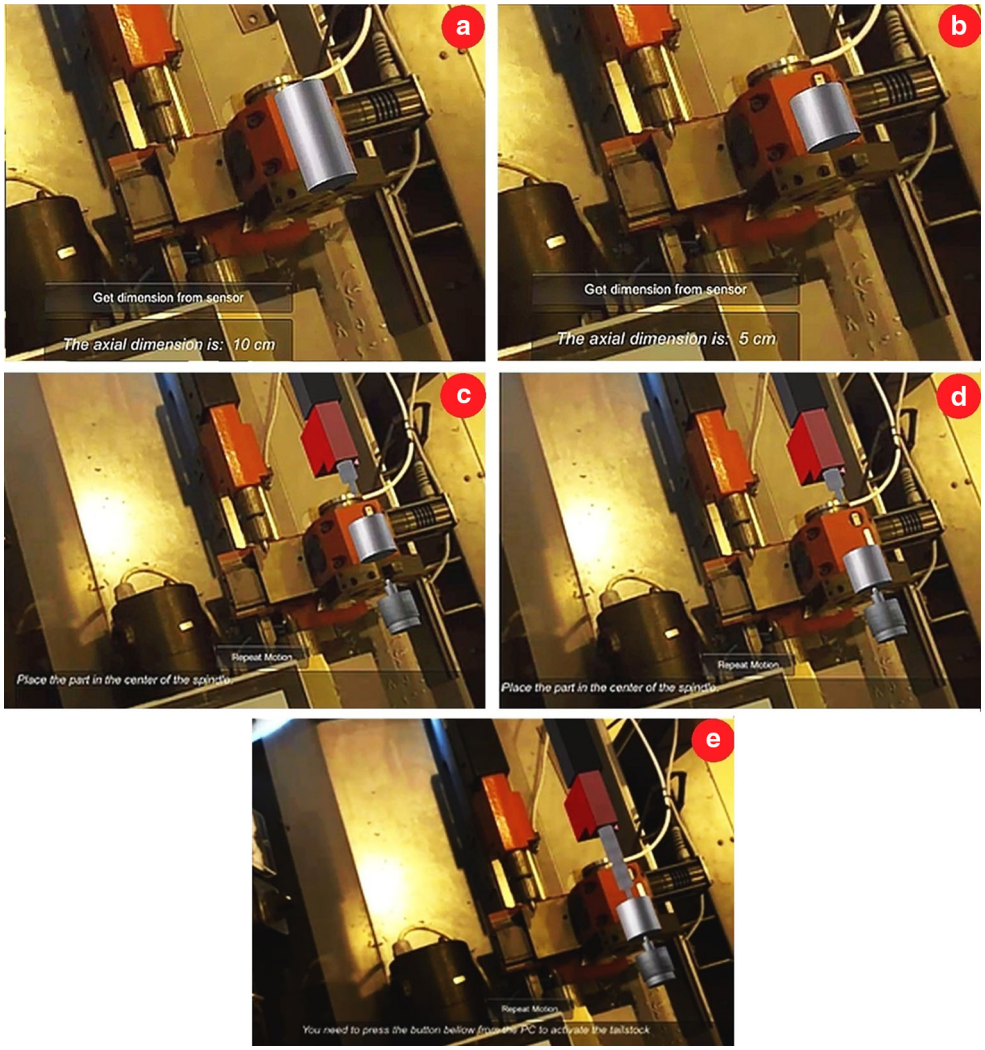

Fig. 7 First application **a** ScaleStateTwo: small height of part **b** ScaleStateTwo: large height of part **c** virtual clamping device emergence **d** desired setup position of the virtual part with respect to the chuck **e** desired position of the tailstock

tate of the *MainManager* and this automatically changes the implementation of *voidUpdate()* and *void OnGUI()*.

It provides publically accessible variables *prefab, image, guiSkin, defaultGuiSkin*. Variables *guiSkin* and *defaultGuiSkin* are responsible for buttons, fonts and dialogue windows that emerge during execution of the application. Variable *image* is a Texture2D array of images and variable *prefab* maps objects to be created by the application to those that are present in folder *Prefabs* which are fully parametrically defined.

It provides methods *CreatePrefab* and *DestroyPrefab*. The first one retrieves an object from the array prefab and makes it into a child of gameobject 'ImageTarget' which is already in the scene. This method has been overloaded with five different implementations to optionally provide additional parameters of the object created. The second method destroys gameobjects. These methods are employed in various States, hence objects comprising the scene change dynamically.

5.1.1 Definition of parts to set up

The parts to be setup are all modelled as described in Sect. 3.1 and then created as *prefabs*, see Fig. 8. They emerge in the scene after the target is recognised by the camera and a setup scenario is instructed linked to part orientation and jigs and fixtures available. They contain the following components:

- *Rigidbody*, with checked *IsKinematic* option, so as to compute movement of the object by *Transform* component and not by *Physics Engine*. The same option applies to all objects except for *ARCamera*.
- *BoxCollider*, with checked *IsTrigger* option. This Component defines a reference volume around the object, with respect to which events of interest are identified, e.g. intrusion by another reference volume, selection by the user etc. Such events can be monitored by other components and be managed by activating specific methods.

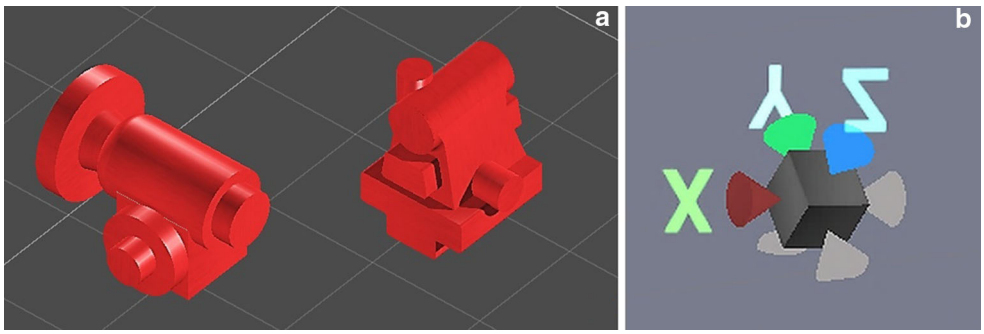

Fig. 8 Prefabs **a** examples of parts to be setup **b** orientation cube

- *MovingPart*, a script component, which assigns motion behaviour to the object. Publically accessible methods are defined to: (a) instantly change orientation of an object to one of 6 predefined standard orientations, (b) linearly move the object along X or Y or Z axis in the positive or negative direction with constant speed and (c) rotate the object cw or ccw with constant speed about Y axis.
- *MouseListener:* possesses two variables: one to change State every time that the object is selected and another one to count the number of selections. These variables are assigned values by a method called by colliders with an active IsTrigger option as well as Physics.queriesHit Triggers.

An auxiliary prefab, *OrientationCube,* was defined to indicate the orientation axes of the parts. This consists of six children objects and possesses script *MovingPart* presented above.

5.1.2 Clamping fixtures

The main clamping fixtures are chucks and vises. They are defined as prefabs. Prefab 'Chuck' consists of four children, see Fig. 9a. The parent object contains script component *ChuckMover*, which via three methods opens or closes the chuck's jaws with constant speed and moves the jaws instantaneously to a specific position. Prefab ***Vise*** contains eight children, see Fig. 9b. The parent object contains script component *ViseMover,* which opens/closes the vice jaw at constant speed, or moves the vice forward/backward along local X-axis.

Before instructing the user to clamp a part on a chuck or vice, the way to secure these devices on the machining centre table needs to be demonstrated, too. The mechanical fittings available for this purpose, also defined as prefabs, are shown in Fig. 10.

Most of the prefabs can perform movements that are useful for part clamping using scripts attached to them. *Flanged nut,* see Fig. 10b, moves down (at constant speed) and rotates in a screwing pattern, *clamp*, see Fig. 10c, moves up/down and rotates cw/ccw, *Threaded studs* move right or left along table T-slots, *StepBlock*, see Fig. 10f, moves forward and back. CAD drawings for all these prefabs were originally taken from manufacturers catalogues and were finally stored in the prefab folder of the application, much like a parametric part library. The different types of moves assigned to objects simulate their real counterparts that the user/setter is expected to apply in practice.

These objects are combined into assemblies with hierarchical relations, usually the *Stud* being the parent of the object tree, since its axis of symmetry constitutes the positional reference. For instance, auxiliary prefab *AssemblySlideStud* is a combination of prefabs*Slide* and*Threaded Stud*. *AssemblySlideStud* is used to instruct how to secure a *Stud* into the *Slide*. This is achieved by script *BoltMover* which makes the *Stud* move towards the *Slide* and then simulate screwing the completion of which is signalled by a Boolean variable. Note that the *Slide* and corresponding part of the *Stud* are invisible when they are inside the T-slot of the table, see Fig. 11; therefore the *Slide* is made invisible indeed, whilst the *Stud* is replaced by a shorter one, i.e. a prefab of type *CutThreadedStuds*.

5.1.3 Videos

The actions expected by the user/setter have been recorded on video. These video as are made into prefabs: FirstVideo, ChuckVideo, ViseVideo. Such prefabs are Plane game objects and, when they appear on the scene, they become children of game object *background Plane*, so that they appear at a fixed position in the scene during application execution. The plane is effectively defined as projection surface of the video through script component *VideoHolder*. Component *VideoObserver* updates objects of other classes.

5.2 Functioning of the application

Some game objects are present in the scene from the beginning whilst prefabs enter the scene during execution of the application. Calling of prefabs, their combination and manip-

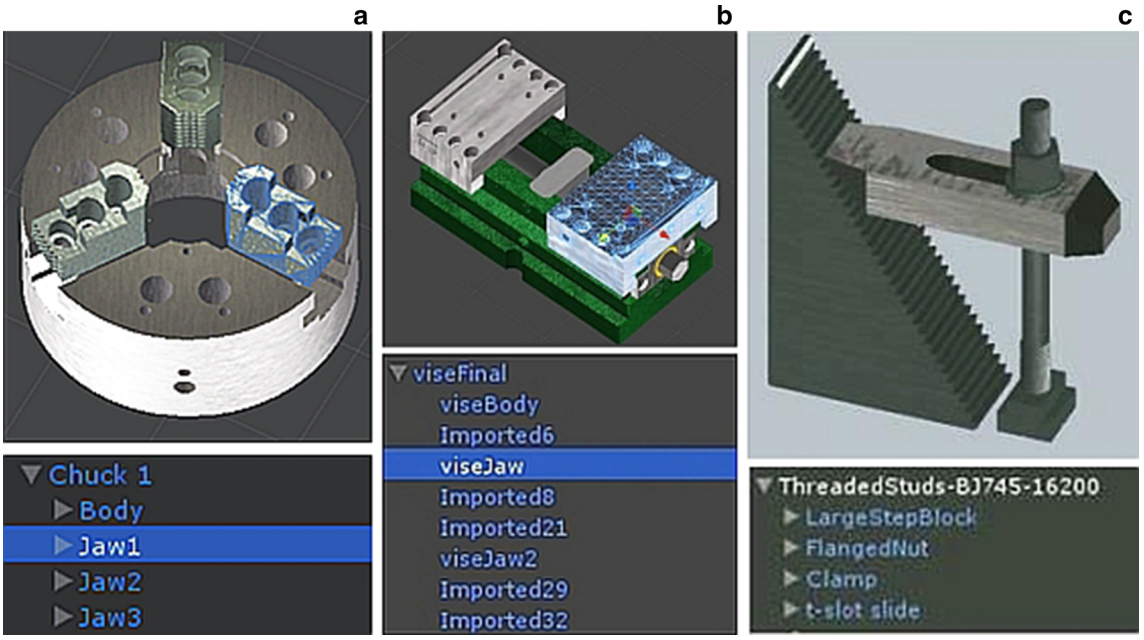

Fig. 9 Clamping fixture structure examples **a** chuck **b** vice **c** threaded stud

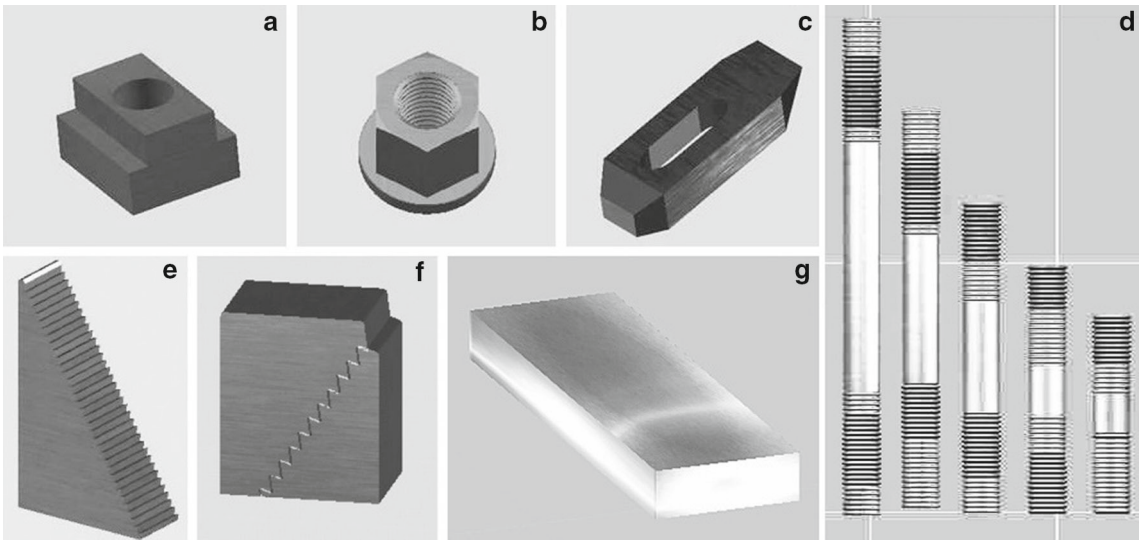

Fig. 10 Prefabs **a** slide **b** nut **c** claw **d** threaded studs **e** wedge **f** step block **g** parallel plate

ulation are performed through States that are alternated by user selection. A number of States have been defined and are presented next.

5.2.1 State structuring

All classes that produce States have the following elements:, inheritance statement, constructor for variable initialisation, implemented method *Action*, implemented method *GUIImage*. The constructor is called each time that an object of a class is instantiated and is linked to gameobject Manager in order to access the methods that Create or Destroy prefabs as well as *Switch* States. Method *Action* contains the code that dictates the movement of the virtual objects in the scene. Each State determines a series of motions, delimited by logic conditions, which achieve better control and accuracy of object positioning, rather than by Colliders, which were used in the turning setup application. This is enabled by the fact that none of the objects involved undergo shape or dimension changes during execution of the present application. In addition, method *GUIImage* defines parametric frames at specific

Fig. 11 Slide and Threaded stud inside a T-slot, the other parts being: step block, wedge, claw and flanged nut

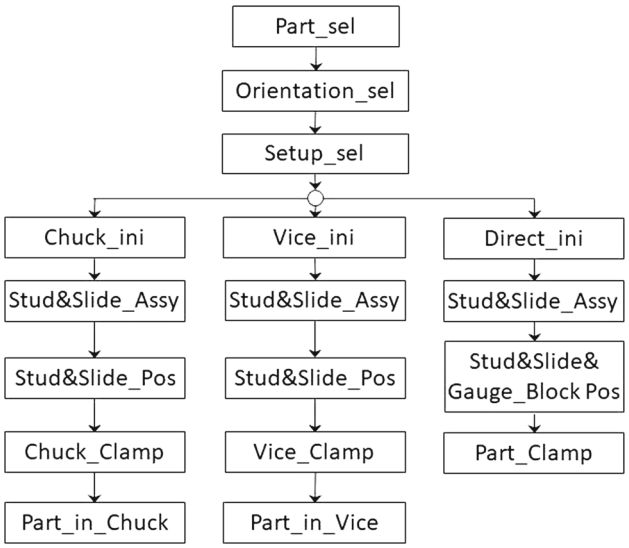

Fig. 12 States and threads defined in the application

regions of the screen, where messages and images are projected to guide interaction between use and the application.

5.2.2 Examples

The States created in this particular application and the transitions from each one to the next are shown in Fig. 12. Awaiting message is displayed upon application launch, until the target is recognised. The next *activeState* concerns Part selection, which displays the parts available in the library and prompts the user to select the one to set up. The selected part is first rotated so as to allow the user to inspect it, whilst the rest of the parts are withdrawn from the scene. Pressing on the button 'confirm', *activeState* concerns now Orientation selection, which generates a list of orientations for the user to select from, helped by gameobject *OrientationCube*, see Fig. 8b, which also emerges in the scene.

The next *activeState* depends on the part-orientation combination: for n parts and m possible orientations per part *nXm* scenarios are possible and the relevant variable is used by the auxiliary S*tate* concerning setup selection. In the particular implementation of the application, three different State threads have been defined, see Fig. 12, but these are obviously extensible.

The first State thread concerns use of a chuck as main clamping element. First, the chuck is placed on the table, see Fig. 13(a-1), then the studs are assembled with corresponding slides, see Fig. 13(a-2), the assemblies are slotted into the table at specific positions, see Fig. 13(a-3), and the chuck is clamped by claws attached to the studs by flanged nuts, see Fig. 13(a-4). Last, the part is clamped in the chuck, see Fig. 13(a-5). The second State thread concerns use of a vice as main clamping element and the instructions provided are analogous to those given for the chuck, see Fig. 13b. The third State thread concerns direct clamping of parts on the table by use of modular clamping systems, see Fig. 14. First, the part is oriented, see Fig. 14(1), then the necessary studs are placed on the table, see Fig. 14(2), and slotted into the table in appropriate positions, see Fig. 14(3). Gauge blocks and step blocks are appropriately placed, see Fig. 14(4) and, last, claws are positioned on the studs and secured by flanged nuts to clamp the part, see Fig. 14(5–6).

6 Discussion

After pilot-testing the applications with users in an academic environment several comments were made leading to improvement suggestions by the application development engineers, as follows.

A first issue concerned the restricted field of view of the conventional cameras employed, namely web cameras or even tablet cameras, which dictates an increase in the distance between camera and machine tool proportional to the volume or table surface to be captured. Machine tools are generally large devices and according to the applications developed the user needs to be able to see a significant part through the visualisation device while still working on the machine tool. For this reason, a fixed camera was ultimately made use of, placed at a high-up point, whilst a computer monitor is used as visualisation device. In this way, the user is free to move without holding any piece of equipment, whilst the large size of the monitor compared to a tablet's or mobile phone's size makes for clear pictures form a larger distance. However,

Fig. 13 Part setup instructions using **a** chuck **b** vise

navigation in the application is performed through a mouse which is still intrusive with respect to the user's normal flow of work. An improvement would certainly be replacing the mouse by a device allowing more intuitive interaction (e.g. a tracking device), allowing the user to control the system seamlessly. A short-term future plan is to run the application with latest technology AR goggles, such as Microsoft HoloLens™.

Another issue, which is well-known, was the significant influence of the lighting conditions in the task of marker recognition, especially in too bright or too dark areas. In

Fig. 14 Part setup instructions using modular elements for direct clamping

extreme cases marker recognition may be impossible with a conventional camera alone.

A third issue concerns dependence on the recognition of the marker-target. This could be improved, if, instead of the marker, the machine tool table or the chuck and tailstock respectively, were possible to recognise as such. Then, the application could be executed on any machine tool of similar configuration. However, the use of markers does not hinder the universality of the approach so long as the used marker is visible in the scene.

A final issue concerns the effort that is necessary in order to develop models and scenarios for all parts that are to be processed on the machines, making use of alternative clamping equipment available in the factory. The digitised equipment is copied from the inventory of the factory. Given the relative standardisation and the availability of CAD models from vendors' sites, creating models for such equipment at a factory-wide scale should not be a problem. The same applies to parts, which are normally modelled on CAD anyway, whereas intermediate forms (semi-finished etc.) should also be accessible through CAM systems. What is perceived as a potential bottleneck is the creation of clamping scenarios, i.e. authoring tasks. However, the concept of states and their potential reuse in different threads imparts some standardisation for certain, thereby reducing authoring effort overall.

Machine setup instructions which were demonstrated to be offered in real time on the actual machine tool have the obvious advantages of immediacy and realism. In this light, this paradigm can be considered superior to the conventional manner based on paper or digital drawings, or even based on verbal informal explanations which is commonplace. The first feedback by six professional tool room personnel to whom the application was demonstrated was very positive, supporting, thus, the hypothesis of efficiency (in terms of clarity of instructions) of such AR guidance applications in manufacturing. However, user tests need to be performed to prove such superiority, both in the laboratory tool room setting for training, as well as in an industrial setting, and under realistic circumstances in terms of workload.

7 Conclusions

Augmented Reality applications in Manufacturing have not attracted the intensity of attention that could have been

devoted to them, which is attributed partly to lack of suitable equipment incl. visualisation devices that can be used in an industrial environment, partly to conservative thinking resulting in lack of convincing applicability and partly to the lack of a suitable conceptual framework in which such applications would be exploited.

The scope of this paper is to point out the feasibility of such novel, low-cost, efficient and user friendly industrial AR applications. This work is focusing on technical aspects of AR application development with Unity™ game engine. The use of low-cost interfaces and sensors in a structured scenario (in terms of states and state threads) ensures expansibility and flexibility, and clearly demonstrates the way of authoring such applications.

Interaction in Manufacturing System execution entails a virtual model of part of the system whereas the rest is the physical system. Interaction is based on primarily visual feedback as well as further sensory feedback regarding the expected part position and orientation with respect to the machine from the virtual model environment. This is enhanced by additional readily furnished information (primarily via multimedia such as videos, sounds and photos) regarding step-by step instructions of machine setup, all making for right-first time setup actions by the user. In addition, in interactive manufacturing system execution the 'standard' expected behaviour of objects forms the basis for effectively delimiting the boundary in 3D space beyond which the user cannot act. As an example, the kinematics of moving parts of a jig or fixture for part clamping or positioning delimit the size of the part that can be held, the movements required to bring it to the clamping position etc.

In this work, low cost devices, such as ultrasonic distance sensors and web cameras have been employed in conjunction with an AR-tailored add-in to a πowerful and widely used open source game development engine. The targeted paradigm involves provision of instructions for setting up machine tools by (a) demonstrating them in the real environment rather than reading them out from drawings or even from CAD models (b) demonstrating them on the actual machine tool rather than on a generic one. A large variety of parts, setup plans and holding devices is dealt with through a database. Universal application of digital tools and generality of scripting safeguard the universality of application. The framework within which such applications are envisaged to function is that of a digitalised production, commonly referred to as Industry 4.0.

The tools and techniques employed for application development were purposefully presented to an appreciable level of detail (instructions, realistic animations, videos etc.), by contrast to common practice, since they may serve as a useful guide for developing similar applications, in particular starting with the concept of States to model the required functionality as well as to enable easy modifications and extensions.

Inclusion of sensors in the applications adds intelligence, independence from human input and associated possible error fostering their interactive character.

In general, hardware improvement should add to the alleviation of restrictions related to the support of manufacturing system execution tasks by AR. In particular, as tracking devices become less intrusive, the envisaged application will become more feasible in day-to-day factory working conditions. In addition, marker-less recognition technology has already been successfully implemented in relation to AR, e.g. ARKit™ in Unity3D™ [36] and its adoption should be straightforward without substantial modification of the application scripts.

Future work is planned for assessing the developed applications in a structured user test by professional workers/setters, as well as for comparing them to their conventional counterparts. An interesting future project might also be the assessment of the applications in the context of training novices. A complementary domain to work on in the future is the data management and communication requirements within the factory network regarding support of multiple machines in parallel.

References

1. Billinghurst, M., Clark, A., Lee, G.: A survey of augmented reality. Found. Trends® Hum.–Comput. Interact. **8**(2–3), 73–272 (2015)
2. van Krevelen, D.W.F., Poelman, R.: A survey of augmented reality technologies, applications and limitations. Int. J. Virtual Real. **9**(2), 1–20 (2010)
3. Fite-Georgel, P.: Is there a reality in industrial augmented reality? In: 2011 10th IEEE International Symposium on Mixed and Augmented Reality, pp. 201–210 (2011)
4. Elia, V., Gnoni, M.G., Lanzilotto, A.: Evaluating the application of augmented reality devices in manufacturing from a process point of view: an AHP based model. Expert Syst. Appl. **63**, 187–197 (2016)
5. Davies, P., Lee, D.: Augmented Reality in Manufacturing at the Boeing Company Lessons Learned and Future Directions (2014)
6. Yew, A.W.W., Ong, S.K., Nee, A.Y.C.: Towards a griddable distributed manufacturing system with augmented reality interfaces. Robot. Comput. Integr. Manuf. **39**, 43–55 (2016)
7. Gorecky, D., Mura, K., Arlt, F.: A vision on training and knowledge sharing applications in future factories. IFAC Proc. **46**(15), 90–97 (2013)
8. Doshi, A., Smith, R.T., Thomas, B.H., Bouras, C.: Use of projector based augmented reality to improve manual spot-welding precision and accuracy for automotive manufacturing. Int. J. Adv. Manuf. Technol. **89**(5–8), 1279–1293 (2016)
9. Syberfeldt, A., Danielsson, O., Holm, M., Wang, L.: Visual assembling guidance using augmented reality. Procedia Manuf. **1**, 98–109 (2015)
10. Fiorentino, M., Uva, A.E., Gattullo, M., Debernardis, S., Monno, G.: Augmented reality on large screen for interactive maintenance instructions. Comput. Ind. **65**(2), 270–278 (2014)

11. Alvarez, H., Aguinaga, I., Borro, D.: Providing guidance for maintenance operations using automatic markerless Augmented Reality system. In: 2011 10th IEEE International Symposium on Mixed and Augmented Reality, pp. 181–190 (2011)
12. Wang, X., Ong, S.K., Nee, A.Y.C.: A comprehensive survey of augmented reality assembly research. Adv. Manuf. **4**(1), 1–22 (2016)
13. Radkowski, R., Herrema, J., Oliver, J.: Augmented reality-based manual assembly support with visual features for different degrees of difficulty. Int. J. Hum. Comput. Interact. **31**(5), 337–349 (2015)
14. Lamberti, F., Manuri, F., Sanna, A., Paravati, G., Pezzolla, P., Montuschi, P.: Challenges, opportunities, and future trends of emerging techniques for augmented reality-based maintenance. IEEE Trans. Emerg. Top. Comput. **2**(4), 411–421 (2014)
15. Pirvu, B.-C., Zamfirescu, C.-B., Gorecky, D.: Engineering insights from an anthropocentric cyber-physical system: a case study for an assembly station. Mechatronics **34**, 147–159 (2016)
16. Rentzos, L., Papanastasiou, S., Papakostas, N., Chryssolouris, G.: Augmented reality for human-based assembly: using product and process semantics. IFAC Proc. **46**(15), 98–101 (2013)
17. Syberfeldt, A., Danielsson, O., Holm, M., Wang, L.: Dynamic operator instructions based on augmented reality and rule-based expert systems. Procedia CIRP **41**, 346–351 (2016)
18. Jo, G.S., Oh, K.J., Ha, I., Lee, K.S., Hong, M.D., Neumann, U., You, S.: A unified framework for augmented reality and knowledge-based systems in maintaining aircraft. Proc. Natl. Conf. Artif. Intell. **4**(October 2015), 2990–2997 (2014)
19. Flatt, H., Koch, N., Rocker, C., Gunter, A., Jasperneite, J.: A context-aware assistance system for maintenance applications in smart factories based on augmented reality and indoor localization. In: 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA), pp. 1–4 (2015)
20. Gorecky, D., Schmitt, M., Loskyll, M., Zühlke, D.: Human–machine-interaction in the industry 4.0 era. In: Proceedings - 2014 12th IEEE International Conference on Industrial Informatics (INDIN 2014), pp. 289–294 (2014)
21. Rodriguez, L., Quint, F., Gorecky, D., Romero, D., Siller, H.R.: Developing a mixed reality assistance system based on projection mapping technology for manual operations at assembly workstations. Procedia Comput. Sci. **75**, 327–333 (2015)
22. Mura, M.D., Dini, G., Failli, F.: An integrated environment based on augmented reality and sensing device for manual assembly workstations. Procedia CIRP **41**, 340–345 (2016)
23. Kollatsch, C., Schumann, M., Klimant, P., Wittstock, V., Putz, M.: Mobile augmented reality based monitoring of assembly lines. Procedia CIRP **23**(C), 246–251 (2014)
24. Zhou, J., Lee, I., Thomas, B., Menassa, R., Farrant, A., Sansome, A.: Applying spatial augmented reality to facilitate in-situ support for automotive spot welding inspection. In: Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry—(VRCAI'11), p. 195 (2011)
25. ART: ART implements 'Window To The World' at AUDI
26. Wang, J., Feng, Y., Zeng, C., Li, S.: An augmented reality based system for remote collaborative maintenance instruction of complex products. In: 2014 IEEE International Conference on Automation Science and Engineering (CASE), pp. 309–314 (2014)
27. Garza, L.E., Pantoja, G., Ramírez, P., Ramírez, H., Rodríguez, N., González, E., Quintal, R., Pérez, J.A.: Augmented reality application for the maintenance of a flapper valve of a fuller-kynion type m pump. Procedia Comput. Sci. **25**, 154–160 (2013)
28. Rolim, C., Schmalstieg, D., Kalkofen, D., Teichrieb, V.: Design guidelines for generating augmented reality instructions. In: 2015 IEEE International Symposium on Mixed and Augmented Reality, pp. 120–123 (2015)
29. Petersen, N., Stricker, D.: Cognitive augmented reality. Comput. Graph. **53**, 82–91 (2015)
30. Ramirez, H., Mendivil, E.G., Flores, P.R., Gonzalez, M.C.: Authoring software for augmented reality applications for the use of maintenance and training process. Procedia Comput. Sci. **25**, 189–193 (2013)
31. Paelke, V.: Augmented reality in the smart factory: supporting workers in an industry 4.0. environment. In: Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), pp. 1–4 (2014)
32. Loch, F., Quint, F., Brishtel, I.: Comparing video and augmented reality assistance in manual assembly. In: 2016 12th International Conference on Intelligent Environments (IE), pp. 147–150 (2016)
33. Webel, S., Bockholt, U., Engelke, T., Gavish, N., Olbrich, M., Preusche, C.: An augmented reality training platform for assembly and maintenance skills. Robot. Auton. Syst. **61**(4), 398–403 (2013)
34. Monostori, L.: Cyber-physical production systems: roots, expectations and R&D challenges. Procedia CIRP **17**, 9–13 (2014)
35. PTC: Unity API Reference
36. Maskrey, M., Wang, W.: Interacting with augmented reality. In: Pro iPhone Development with Swift 4, pp. 419–446. Apress, Berkeley, CA (2018)