Training Manual

**Name: Xuan Linh Tang**

SID: 218559008

Concept and context:

Concept: Interactive ArchViz, specially focus on interior decor

Context: In this case, the application is focusing on interior design of the house by generating 3D model of a real furniture/décor and placing that model into a real room through AR camera on a smartphone. Furthermore, the application also provides some extra useful features such as space measurement, voice command etc. to bring the most comfortable experience to users.

Application title: AR Furniture.

Project manager:

Full name: Xuan Linh Tang

Phone number: +61444547079

Email address: [xltang@deakin.edu.au](mailto:xltang@deakin.edu.au)

Team and roles:

Client: Shaun Bangay

Project manager: Xuan Linh Tang

Designer: Xuan Linh Tang

Developer: Rohan Brahmakshatriya

Q/A testing: to be assigned

**I. Updated Requirements:**

After receiving feedback from the client, the list of requirements needs to be adjusted to reflect more AR potential features and provide more real experiences to users. The most major changes are adding physical measurement and real-time object detection to the application. Also, the 3D scanning feature is removed, due to the complicated setting up additional devices. The latest list includes below requirements:

**Furniture preview features:**

1. The user should be able to generate virtual objects and put them in to the real scene utilizing the smartphone camera.
2. The application should be able to track physical location of the objects, even when the camera is not pointing at them.
3. The application should be able to detect horizontal and vertical planes of the real scene, so that generated objects will land on those surfaces instead of floating in the air.
4. The virtual objects should be able to move and rotate as the user see fit.

**Physical measurement features:**

1. The user should be able to measure the physical space between two points in the detected surfaces.
2. The application should be able to detect objects in the real-time and measure their size.

**Modalities:**

1. The user should be able to use the voice commands to control some functionalities.

**UI requirements:**

1. The application should have a menu for the user to choose the type of furniture.
2. The application should have a button to toggle on/off the physical measurement feature.
3. The application should have a button to toggle on/off the object detection feature.
4. The application should have a button to toggle on/off the plane grid.

**II. Implementation guideline:**

You can follow the guideline below for each of the requirement. The full source codes are provided in the related prototype.

**1. The user should be able to generate virtual objects and put them in to the real scene utilizing the smartphone camera:**

|  |  |
| --- | --- |
| **Instruction** | 1. Download AR Core for Unity SDK from <https://github.com/google-ar/arcore-unity-sdk/releases>. The version I used to developed prototype is v1.11.0. Import the package to Unity.  2. Add Asset/GoogleARCore/Prefabs/ARCore Device to the scene. Delete the default main camera. After this step, the scene seen on the phone screen is the scene captured from camera. It is worth to notice here that AR Core sync the Unity coordinate with the real-world position in the very first moment when the application opens. In the very first update, the current position of the phone is (0,0,0), the positive Z-axis is the forward direction of the camera, positive X-axis is the right side of the camera, positive Y-axis is the upper of the camera. 1 unit in Unity equal to 1 meter in the real world.  3. Add an empty object called *ObjectController* to the scene. Add the *ObjectController.cs* script to it. Create a cube prefab (the cube represents the furniture). Add a button to the scene, when ever the button is clicked, a cube is generated at position (0,0,2) (this is just a test position to see if the virtual object overlay appear on the real world) |
| **Testing process** | 1. Make sure that the scene appear on the phone is taken from rear camera after step 1 and 2  2. Make sure that you see the cube appear on the phone screen when click the button after step 3 (Do not move the phone camera yet in this step to make sure the camera point at the Z-axis) |
| **Related Prototype:** | AR Object Control, <https://github.com/deakinshaun/arvrapps-t2-2019/tree/Linh_DesignerBranch/Prototypes/ARObjectControl> |

**2. The application should be able to track physical location of the objects, even when the camera is not pointing at them:**

|  |  |
| --- | --- |
| **Instruction** | 1. Start with the scene from Requirement 1  2. Because AR Core support the feature to sync the Unity coordinate with the real-world position, the location of the objects will be tracked even when the camera is not pointing at them. |
| **Testing process** | 1. Rotate the camera around. You should see the object at the exact same spot in the real world via camera screen. |
| **Related Prototype:** | AR Object Control, <https://github.com/deakinshaun/arvrapps-t2-2019/tree/Linh_DesignerBranch/Prototypes/ARObjectControl> |

**3. The application should be able to detect horizontal and vertical planes of the real scene, so that generated objects will land on those surfaces instead of floating in the air:**

|  |  |
| --- | --- |
| **Instruction** | 1. Start with the scene from Requirement 1  2. Add Assets/GoogleARCore/Examples/Common/Prefabs/PointCloud.prefab to the scene. This component detects the feature points and visualize them as blue dots on the screen.  3. Create empty object named *PlaneGenerator*. This object purpose is to visualize the detected planes. It also creates mesh for the plane to support the collision. Add this script Assets/GoogleARCore/Examples/Common/Scripts/DetectedPlaneGenerator.cs to the object. In the field *Detected Plane Prefab*, chose Assets/GoogleARCore/Examples/Common/Prefabs/DetectedPlaneVisualizer.prefab.  4. To enable the collision of the plane, we need some modification. Open DetectedPlaneVisualizer prefab and add component *Mesh Collider*. Then open Assets/GoogleARCore/Examples/Common/Scripts/DetectedPlaneVisualizer.cs, add these two lines at the end of *\_UpdateMeshIfNeeded()* function:  GetComponent<MeshCollider>().sharedMesh = null;  GetComponent<MeshCollider>().sharedMesh = m\_Mesh;    5. Now make a small red point in the centre of the screen to indicate where the camera is looking at by adding an image object in the scene.  6. AR Core provides Raycast function to detect the hit when cast a ray at a point in the phone screen to the detected plane. We want to create object at the centre of the screen, so we cast a ray from the centre of the screen to the detected plane, the hit will be the position of the furniture. Also, we want to add a little offset in the Y-axis *distanceFromSurface* to simulate the falling effect of the object. Remember to enable the Gravity option in the rigid body component of the furniture.  TrackableHit hit;  TrackableHitFlags raycastFilter = TrackableHitFlags.PlaneWithinPolygon |  TrackableHitFlags.FeaturePointWithSurfaceNormal;  //cast a ray from center screen to the detected plane  Frame.Raycast(Screen.width \* 0.5f, Screen.height \* 0.5f, raycastFilter, out hit);  Vector3 objectPos = hit.Pose.position + distanceFromSurface;  GameObject gameObject = Instantiate(gameOjbectPrefab, objectPos, Quaternion.identity); |
| **Testing process** | 1. After step 2, you should see the blue dot appear on the screen when the feature points are detected.  2. After step 3, you should see the visualization of the detected plane as a white grid. Try to scan some flat surface such as the floor, table top etc. to see the effect.  3. After step 5, you should see a small red dot at the centre of the screen.  4. After step 4 and step 6, when you click the create object button, it should appear at the red dot point, with a falling effect and then stop when it lands on the detected surface. |
| **Related Prototype:** | AR Object Control, <https://github.com/deakinshaun/arvrapps-t2-2019/tree/Linh_DesignerBranch/Prototypes/ARObjectControl> |

**4. The virtual objects should be able to move and rotate as the user see fit:**

|  |  |
| --- | --- |
| **Instruction** | 1. Start with the scene from Requirement 3  2. Create a button at the centre bottom of the screen to pick object. Unity provides method *ScreenPointToRay* to create a ray from a point on the screen. To select objects, cast a physic raycast from the red dot (screen centre point), if the game object hit is tagged “furniture”, select that object.  RaycastHit hit;  Ray ray = ARCam.ScreenPointToRay(new Vector2(Screen.width \* 0.5f, Screen.height \* 0.5f));  if (Physics.Raycast(ray, out hit, 30.0f))  {  if (hit.transform.gameObject.CompareTag("furniture"))  {  selectedObject = hit.transform.gameObject;  Debug.Log("selected object name: " + selectedObject.name);  }  }    3. When object is selected, it will be lifted up a little. Object will move along with the camera, and its position is defined by the hit of the ray cast from the red dot to the detected plane. Object rotation is controlled by the swipe action on the phone screen.  if (selectedObject != null)  {  TrackableHit hit;  TrackableHitFlags raycastFilter = TrackableHitFlags.PlaneWithinPolygon |  TrackableHitFlags.FeaturePointWithSurfaceNormal;  Frame.Raycast(Screen.width \* 0.5f, Screen.height \* 0.5f, raycastFilter, out hit);  selectedObject.transform.position = hit.Pose.position + distanceFromSurface;  if (Input.touchCount == 1)  {  Touch touch = Input.GetTouch(0);  float deltaX = touch.deltaPosition.x;  selectedObject.transform.Rotate(0, deltaX \* 0.2f , 0);  }  } |
| **Testing process** | 1. After step 2, you should be able to point the red dot at the object and be able to select that object. You will know the selection is success if the object is lifted up a little.  2. After step 3, you should be able to select object, then move it through the movement of the camera. While moving, the object still stay on top of the plane. The rotation of the object is tested by swipe the finger left and right and see if the object is rotated. |
| **Related Prototype:** | AR Object Control, <https://github.com/deakinshaun/arvrapps-t2-2019/tree/Linh_DesignerBranch/Prototypes/ARObjectControl> |

**5. The user should be able to measure the physical space between two points in the detected surfaces:**

|  |  |
| --- | --- |
| **Instruction** | 1. Follow step 1,2,3,5 of requirement 3.  2. Create empty object named *PhysicalMeasurement*. Add *LineRenderer* component to this object to visualize the measurement line. Make a jpg image with red background and add it to the *LineRenderer* material. In the *Positions* field, set Size to 2, and both points position to (0,0,0). Set the width of the *LineRenderer* to 0.01  3.Add a “Start” button to start/end the measurement. Add a text to display the distance.  4. The principle of physical measurement used in this application is the distance calculation between two defined point on the detected surface. AR Core already provide the possibility to remember the physical location of points, so we just need a simple math formula to calculate the distance of two point in XYZ coordinate.  When click on “Start” button, the first point of the measurement line will be the hit of the ray from the red dot (screen centre point) to the detected surface. We set the first point of the *LineRenderer* component to this point. More details could be found in Assets/Scripts/PhysicalMeasurement.cs  public void Measure()  {  if (!isMeasuring)  {  //cast a ray from center screen to the detected plane  Frame.Raycast(Screen.width \* 0.5f, Screen.height \* 0.5f, raycastFilter, out startPoint);  measureLine.SetPosition(0, startPoint.Pose.position);  isMeasuring = true;  MeasureConfirm.GetComponentInChildren<Text>().text = "End";  }  else  {  isMeasuring = false;  MeasureConfirm.GetComponentInChildren<Text>().text = "Start";  }  }  5. While the measuring is happening, the second point of the line will be continuously updated based on the hit of the ray from the red dot to the detected surface. We set the second point of the *LineRenderer* component to this point. The measured distance will be display as a text at the top of the screen. More details could be found in Assets/Scripts/PhysicalMeasurement.cs  void Update()  {  if (isMeasuring)  {  Frame.Raycast(Screen.width \* 0.5f, Screen.height \* 0.5f, raycastFilter, out endPoint);  measureLine.SetPosition(1, endPoint.Pose.position);  //get the distance  float dx = startPoint.Pose.position.x - endPoint.Pose.position.x;  float dy = startPoint.Pose.position.y - endPoint.Pose.position.y;  float dz = startPoint.Pose.position.z - endPoint.Pose.position.z;  float distanceMeters = (float)Mathf.Sqrt(dx \* dx + dy \* dy + dz \* dz);  distanceText.text = distanceMeters + "m";  }  } |
| **Testing process** | 1. First, scan to detect the surface. Point the red dot to one end of the measurement line, press “Start” (for example, you can point at one leg of the chair like the image above). You should begin to see the red line run from the start point to the point currently the camera looking at. You also see the text displaying the distance on the top of the screen. |
| **Related Prototype:** | Object Detection and Physical Measurement, <https://github.com/deakinshaun/arvrapps-t2-2019/tree/Linh_DesignerBranch/Prototypes/ObjDetection_PhysicalMeasurement> |

**6. The application should be able to detect objects in the real-time and measure their size:**

|  |  |
| --- | --- |
| **Instruction** | 1. Set up the AR Core device and the plane visualization like other examples.  2. The principle of object detection used in this application is using the combination of computer vision and neural network model. More explanation could be found in Example 25 of the course note. Currently, the pool of detectable object is limited because the used neural network model only provides a few. All the computer vision functionalities are done via the library files Assets/Plugins/Android/libopencv\_java4.so and Assets/Plugins/Android /libVisualRecognition.so. We just need to put those two files to the exact path like the prototype.  3. One major change compared to the original example is that we are using AR Core now, and AR Core use the phone camera, so the phone camera cannot be used by *WebCamTexture* component anymore. Instead, I suggest another solution:  3a. Create a *Render Texture* named *ARRenderTex*. Set the size to 512x512.  3b. Open *AR Core Device* object, unpack prefab completely, then duplicate the *First Person Camera,* name the clone *RenderTextureCamera*. Set the depth of the second camera to 0 and the target texture to *ARRenderTex*. The idea is to use 1 camera to display the scene on the phone screen and another camera to render the scene into an image to pass to OpenCV for processing. Details could be found in Assets/Scripts/AccessOpenCV.cs.  4. The method of marking the boundary of detected object is also different from example 25:  4a. Create an image with a square frame on a transparent background. Set RectTransform anchor preset to bottom left. Set Width and Height to 100. Save it as a prefab (the *Frame* prefab in the prototype). Inside *Frame* create 2 text field, one at the top to display the type of detected object and one at the bottom to display the width of the object.  4b. The detection method from OpenCV does provide the information of the boundary by returning the position of the bottom left corner and the top right corner. We use those value to adjust the position and the scale of the Frame. More details could be found in a*ddVisual* function in Assets/Scripts/AccessOpenCV.cs  5. Object width is measured by the same method mentioned in requirement 5. First scan to detect the surface the object land on, then use AR Core to cast a ray from the bottom left and bottom right corner of the rectangle boundary, in order to locate the position of two corner points in the real world. Then calculate the distance between those two points. More details could be found in a*ddVisual* function in Assets/Scripts/AccessOpenCV.cs. |
| **Testing process** | 1. First scan the around surfaces.  2. Currently, the application could detect those object: "background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow", "diningtable", "dog", "horse", "motorbike", "person", "pottedplant","sheep", "sofa", "train", "tvmonitor". Put one of those to the scan surface, for example a botte like the image below:    3. If everything run correctly, you should see a black frame as object’s boundary, along with the type of object and the its width. |
| **Related Prototype:** | Object Detection and Physical Measurement, <https://github.com/deakinshaun/arvrapps-t2-2019/tree/Linh_DesignerBranch/Prototypes/ObjDetection_PhysicalMeasurement> |

**7. The application should be able to simulate light based on the real lighting condition:**

|  |  |
| --- | --- |
| **Instruction** | 1. Start with the scene from Requirement 4  2. Delete the default light in the scene. Add Assets/GoogleARCore/Prefabs/Environmental Light.prefab to the scene. |
| **Testing process** | To be defined. |
| **Related Prototype:** | To be defined |

**8. The user should be able to use the voice commands to control some functionalities:**

|  |  |
| --- | --- |
| **Instruction** | The principle of voice command used in this application is using the Speech Service REST API provided by Microsoft Azure. The detailed explanation could be found in Example 39 in the course note. However, I would like to mention some of the important changes to process the response string from server and to begin the recording; as well as clarify some unclear points in the original example:  1. First of all, unlike Google Speech API that run as background service which can record voice all the time, this application is designed to only record voice when there is a signal (in this case a phone shake). The record time used in the prototype is 2 seconds. I think 2 seconds is long enough to get all important short commands. Phone shake is detected by calculating the acceleration of the phone, if it over *shakeDetectionThreshold* (1.5f) then it is considered a shake. Also, a small interval is applied to prevent the detection of many shakes in one physical moving.  if ((Input.acceleration.magnitude >= shakeDetectionThreshold) && (Time.time >= timeFromLastShake + shakeDetectionInterval))  {  timeFromLastShake = Time.time;  //begin record audio  Trigger();  }  Also make sure there is a text to inform user that the recording has begun (for example “*Waiting for your command”)*  2. Make sure the game object controlling the voice recognition process has an Audio Source component to store the recorded voice.  3. A new subscription key is needed. Do not copy the same string *subscriptionKey* from the example. Follow the instruction in the example 39 to create a new key.  4. Change the http request type from *detailed* to *simple.* This change is for easier JSON convert.  HttpWebRequest request = (HttpWebRequest)WebRequest.Create(fetchUri + "?language=en-US&format=simple");  5. Use JsonUtility to convert JSON response string from server to separate components. We must define a class with exact field name as the JSON object.  [Serializable]  public class TextResponse  {  public string RecognitionStatus;  public string DisplayText;  public int Offset;  public int Duration;  }  Then retrieve the result text:  TextResponse textResponse = JsonUtility.FromJson<TextResponse>(responseString);  string result = textResponse.DisplayText;  Also make sure there is a text to display the result.  6. If the result text includes below string, the corresponding command will be executed:  - “Menu”: open menu for selecting furniture item (\*)  - “Surface”: toggle on/off the visualization of the plane grid (\*)  - “Detect”: toggle on/off the real-time object detection (\*)  - “Measure” : toggle on/off the physical measurement mode (\*)  (\*) All those features will be discussed in Requirement 10,11,12,13 |
| **Testing process** | 1. To test the shaking detection after step 1, just shake the phone to begin the recording, you should the text “Waiting for your command” showing.  2. To test if the microphone of the device works, say something when the text “waiting for your command” appear. After 2 seconds, there should be a playback of what you said. Check the microphone permission on the phone if problems occur.  3. To test if server return the recognition result, just look at the result text. Problems may occur if the key is expired, try to create another key. |
| **Related Prototype:** | Voice control , <https://github.com/deakinshaun/arvrapps-t2-2019/tree/Linh_DesignerBranch/Prototypes/VoiceControl> |

**III. Updated Schedule:**

***Trimester week 8:***

**Goals:**

For designer:

- Explain to the developer in detail every components of the training manual, as well as give the access to all the prototype applications that the designer has done before.

- Support the developer if he/she has problems following the instruction

For developer:

- Meet with the designer to know what component should be implemented and what platform/software/techniques should be used.

- Spend time to review all prototypes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Role** | **Task description** | **Related Requirement** | **Estimated time spent** | **Progress** |
| **Designer** | Explain and showcase all components in the training manual to the developer | n/a | 2 hours | Completed |
| Submit 3 prototype to GitHub | n/a | 0.5 hours | Completed |
| **Developer** | Discuss with the designer to understand all components of the final application | n/a | 2 hours | Completed |
| Review the source code of *AR object control* prototype | 1,2,3,4 | 2 hours | Completed |
| Review the source code of *Object Detection and Physical Measurement* prototype | 5,6 | 2 hours | Completed |
| Review the source code of *Voice Control* prototype | 7 | 2 hours | Completed |

***Trimester week 9:***

**Goals:**

For designer:

- Support the developer if he/she has problems following the instruction

- Check the progress and consider alternative approach if some features are hard to implement.

For developer:

- Start the implementation. These features should be completed and available to test: place multiple 3D objects into the scene, tracking their physical location and interact with the objects. The application should have a simple UI with options to choose the type of object. The application should be able to put many objects into a room and remember the physical location so when user point the camera elsewhere and point back, the objects still stay at the same position. User also could rotate and move every single object in the scene.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Role** | **Task description** | **Related Requirement** | **Estimated time spent** | **Progress** |
| **Designer** | Meet with developer to check the progress | n/a | 2 hours | Completed |
| **Developer** | Meet with designer to discuss about the progress | n/a | 2 hours | Completed |
| Use AR Core to display the real scene on the phone screen | 1 | 0.5 hour |  |
| Use AR Core to display detected surface and add collision to the surfaces | 1,3,4 | 1 hours |  |
| Add a red dot in the middle of the screen | 1,2,3,4 | 0.5 hour |  |
| Place virtual object to the scene by pressing a button | 1,2 | 2 hours |  |
| Select virtual object by pressing a button | 4 | 1 hours |  |
| Move object along with the phone camera | 4 | 1 hours |  |
| Rotate object by screen swipe gesture | 4 | 1 hours |  |

***Trimester week 10:***

**Goals:**

For designer:

- Support the developer if he/she has problems following the instruction

- Check the progress and consider alternative approach if some features are hard to implement.

For developer:

- Continue developing the application. These features should be completed and available to test: physical measurement between two point in space; real-time object detection and measurement.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Role** | **Task description** | **Related Requirement** | **Estimated time spent** | **Progress** |
| **Designer** | Meet with developer to check the progress | n/a | 2 hours |  |
| Finalize the training manual for submission | n/a | 1 hour |  |
| **Developer** | Meet with designer to discuss about the progress | n/a | 2 hours |  |
| Implement distance measurement between two points in the detected surface. User will able to select start point and endpoint. There will be a text on top of the screen to show the distance. | 5 | 1 hour |  |
| Display a line connecting the two points. | 5 | 0.5 hour |  |
| Implement object detection based on the provided lib files. | 6 | 2.5 hours |  |
| Display the boundary around the detected objects | 6 | 1 hour |  |
| Calculate the size of the detected objects | 5,6 | 1 hour |  |

***Trimester week 11:***

**Goals:**

For designer:

- Support the developer if he/she has problems following the instruction

- Check the progress and consider alternative approach if some features are hard to implement. Also, in this late stage, consider which features should be skipped if the progress is too slow.

- Prepare for the final demonstration

For developer:

- Complete the application. The voice control features should be completed and available to test. Also, polish the UI by complete requirement 8,9,10,11.

- Prepare for the final demonstration

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Role** | **Task description** | **Related Requirement** | **Estimated time spent** | **Progress** |
| **Designer** | Meet with developer to check the progress | n/a | 2 hours |  |
| Prepare the presentation for the final demonstration | n/a | 1 hour |  |
| **Developer** | Meet with designer to discuss about the progress | n/a | 2 hours |  |
| Implement the voice command feature | 7 | 2 hours |  |
| Implement the item menu for selecting many types of item | 8 | 1 hour |  |
| Implement the feature to toggle on/off the physical measurement mode. | 9 | 0.5 hour |  |
| Implement the feature to toggle on/off the object detection | 10 | 0.5 hour |  |
| Implement the feature to toggle on/off the plane grid | 11 | 0.5 hour |  |
| Prepare the presentation for the final demonstration | n/a | 1 hour |  |

**IV. Changelog:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Changes** | **Rationale** |
| V 1.0.0 (Initial) | 4/9/2019 | - Add the updated requirement from the case study  - Add the updated schedule from the case study  - Add detailed instruction for the requirements | Update all components based on the feedback from the client |
| V 1.0.1 | 11/9/2019 | - Update detailed instruction of how to set up the plane visualization component | In previous version, I just told the developer to take reference from HelloAR example of AR Core. It’s not detailed enough, so I decide to provide more instructions. |
| - Add 3 more UI requirements to toggle on/off some features | Based on the discussion with the developer. |