Touch Table Object Recognition

TTOR is a powerful Unity Asset designed to recognize and follow 3D printed objects on capacitive touchscreens. Whether you're working with iPads, Windows devices, or Android touchscreens, this cross-platform solution offers quick and stable detection of physical objects.

Download a demo with all example scenes and the calibrator here.
You can find the 3D files for the pucks here.

Last Update: Feb 5, 2024
Version: 1.0.1

Disclaimer

We’re not responsible for scratched screens! Make sure your 3D printed pucks have no sharp edges and don’t contain any particles from sanding before using them.

Working Principle

To understand the limitations of the asset, it's important to grasp the basic working principles.

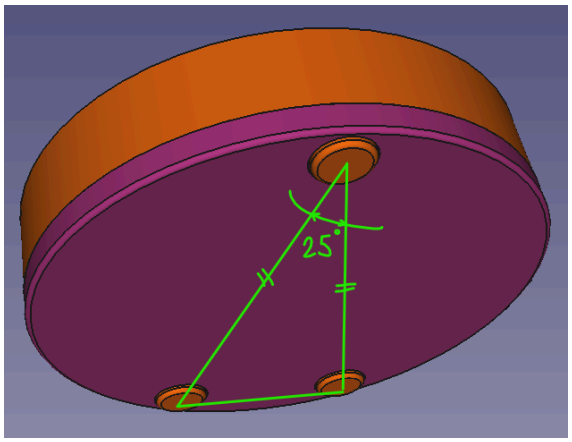
The asset detects an object by monitoring combinations of **3 touchpoints** that form an **isosceles** triangle. When 3 touchpoints create such a triangle, we calculate the **key angle** and **surface area** and compare them with a pre-calibrated list of "PuckData."
If a match is found, the asset reports the creation of a new "Puck."

The asset provides ready-made components to track the rotation and position of a specific puck, enabling the creation of engaging and dynamic content that serves as an eye-catcher and effectively showcases physical products.

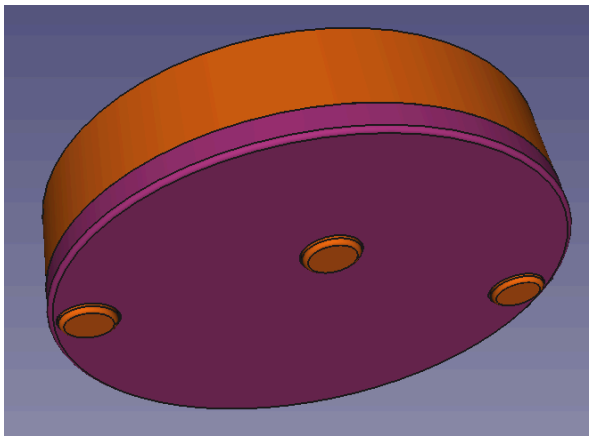
You can use 3D printing to produce these pucks, the base discs need to be printed with conductive filament. An insulator disk is affixed between the conductive legs to provide a polished appearance and ensure stability.

Optimal key angles fall between **25** and **135** degrees. **Angles close to 60 degrees are not suitable**, as determining the upright side becomes impossible.

Given variations among touchscreens and the need for tolerances, we recommend limiting the number of pucks to 12 for pucks with a diameter of 10cm. Smaller pucks might need larger tolerances resulting in fewer possible pucks.



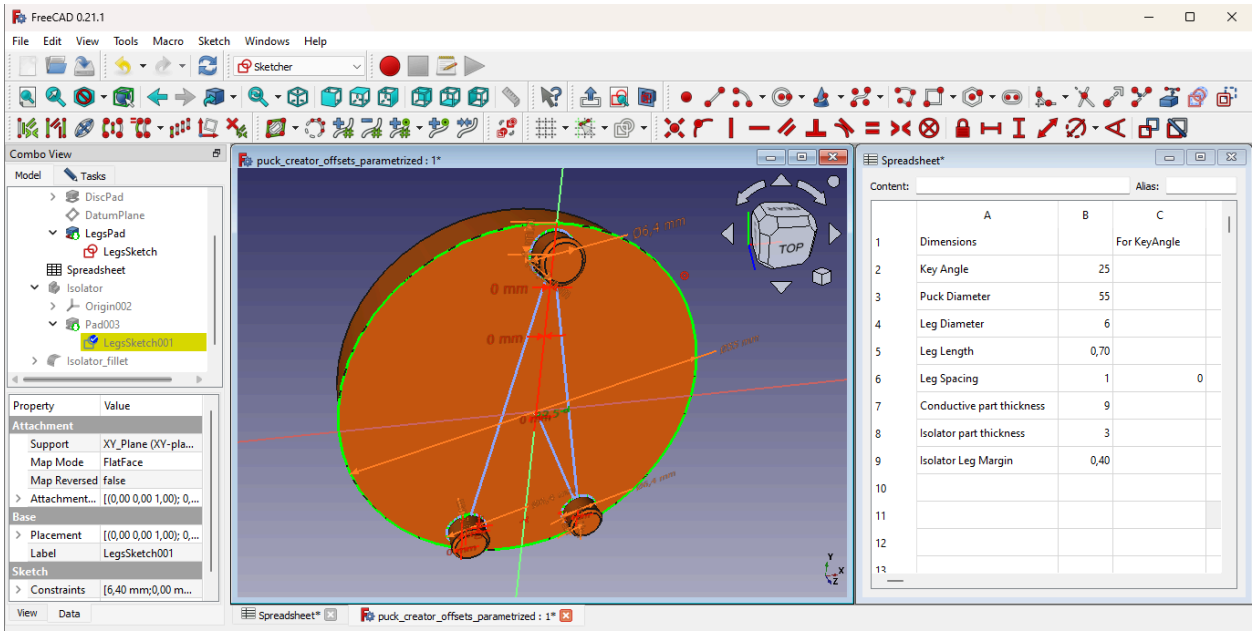
....



3D Printing the pucks

Premade STL files can be downloaded here for different diameters, but you may prefer to create your own designs.

To simplify this process, we provide a parametric 3D model using the open-source modeling software FreeCAD. It's as straightforward as adjusting the parameters to generate new STL files.



Conductive Filament

Use it to print the conductive part. Use normal (non conductive) filament to print the isolator model.

Conductive Filaflex is a TPU based filament that works great! It's the ideal amount of rubbery to not scratch but still slide smoothly over the screen. A small 50mm diameter and 3mm. thickness allows passive detection on an iPad Pro. https://recreus.com/gb/filaments/3-filaflex-conductivo.html

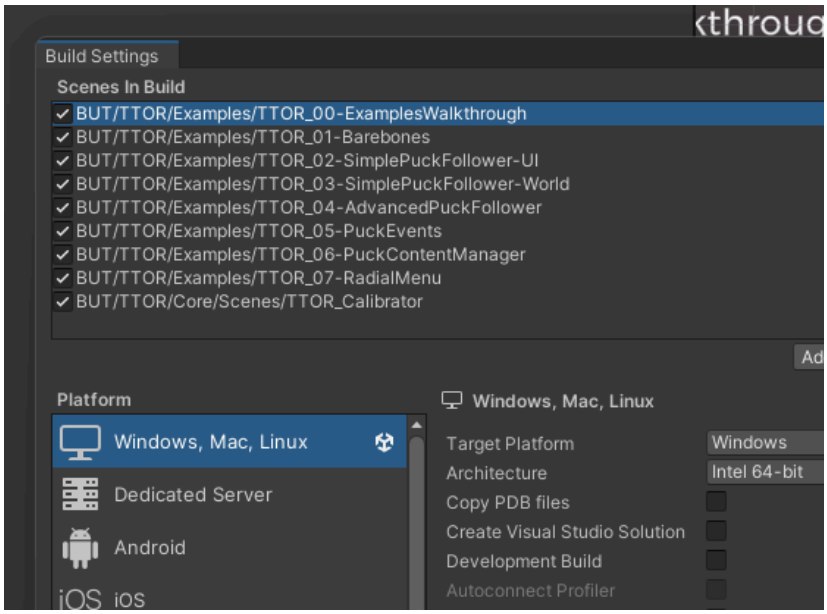
Protopasta's conductive PLA based filament works, but it's less conductive than the above. To have enough capacity to passively detect the puck (without your body acting as a giant capacitor) you have to make the conductive part of the puck at least 10 cm diameter and 4 mm thick. Also, PLA is pretty hard, be careful not to scratch your screen.
https://proto-pasta.com/products/conductive-pla

Getting started

Start by importing the Unity asset in your project.

We suggest you read through this document first and then take a look at the example scenes included.

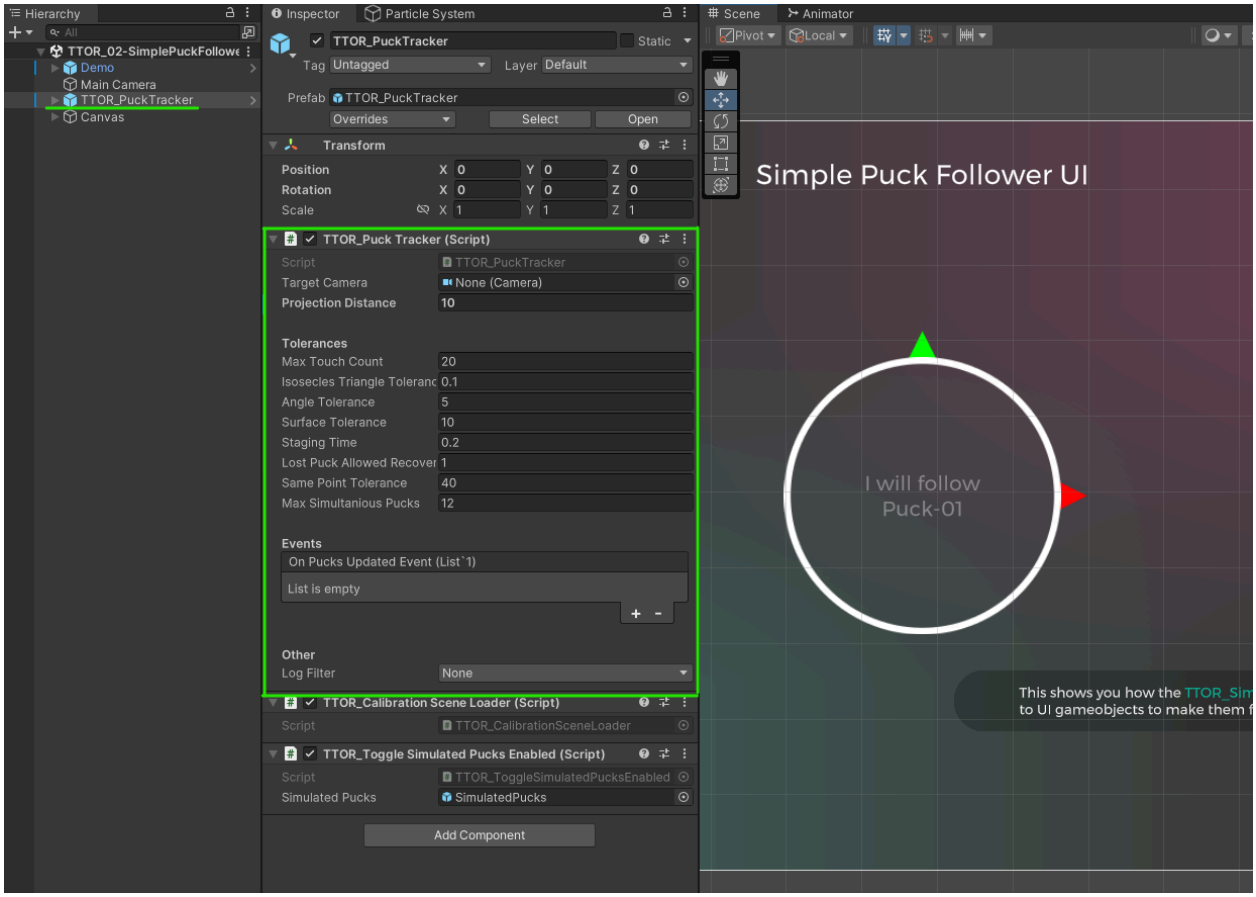
Add all example scenes and the calibrator to the build settings.



Unity Asset Code

TTOR_Pucktracker

To track pucks on your screen you will need to add a TTOR_Pucktracker component. We ‘ve provided a prefab with sensible defaults and everything to simulate pucks without the need of a touchscreen and physical prints.



The TTOR_Pucktracker component handles the loading of calibration data and the detection of the pucks. Once it detects any calibrated puck an event will be fired “OnPucksUpdatedEvent” that contains a list of found pucks. This event is essentially all you need to do anything with the pucks.

Let’s go over all properties:

TargetCamera: leave empty to default to Camera.main, this is the camera that will be used to calculate the world position of the pucks alongside with ProjectionDistance.

ProjectionDistance: The worldspace distance from the camera that will be used to position a puck from the camera.

MaxTouchCount: How many touchpoints we are dedicating to finding pucks. We need 3 touchpoints to detect 1 puck.

IsoseclesTriangleTolerance: The maximum percentage of length difference between two edges of a triangle before they are no longer considered the same.

AngleTolerance: The amount of degrees an angle can be larger or smaller to be considered the same as the pucks.

SurfaceTolerance: The amount of square cm a triangle can be larger or smaller to be considered the same as the pucks.

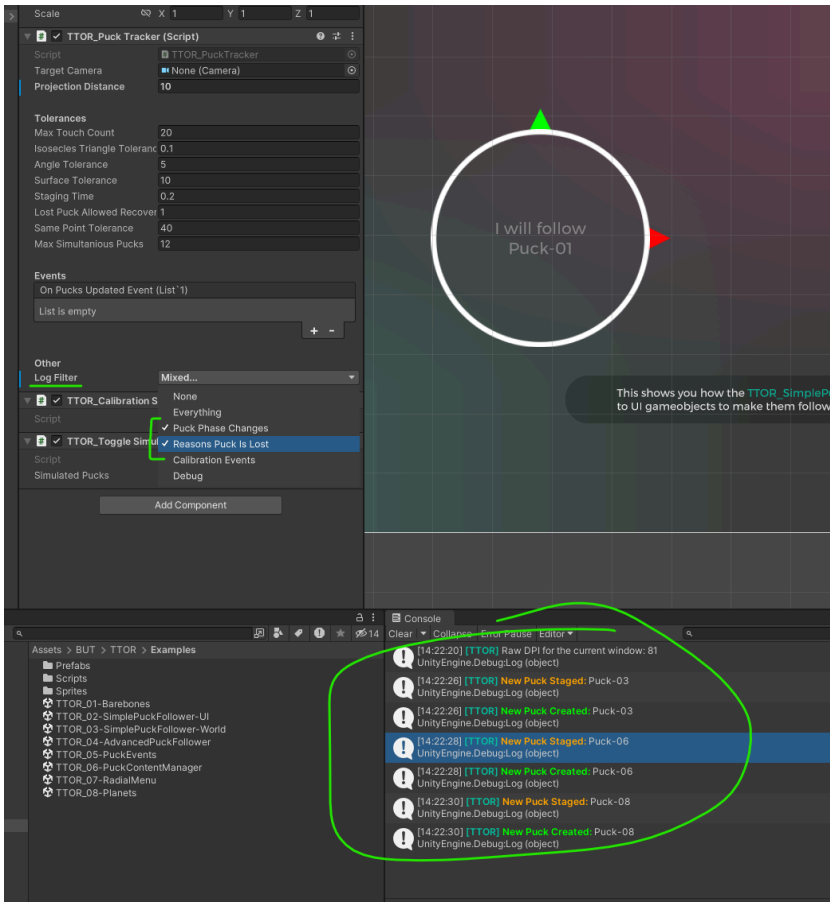
StagingTime: The time in seconds a puck needs to be present before it's created.

LostPuckAllowedRecoveryTime: The time in seconds a puck is allowed to be lost before it's removed

SamePointTolerance: The distance in pixels a point can be from another point to be considered the same point.

MaxSimultaniousPucks: The maximum number of pucks that can be on the screen at the same time.

The LogFilter allows you to limit the editor debug logs that are shown to certain categories. The below screenshot shows how this works.



Here's a quick code example of how we would use the OnPucksUpdatedEvent event from TTOR_Pucktracker to track a puck's position on the screen.

(loosely copied from the TTOR_SimplePuckFollower)

```
private void OnPucksUpdated(List<Puck> pucksList)
{
   …

    foreach (Puck puck in pucksList)
    {
        if (PucksToFollow.Count == 0 || PucksToFollow.Contains(puck.Data))
        {
          transform.position = puck.GetPosition(_tracker.ProjectionDistance);
          transform.rotation = puck.GetRotation();
        }

        return;
    }
}
```

Puck

As you can see in the above code, OnPucksUpdated from the TTOR_Pucktracker's OnPucksUpdatedEvent event passes a list of Pucks. This Puck object is a unique instance each time a physical 3D puck has been put down. If a Puck is temporarily Lost and then found again you will still have the same instance of Puck to work with.

Once a puck has been removed (by lifting it off the screen longer than the LostPuckAllowedRecoveryTime) and put back down, you'll receive a new instance.

Puck has 3 important methods:

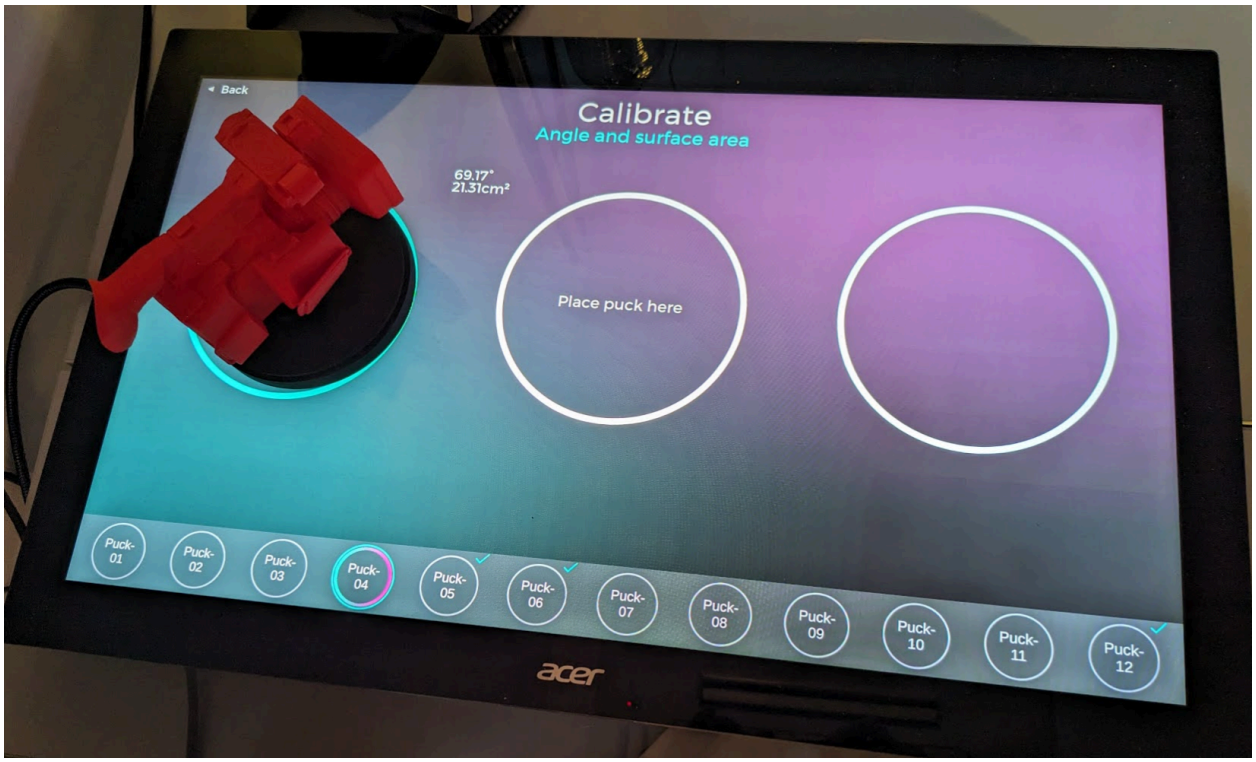
GetPosition: requires you to pass the distance from the camera to return a worldSpace position of the puck.

GetScreenPosition: returns the screen position of the Puck.

GetRotation: returns the rotation of the Puck.

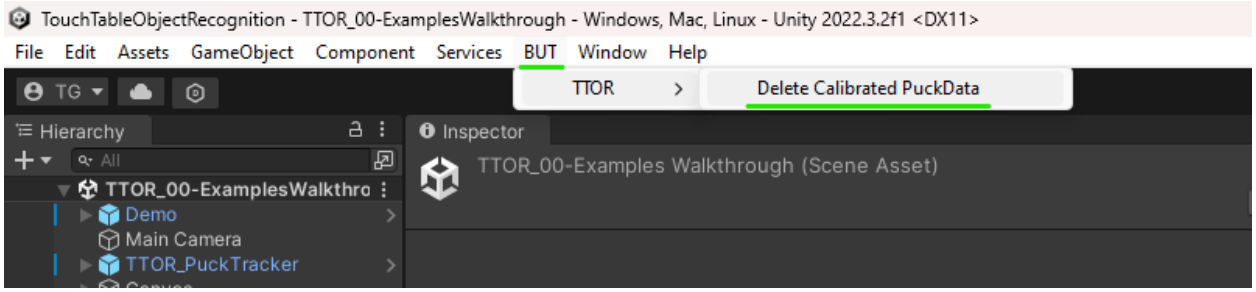
TTOR_CalibrationSceneLoader

This code loads the calibration scene when the shortcut Ctrl/Cmd+C is pressed on the keyboard.

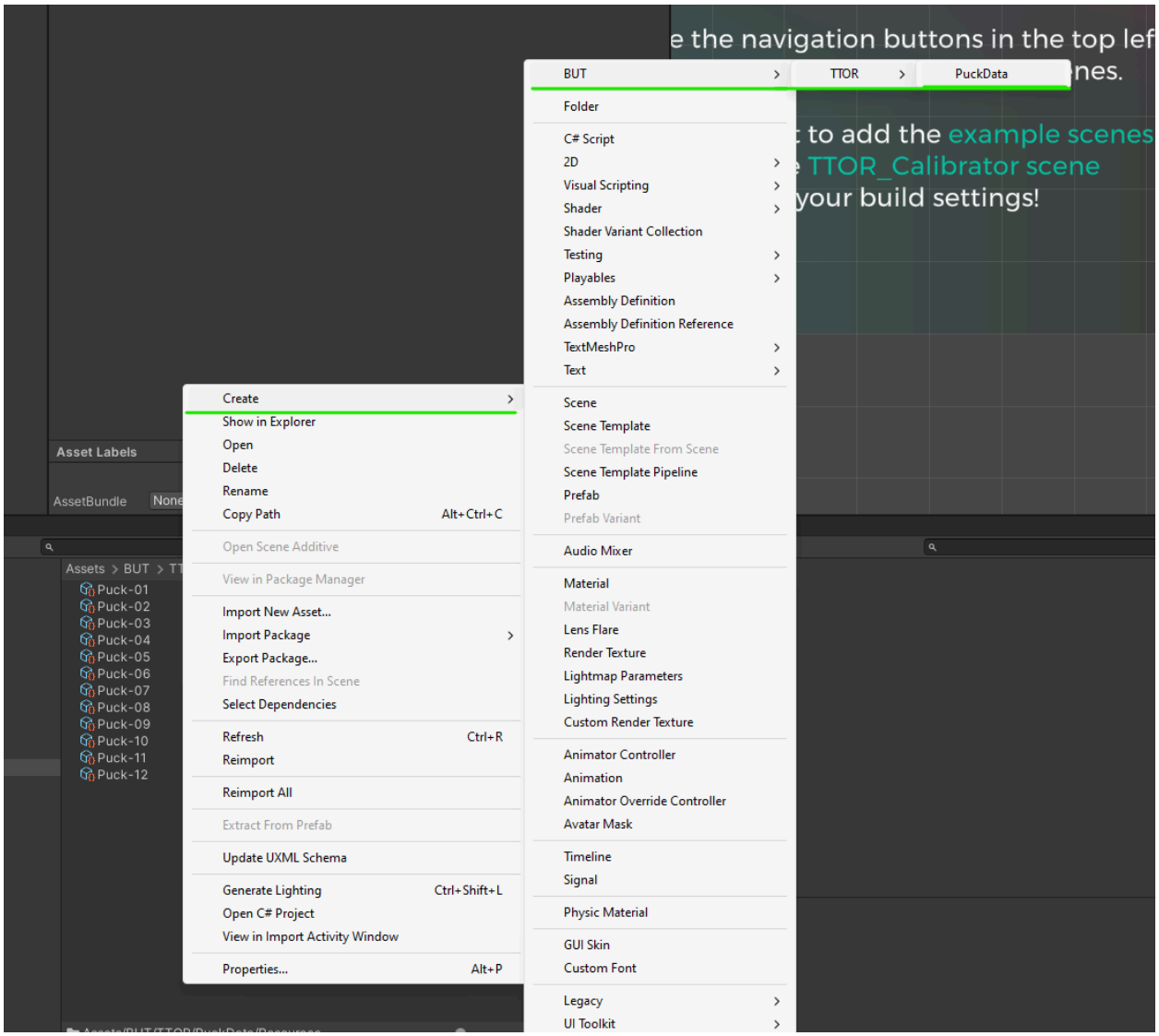


This picture shows the calibration scene. Our pucks are numbered up to 12 by default but renaming the PuckData files will show the names that are tailored to your project. (It could have been named “Camera”)

Notice the checkmarks on pucks 5,6 and 12. This means they have calibration data. Selecting them allows you to delete the data and to calibrate them again.



If you want to quickly delete all calibration data on your development machine, you can use the above menu in Unity.

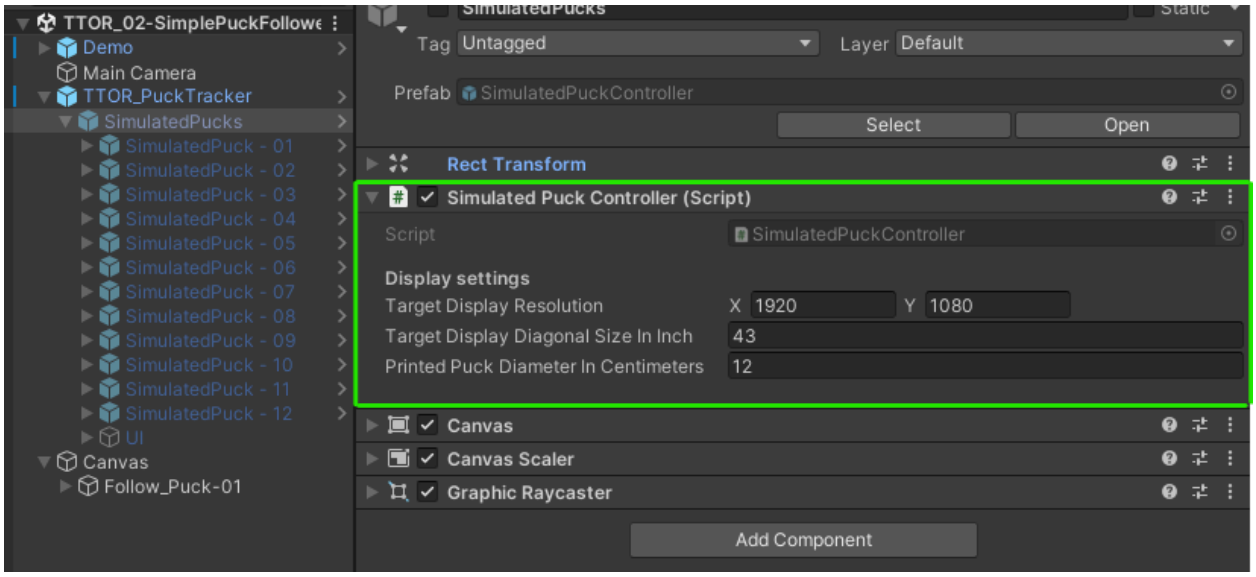


To create additional PuckData (these define your physical objects) you can use the menu. Best to name your PuckData file something that relates to the physical object. The name of the PuckData file will be featured in the Calibrator.

Notice the provided PuckData files reside in a folder called “Resources” this is required. The PuckTracker will look for all PuckData files in the whole project but only if it’s inside a Resources folder.

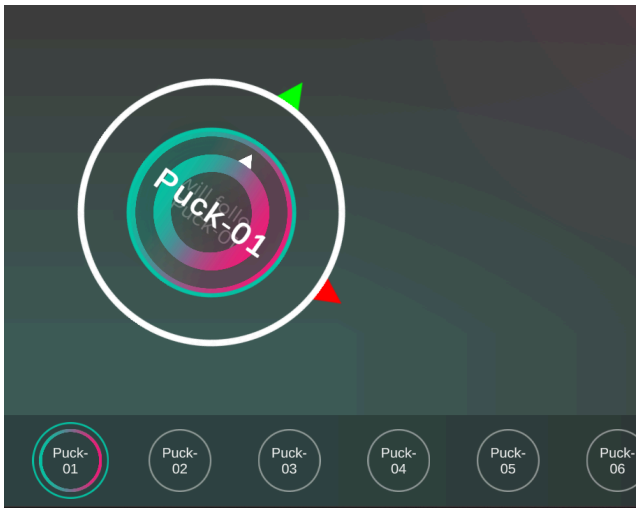
TTOR_ToggleSimulatedPucksEnabled, SimulatedPuckController, SimulatedPuck

TTOR_ToggleSimulatedPucksEnabled starts the Puck Simulator when pressing Ctrl/Cmd+X.
The prefab for the PuckTracker contains all assets to launch the simulator.



The SimulatedPuckController requires information about the type of touchscreen you want to emulate for accurate scaling of the simulated pucks.

The simulator enables rapid iteration within the Unity editor without the need for an attached touchscreen. After pressing Ctrl/Cmd+X, you can click a puck in the bottom bar to simulate placing it on the screen. You can move it by dragging with touch or mouse controls and rotate it using the mouse scroll wheel or a two-finger drag.

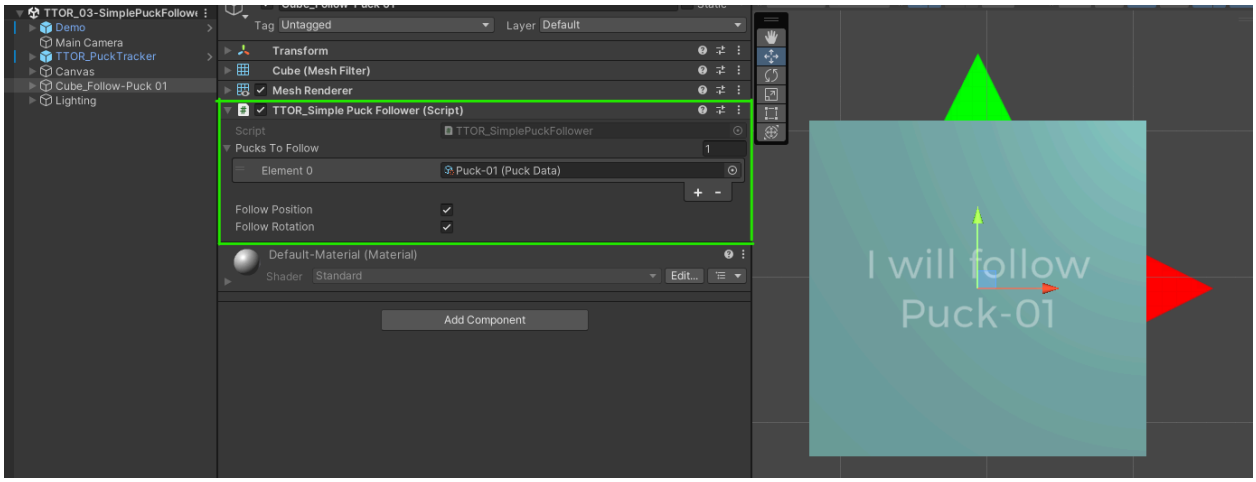


TTOR_SimplePuckFollower

Adding this component to a Gameobject will make it follow a Puck. It works for both UI RectTransforms and Regular Transforms.

You add a list of PuckData objects to define which pucks it should follow. If you leave the list empty it will follow all pucks.

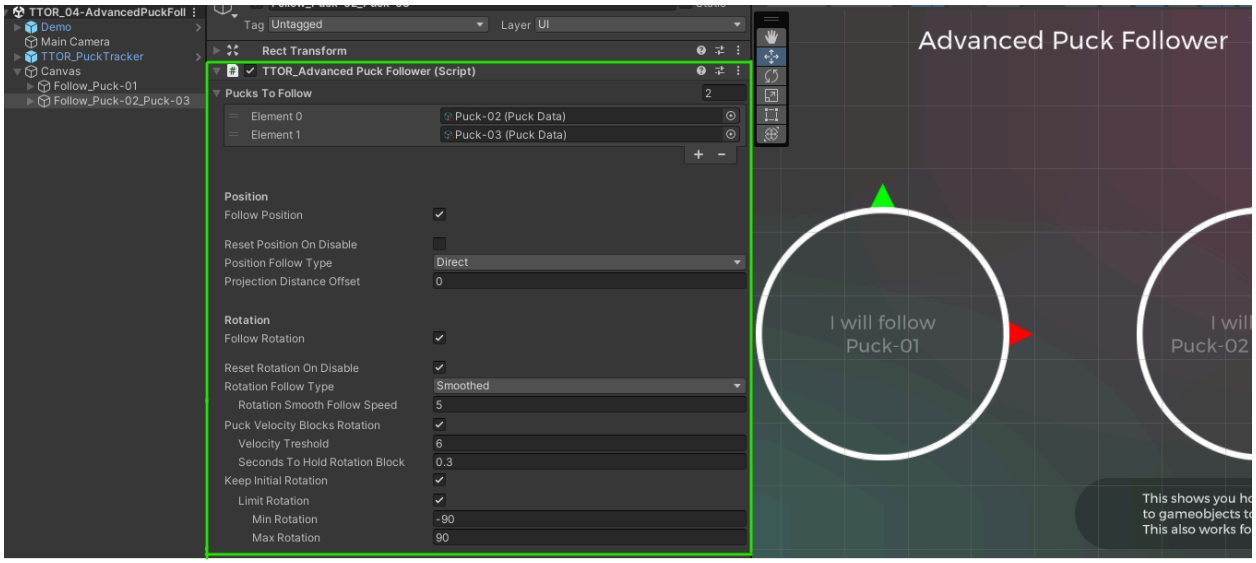
Additionally, this component implements the TTOR_IPuckContent interface. When the AssignPuck method is called passing a Puck object in the process. The TTOR_SimplePuckFollower will only follow that specific puck until its destruction.



See example: TTOR_02-SimplePuckFollower-UI

TTOR_AdvancedPuckFollower

Enables you to follow a Puck with more advanced settings. Just as its simple version, this also implements TTOR_IPuckContent.

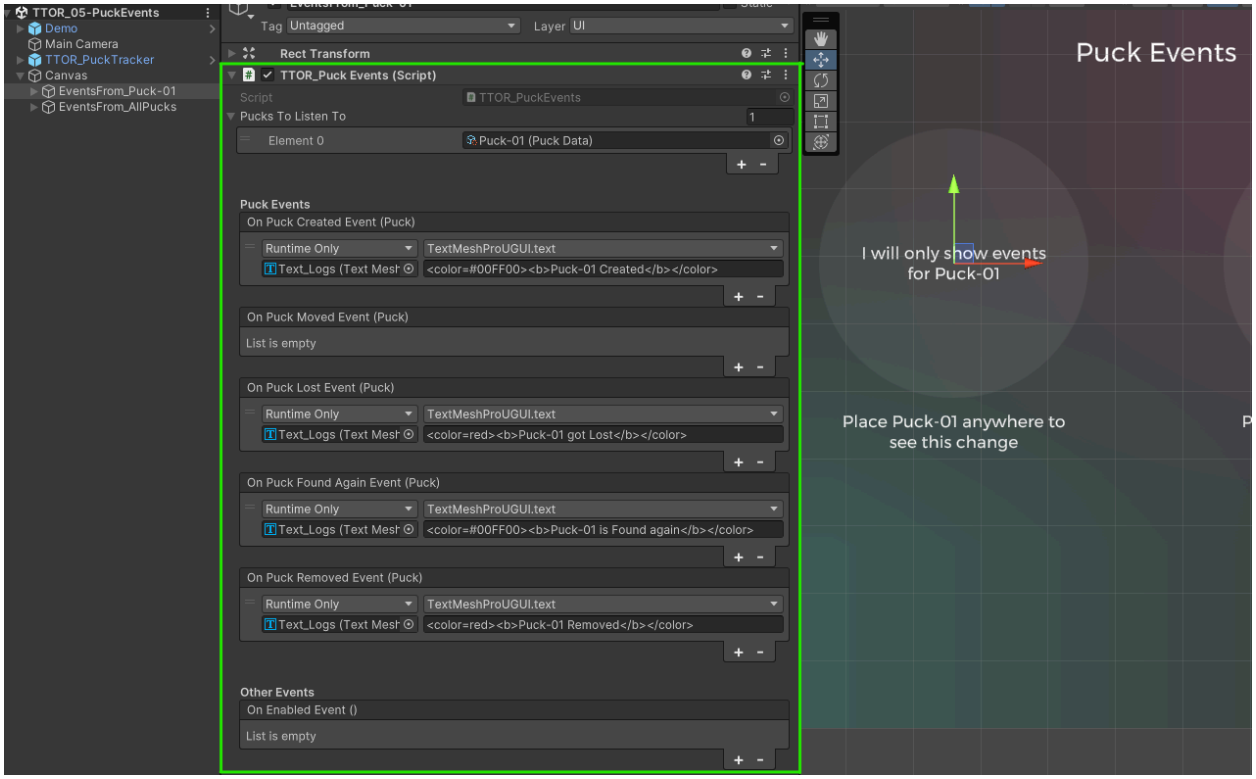


See example TTOR_03-SimplePuckFollower-World

TTOR_PuckEvents

Instead of listening to the TTOR_PuckTracker's OnPucksUpdated event directly you can use this component. Use the PucksToListenTo property to define the types of puck(s) that will invoke these events.

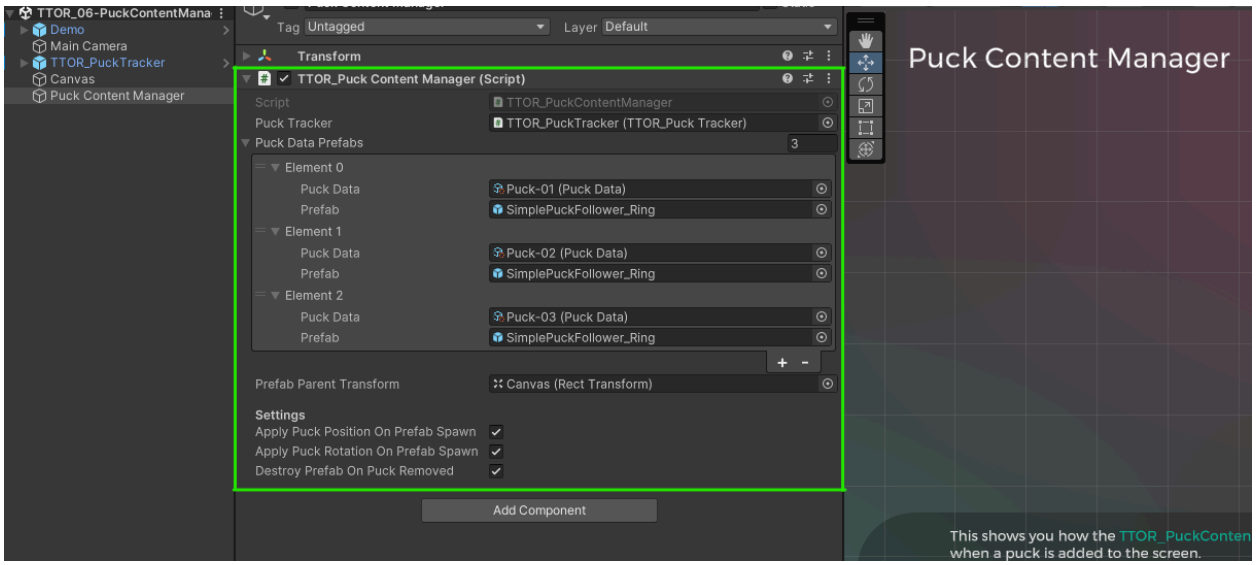
This component implements the TTOR_IPuckContent interface. When the AssignPuck method is called passing a Puck object in the process. The TTOR_PuckEvents will only invoke events coming from that specific puck until its destruction.



TTOR_PuckContentManager

This component will instantiate a prefab in your scene depending on the type of puck you put down. Additionally, it can place the newly created instance at the position and rotation of where you put the puck down on the screen. It can also remove the instance once the puck has been removed.

Define where in the scene's hierarchy you want your new instance to be created by providing a Transform to the "prefabParentTransform" property. Leave it empty to spawn on the root.



Once a prefab instance has been created by the TTOR_PuckContentManager, it will search for components that implement the TTOR_IPuckContent Interface. These components will have their AssignPuck and RemovePuck methods called, passing the Puck responsible for their creation to the respective methods.

Interface TTOR_IPuckContent

This interface is implemented by TTOR_SimplePuckFollower, TTOR_AdvancedPuckFollower and TTOR_PuckEvents. This forces these components to implement the AssignPuck and RemovePuck methods.

Contact

We've spent a lot of time perfecting the tracking and making the API as easy to use as possible. We hope you value our hard work by creating amazing experiences with it.

Should you have an issue or a good idea for a feature please contact us by e-mail

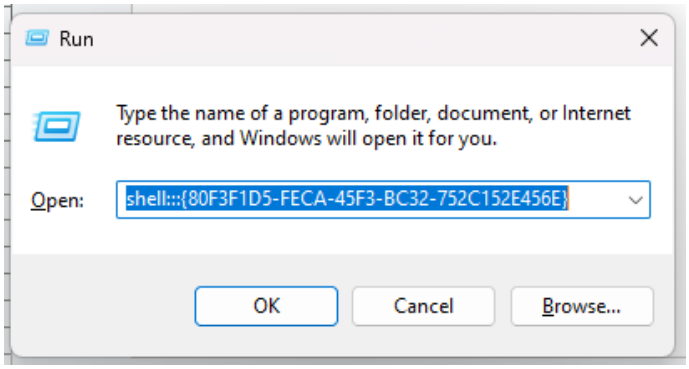
dev@but.be

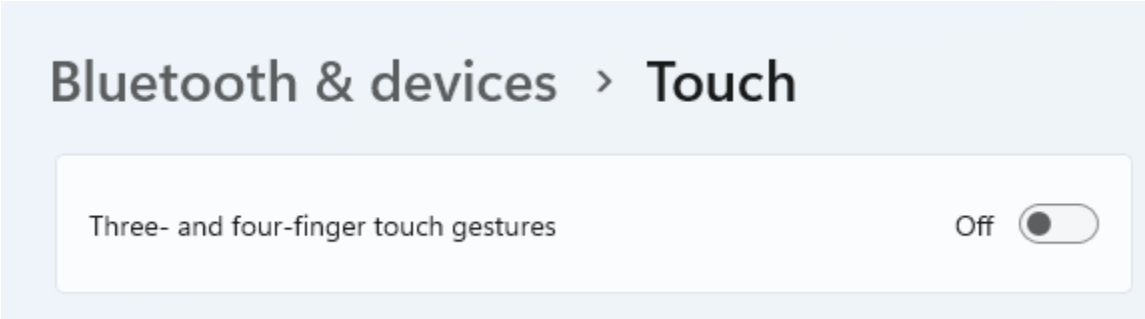
Download a demo with all example scenes and the calibrator here.
You can find the 3D files for the pucks here.

Knowledge base

Using windows? To quickly show the native touch calibration of the OS use this code in the "run"

shell:::{80F3F1D5-FECA-45F3-BC32-752C152E456E}



To disable the windows 11 screen edge swipe, use the link below to download the .reg file:
https://www.elevenforum.com/t/enable-or-disable-screen-edge-swipe-in-windows-11.3717/#Two
Save the .reg file to your desktop.
Double click/tap on the downloaded .reg file to merge it.
When prompted, click/tap on Run, Yes (UAC), Yes, and OK to approve the merge.
Either restart the explorer process, sign out and sign in, or restart the computer to apply.
You could now delete the downloaded .reg file if you like.

Under Start > Settings > Bluetooth & devices > Touch > Three- and four-finger touch gestures make sure it’s turned off.