MArC ReadMe

Masterprojekt Lukas Kolhagen, Vera Brockmeyer, Laura Anger, Paul Berning

Starting the System

The following software is needed:

- IDS uEye Driver and SDK (https://de.ids-imaging.com/download-ueye-win32.html)
- Steam VR (http://store.steampowered.com/about/)
- Leap Driver (https://developer.leapmotion.com/windows-vr)
- The OpenCV Framework (http://opencv.org/downloads.html)

In order to start the system, make sure the following requirements are met:

- 2 Computers are available:
    - one that runs the IDS uEye tracking application [A] and
    - one for the Unity application [B]
- The HTC Vive head-mounted display has been connected to computer [B]
- The Leap Motion controller has been connected to computer [B]
- The IDS uEye software suite has been installed on computer [A]
- Computers [A] and [B] are connected via a network cable
- The IP address of the ethernet adapter of computer [A] has been set to 192.168.0.7 with the standard subnet mask of 255.255.255.0
- The IP address of the ethernet adapter of computer [B] has been set to 192.168.0.13 with the standard subnet mask of 255.255.255.0
- It is necessary to use the uEye camera with another computer than the rendering, because the performance could be worse while the computer renders the scene and does the tracking at the same time.

The start procedure is as follows:

- Start the application [Masterprojekt.exe] located in the [MArC_Tracking] directory on computer [A]
- Make sure the opened command window indicates that the server socket on port 10000 is set to listen
- Start the Unity build [MArC.exe] located in the [MArC_Unity] directory on computer [B]
- Follow the instructions on the screen of computer [B]
- If there is an offset between the table plane and the leap hand model, place the right hand onto the table and press the trigger button of the Vive Controller. This eliminates the height offset

Calibrating the System

- After starting the application on computer [B], the user will be asked whether he wants to calibrate the system
- The user can choose between calibrating “camera and workspace” or only the “workspace”

1) Calibration of the uEye camera:

- NOTE: It is explicitly recommended to calibrate the camera only when it is needed!
- Once chosen, a camera window will be opened on computer [A]. The following steps can be controlled on this computer, while computer [B] will be activated again, when it comes to the calibration of the workspace
- Use the elevated chessboard to calibrate the system
- Follow the instructions on the console of computer [A] and press “g”
- The system will do 25 captures automatically and the user will hear a beep after each capture. (Taking the first 15 captures, while placing the elevated chessboard on the desk and the last 10 in a height of 20 cm leads to the best results)
- After a longer beep (signal that the user has finished the capturIng. Process) the calculation of the intrinsic camera parameters and distortion coefficients is made and a text file will be saved
- This text file allows the user to use the system without making a new camera calibration by loading the parameters directly from the text file
- After the calculation is completed, the calibration of the workspace can be done (see 2) Calibration of the workspace)

2) Calibration of the workspace:

- Follow the instructions displayed on computer [B]
- Use the HTC Vive controller with the attached Aruco Marker (ID 49, dictionary: DICT_4X4_50) and press the button as described
- 25 captures are necessary (each capture is confirmed by a haptic impulse)
- NOTE: The order of the first four captures is important!
- The user have to make sure to form up a rectangle which is equal to his desired workspace. Starting in the lower left corner and going clockwise
- it is recommended to place the controller on different positions on the table for the next 11 captures as well
- the last 10 captures can be done lifting the controller up to 20 cm and vary its position
- After capturing all positions follow the instructions on computer [B]

Glossary of recurring terms:

Marker Cubes:

- There are two types of marker cubes: real ones that the user can touch and interact with and virtual ones that are rendered at the positions of the real world marker cubes

Context Menu:

- Every (virtual) marker cube has a menu that lets you scale the building associated with the marker and offers context information on the building (such as the living area or the number of floors). This is called the context menu.

Table Menu:

- On the right side of the workspace is a menu dedicated to saving and loading scenes. This is called the table menu.

Match Mode:

- When loading a previously saved scene, the match mode is entered. This mode is necessary to ensure that each virtual marker cube can be associated with a marker cube in the real world. In order to connect a virtual marker with a real one, position the real marker cube so that its position coincides with the virtual marker cube that is pulsating in red. As soon as the color of the virtual marker turns to green, the position is close enough and you may move on to the next marker (if there are more). Once this has been done for all markers the “Apply”-button appears on the table menu, which lets the user exit the match mode.

File overview of Unity assets:

1. Plug-In assets

/Assets/LeapMotion	Leap Motion core assets
/Assets/SteamVR	SteamVR core functionality necessary to run the HTC Vive as part of the Unity simulation

2. Script assets

/Assets/ObjectMenu/Scripts

ContextMenu.cs	Fills the object menu canvas with informations
contextMenuTrigger.cs	Handles the user input for the context menu
markerScale.cs	Changes the size of the cube when the context menu handlers are moved
MatchMode.cs	Controls the MatchMode of the markers
MatchModeReady.cs	Is attached to to markers at the MatchMode, tells the system if a tcp-controlled marker is set correctly onto a loaded marker position
StopMatchMode.cs	Tells the system the status of the 4 triggers inside every cube while the MatchMode is running

/Assets/Skripts/

ControllerPos.cs	Handles the Vive controller button input
Marker.cs	Getter and setter for the markers
OutputNormalizedControllerPos.cs	Calculates the normalized position of the Vive controller
printFPS.cs	Debug script, shows FPS on an GUI
readInNetworkData.cs	Handles the incoming TCP datastream

setupScene.cs	Dynamic scene setup based on the calibration data
TableCalibration.cs	Calibrates the table in unity
DataHandler.cs	Handles the data copy process to tcp-controlled markers during the MatchMode
LeapHandCalib.cs	If there is an offset between the leap tracked hand and the plane, the offset is removed by this script

/Assets/Menus/

	Contains the StartMenu GUI scenes
/scripts/	Contains scripts for the startMenu GUI

/Assets/TableMenu/scripts

FileBrowser.cs	Contains functions for loading files
open.cs	Loads scenes from the saved xml documents
save.cs	Saves scenes as XML documents in /Resources/saves/
Timeline.cs	Handles the filebrowser (“timeline”) in the table menu
TableMenuTrigger.cs	Handles the user input for the table menu

File overview Tracking:

1. External Sources

OpenCV Library	Image Processing Library
Aruco Library	part of OpenCV Library
uEye SDK	SDK runs, actuates and configures the uEye Camera
TCP Lib	Winsock

2. Framework

Classes

TCP.cpp	configures and runs TCP data connection, normalizes the marker pixel coordinates in order to the workspace size
uEye_input.cpp	configures the uEye parameters and runs the capturing
PoseEstimation.cpp	runs the OpenCV camera calibration and saves or loads the camera matrix and the distortion coefficients files
PlaneAndAffineCalibration.cpp	generates the workspace in picture space; calculates affine transformation from uEye camera space and unity world
CoordsTransformation2Unity.cpp	transforms marker center in uEye camera space to unity with affine Transform mat
Calibration.cpp	offers access to the Camera Calibration and Plane Calibration, offers all required attributes of both calibrations
MarkerManagement.cpp	organizes the current Markers and actualises the current position and angle; either matches recently detected Markers with current Markers or register new Marker; delete Markers if they out of workspace
MarkerDetection.cpp	detects the green rectangles and the

	aruco markers
IdMapping.cpp	offers all requested functions to compute matches of recently detected Markers and current Markers
Marker.cpp	Objectclass with all marker attributes
CreateCharucoBoard.cpp	generates CharucoBoard for Camera Calibration
ArucoCodeGenerator.cpp	generates new Aruco Codes to print