University of Trento

Department of industrial engineering
Computer Vision course report

Professor: Conci Nicola **Tutor:** Garau Nicola

Student: Strobbe Cristiano **Esse3:** 196763 **A.Y.** 2018-2019
Mail: cristiano.strobbe@studenti.unitn.it

Stereo vision simulator for driverless FSAE vehicles

Introduction

The formula SAE competition is an international student racing competition created by SAE (Society of Automotive Engineer). The goal of the competition is to design and build electric/combustion/driverless vehicles which race together all over the world. The race is usually divided into static and dynamic events. During the first one the judges evaluate the vehicle from the design point of view while the second one is a series o dynamical test such as acceleration, endurance, skidpad and autocross. All the tracks are different and designed by the competition's judges. Usually the tracks are large 3-4 meters and they are composed by short straight parts ($\sim$ 40-70 meters) and many corners with very low turning radius parts ($\sim$ 5 meters). The track boundaries are marked by means of yellow and blue cones. Usually the blue ones are used for the left track boundary. The following figure 1.1 shows part of a typical endurance track. As shown in the previous figures the environment is quite poor and it is composed by: track surface, cones and the surrounding landscape (track buildings and trees).

Formula SAE driverless cars are equipped with many sensors such as GPS, IMUs, Lidar and Cameras. These sensors are used in order to acquire vehicle's pose (IMU-GPS), estimate position of the cones (Cameras-Lidar) or to generate the 3D map of the environment (SLAM).
Stereo cameras are widely used in the competition in order to estimate cone-position and map the environment. Usually they are mounted on top of the main-hoop which is the highest part of the car. The following figure 1.2 shows a frame taken from the right camera of the ETH Zürig stereo system.

Most of the stereo setup are composed by two monochrome/RGB low-resolution cameras which baseline goes from 100mm to 250mm. All the extrinsic/intrinsic cameras parameters depends on the design choices of each team.

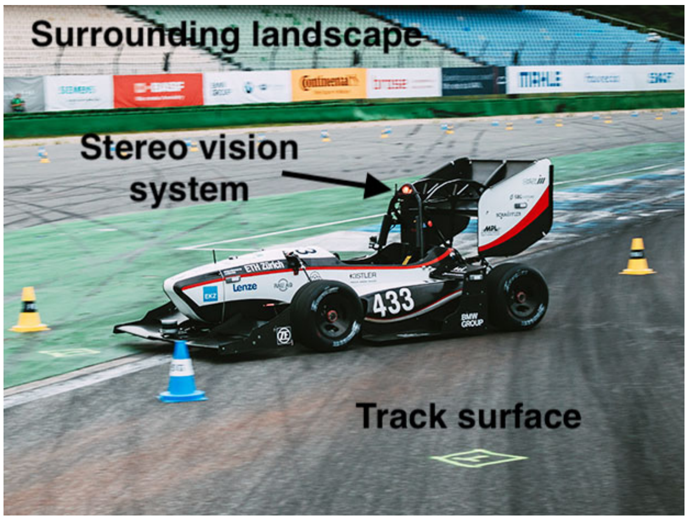


Figure 1.1: Formula SAE Germany Driverless track 2019

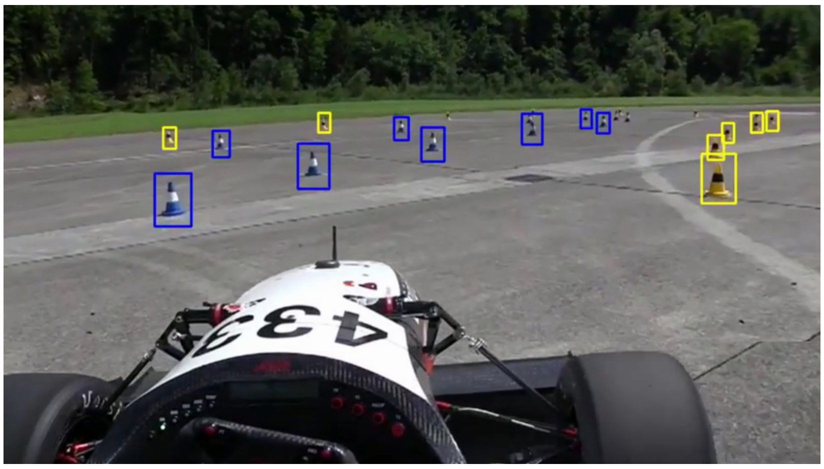


Figure 1.2: ETH Zürig driverless car view. Right frame FLIR (f:20mm - FoV:55°)

The goal of my project was to build a 3D simulation environment capable of model and simulate the stereo-vision system of a SAE driverless car. The simulation can be also used to create testing video/data with their ground truth. The last point became very helpful in case of Machine Learning based algorithm (for example cone detection) because allow the user to create verified training and testing dataset.
The main goal of my project can be summarised as follow:

- Create a 3D environment similar to the competition one
- Be able to set intrinsic/extrinsic stereo parameters
- Be able to acquire stereo cameras (RGB/Mono)
- Be able to acquire depth informations (disparity map)
- Be able to acquire cones/objects informations (position and color)
- Be able to save ground truth data (car telemetry)

Simulation environment

I developed my project using *Unity 2019* for the 3D simulation environment and *OpenCVforUnity* package for the stereo vision part.
The following figure 1.3 shows the Unity scene that I developed.

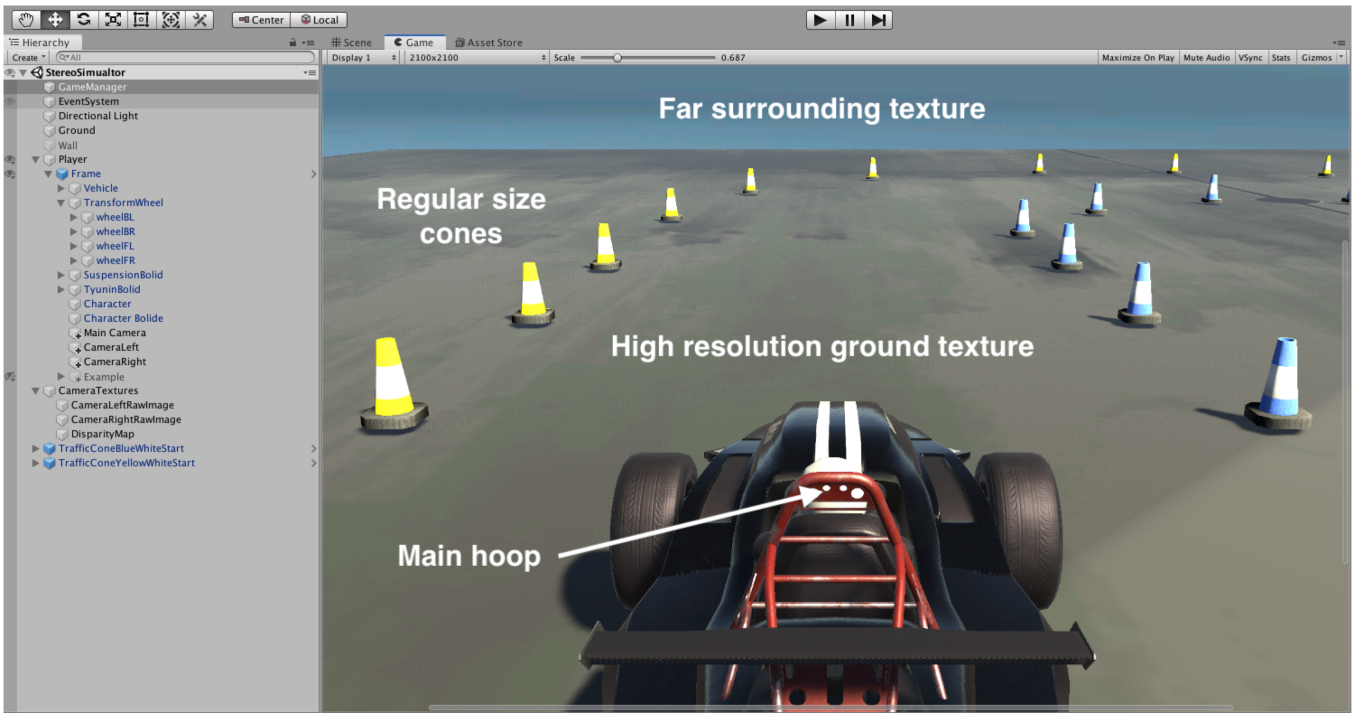


Figure 1.3: Scene

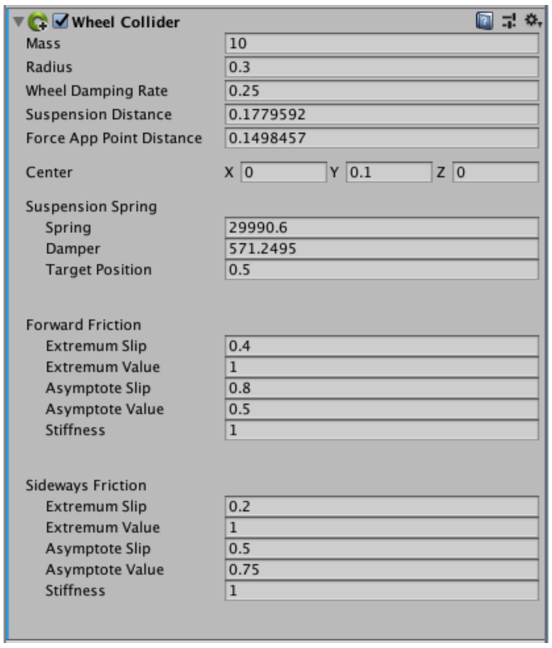
As shown in the previous figure I build the scene paying attention to model only the necessary parts so: ground texture, cones shape/color and landscape texture. For both ground and landscape I choose high resolution textures.

Vehicle model

Wheel model

I choose a vehicle very similar to the one of the formula SAE. My goal was to make its physical behaviour as close as possible to a real one. In particular I decided to model the suspension system because it influences the position and the orientatation of the stereo system. For this part I used the physical proprieties and interactions of the road surface collider and the *wheel collider* figure 1.4.

As shown in figure 1.4 there are many wheel physical proprieties that can be set. For my purposes I set the wheel mass, radius, damping ratio, forward slip and sideway slip. These are the main parameters that are usually used in order to create a linear tyre model. More complicated tyre can be model. For example, using the wheel damping rate the user can model the tyre's rubber damping ratio or by changing the



Wheel Collider

Mass	10
Radius	0.3
Wheel Damping Rate	0.25
Suspension Distance	0.1779592
Force App Point Distance	0.1498457
Center	X 0 Y 0.1 Z 0
Suspension Spring	
Spring	29990.6
Damper	571.2495
Target Position	0.5
Forward Friction	
Extremum Slip	0.4
Extremum Value	1
Asymptote Slip	0.8
Asymptote Value	0.5
Stiffness	1
Sideways Friction	
Extremum Slip	0.2
Extremum Value	1
Asymptote Slip	0.5
Asymptote Value	0.75
Stiffness	1

Figure 1.4: Wheel collider proprieties

values extremum and asymptote slip the user can also implement non-Linear tyre friction models, figure 1.5.

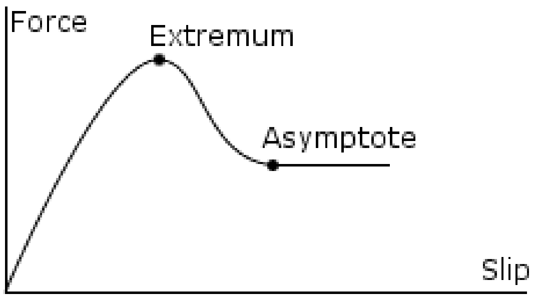
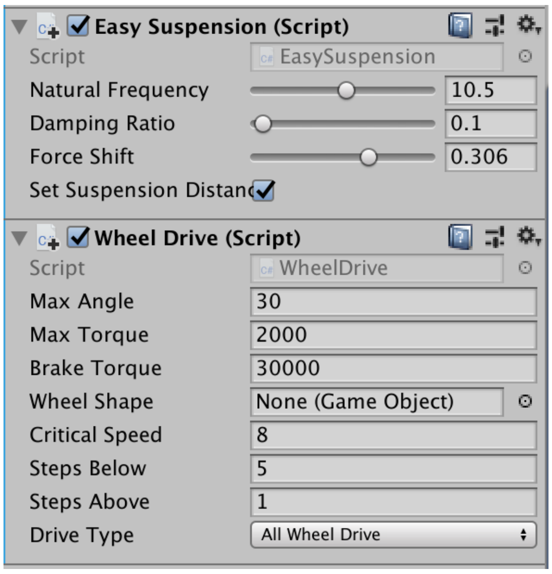


Figure 1.5: Vertical load - Forward friction coefficient general curve - Source: *Unity - Manual*

Another feature of the wheel collider is that it implements: motorTorque, brakeTorque and steerAngle functions. Many vehicle control system (ABS, ESP and Torque Vectoring) acts directly on wheel's torque and steering angle so the implementation of these control system inside the simulation became easier.

Suspension model

In order to simulate the suspension system I decided to implement a simple 1 DoF spring-damper system that can be attached to each wheel. This script allows me to model the typical pitch and roll motion of the vehicle. The following figure 1.6 shows the suspension script and the wheel drive parameters. As shown in figure 1.6 the user can set three parameters: natural frequency ($\omega_n = \sqrt{k_{spring}/m_{wheel}}$), damping ratio ($\zeta = c/c_{critical}$ where $c_{critical} = \sqrt{k_{spring}m_{wheel}}$) and force shift (phase between force and displacement). These three parameters can be then used to compute spring stiffness (k) and damper value (c).


Easy Suspension (Script)

Script	EasySuspension
Natural Frequency	10.5
Damping Ratio	0.1
Force Shift	0.306
Set Suspension Distanc	✓

Wheel Drive (Script)

Script	WheelDrive
Max Angle	30
Max Torque	2000
Brake Torque	30000
Wheel Shape	None (Game Object)
Critical Speed	8
Steps Below	5
Steps Above	1
Drive Type	All Wheel Drive

Figure 1.6: Easy suspension configuration script

The wheel drive script allow the user to set the vehicle type between FWD (front rear drive), RWD (rear wheel drive) or 4WD (four wheel drive). Moreover is possible to set the maximum wheel's steering angle and the maximum traction and braking torque that can applied to each wheel.

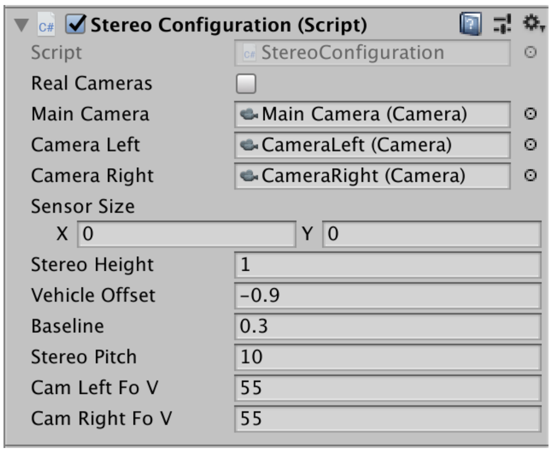
Stereo system configuration

Intrinsic/extrinsic parameters

For each RGB/Mono camera based stereo vision system there exist two type of parameters that are used in order to model it, the intrinsic and the extrinsic one. The first ones define the sensor/lens parameters (camera matrix $[A]$) while the second one are referred to the the relative position/orientation between cameras (roto-traslation matrix $[R|t]$). The following equation 1.1 represents the general mathematical model which maps the 3D points into the image plane using the perspective transformation and the pinhole camera model.

$$\begin{gathered} sm' = A[R|t]M' \\ s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \end{gathered} \tag{1.1}$$

I order to configure the stereo system I created a script called *Stereo Configuration*. The goal of this script is to set the extrinsic stereo parameters. The following figure 1.7 shows the script that I implemented.
As shown in the previous figure the user can choose: *stereoheight*, *vehicleOffset*, *baseline* which are the translation component of $[R|t]$ matrix. While *Stereo Pitch* represents the pitch rotation with respect to the vehicle reference system. Sensor size are the sensor width and height and are used in order to map the point from the image to the pixel of the camera sensor.
I neglect the camera matrix and the lens distortion coefficients because these parameters are usually

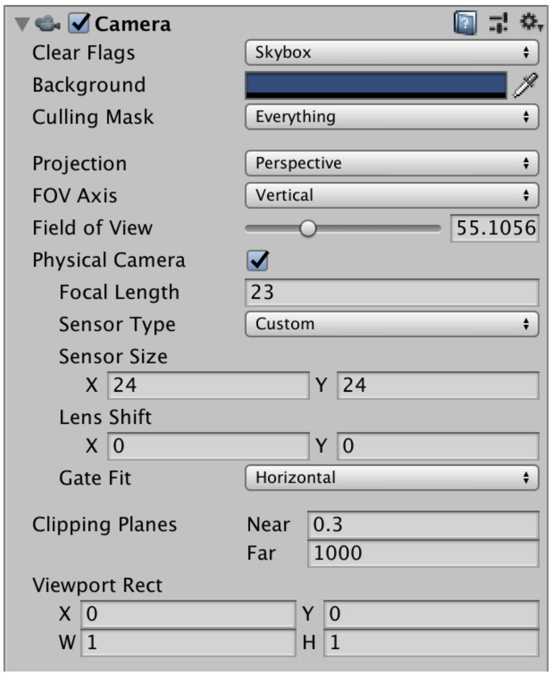


Stereo Configuration (Script)

Script	StereoConfiguration
Real Cameras	☐
Main Camera	Main Camera (Camera)
Camera Left	CameraLeft (Camera)
Camera Right	CameraRight (Camera)
Sensor Size	X 0 Y 0
Stereo Height	1
Vehicle Offset	-0.9
Baseline	0.3
Stereo Pitch	10
Cam Left Fo V	55
Cam Right Fo V	55

Figure 1.7: Stereo algorithm setup

computed using a calibration procedure.
Anyway Unity can model camera with desired: focal length, field of view, clipping planes and can also define the type of projection model used, figure 1.8, so the the camera matrix A can be computed.



Camera

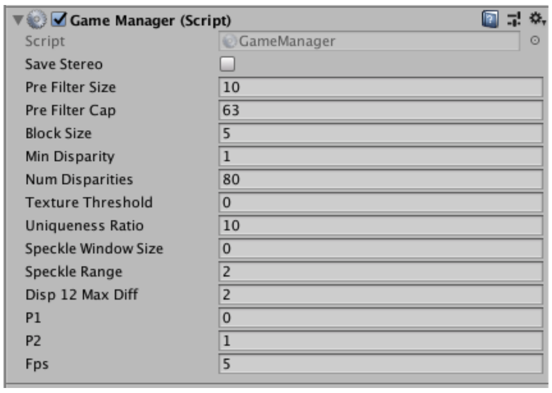
Clear Flags	Skybox
Background	
Culling Mask	Everything
Projection	Perspective
FOV Axis	Vertical
Field of View	55.1056
Physical Camera	✓
Focal Length	23
Sensor Type	Custom
Sensor Size	X 24 Y 24
Lens Shift	X 0 Y 0
Gate Fit	Horizontal
Clipping Planes	Near 0.3 Far 1000
Viewport Rect	X 0 Y 0 W 1 H 1

Figure 1.8: Unity camera configuration

Where $f_x = f_y$ are the focal lengths and c_x, c_y are principal point coordinates. Although this approach allow the user to compute the intrinsic camera parameters it does not include the distortion camera parameters which are used to model the image distortion caused by the lenses. Distortion parameters are necessary in stereo vision especially in the rectification stage otherwise the disparity map cannot be computed correctly.

Stereo algorithm setup/tuning

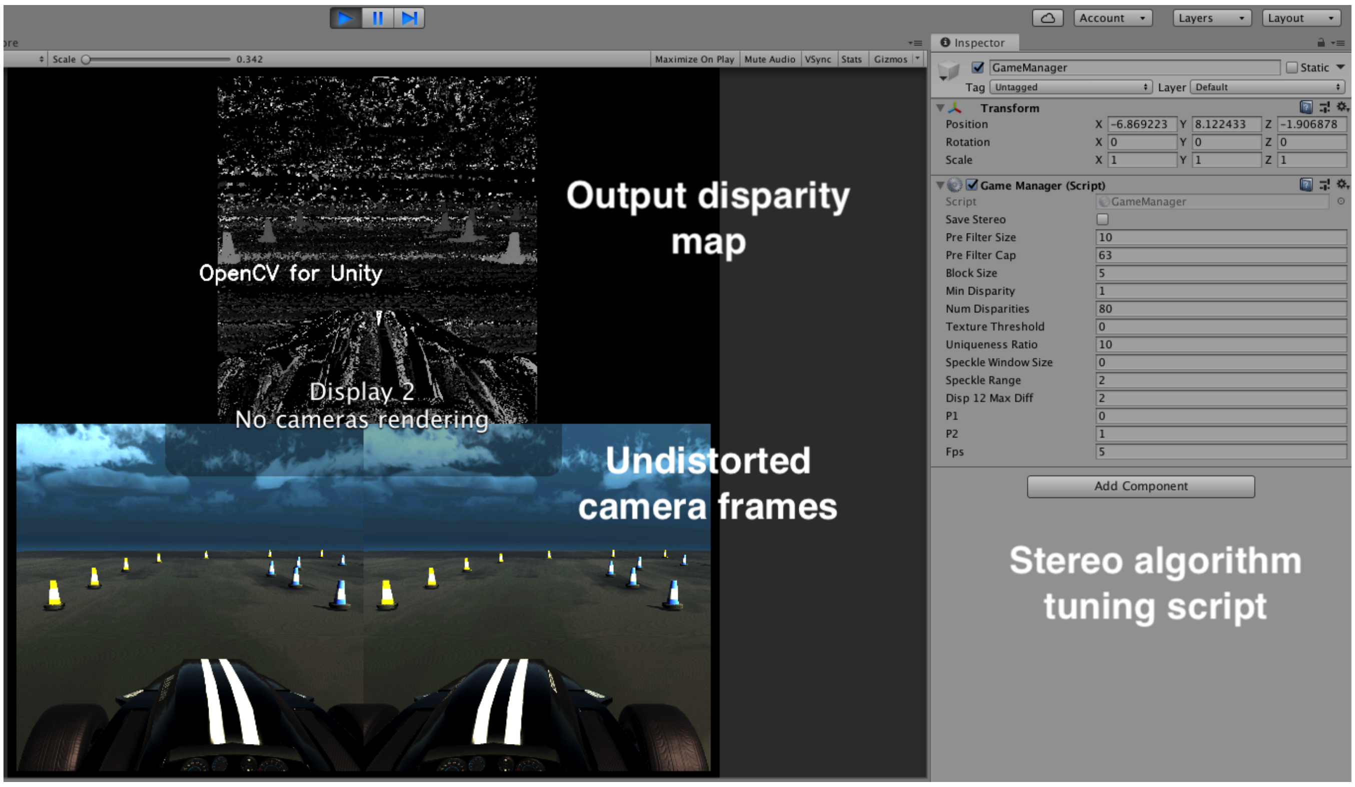
In order to compute the disparity map I use the *OpenCVforUnity* package which includes two stereo vision algorithm: *StereoBM* and *StereoSGBM*. Both the algorithm are very similar in terms of quality, speed and they require the same input parameters. The list of parameters can be set by the user in the *gameManager* script, figure 1.9.


Game Manager (Script)

Script	GameManager
Save Stereo	☐
Pre Filter Size	10
Pre Filter Cap	63
Block Size	5
Min Disparity	1
Num Disparities	80
Texture Threshold	0
Uniqueness Ratio	10
Speckle Window Size	0
Speckle Range	2
Disp 12 Max Diff	2
P1	0
P2	1
Fps	5

Figure 1.9: Stereo algorithm setup

Thanks to this script the user can also tune the stereo algorithm parameters also when the simulation is running.
The following figure 1.10 shows the unity display that allow the user to control the cameras and the disparity map.



GameManager ☐ Static
Tag Untagged Layer Default

Transform

Position	X	-6.869223	Y	8.122433	Z	-1.906878
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

Game Manager (Script)

Script	GameManager
Save Stereo	☐
Pre Filter Size	10
Pre Filter Cap	63
Block Size	5
Min Disparity	1
Num Disparities	80
Texture Threshold	0
Uniqueness Ratio	10
Speckle Window Size	0
Speckle Range	2
Disp 12 Max Diff	2
P1	0
P2	1
Fps	5

Figure 1.10: Unity stereo vision display

Output data

Stereo data

In order to acquire frame from the left and right cameras I used the Unity *Texture2D*, *rawImage* proprieties and the *texture2DtoMat* function included in the OpenCV plugin. One *rawImage* script is linked to each camera, figure 1.11, and take as input the camera frame and the camera texture2D. So, the script convert each camera frame as texture2D. The resolution of the frame depends on the resolution of the texture.
Then, the texture has to converted into a *Mat* variable with *texture2DtoMat* because OpenCV can't work

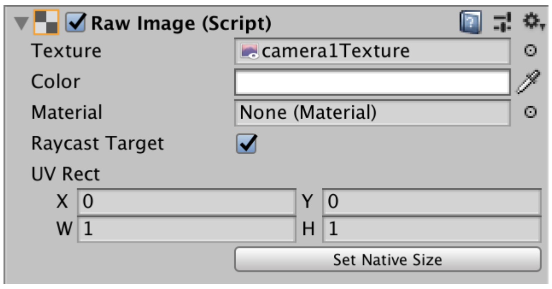

Raw Image (Script)
Texture: camera1Texture
Color:
Material: None (Material)
Raycast Target: ✓
UV Rect
X 0 Y 0
W 1 H 1

Figure 1.11: Raw image script

with texture files. Once the right and left frame are converted I computed the disparity map using the *stereoBM* algorithm.
All the frames are then saved inside three different video. The frame rate of the output video depends on the scene fixed stamp. During my test I fixed that value to 0.05 seconds. This allowed me to ensure that the scene frame rate is the same as the output video.

Simulation parameters

Together with the video information there are also four text files that are generated. The output files can be grouped into two categories: configuration and telemetry one. The configuration script contain the fixed simulation parameters while the telemetry files contains the simulation parameters that change during the simulation.

- Configuration script
  1. Car/stereo-vision parameters (.json): represent the car initial conditions such as: absolute stereo-vision position and orientation, vehicle mass, wheelbase and width.
  2. Cones absolute positions (.csv): Contains the real position of the cone with respect to the absolute reference frame.
  3. Stereo extrinsic parameters (.json): Contains the extrinsic parameters of the stereo- system.
- Telemetry script
  1. Car/stereo-vision telemetry (.csv): Contains all the absolute positions, velocities and orientation of the stereo-system during the simulation.

All the telemetry data are synchronised with the output video frame rate because of the scene fixed time stamp.

Conclusions

Work with a simulation environment can help in the design and test of autonomous vehicles from different point of view. The goal of my project project was to build a simple simulator for a driverless vehicle which is less complicated than an autonomous car but allowed me to understand their advantages and their disadvantages. Thanks to Unity and OpenCV I was able to introduce inside my simulation all the models that I needed. From a simple suspension/tyre model to the camera model. So this simple simulation environment allow me to mix different discipline inside the same environment. Another great thing is that all the models that I used can be more complicated and can integrate also high-level parts, for instance: vehicle stability controls (torque vectoring) or machine learning based algorithm (lane detection, road segmentation or cone detection).