

GRIFFITH UNIVERSITY

3006ICT | ROBOTICS AND COMPUTER VISION

PRACTICE BOOK

Week 6

Stereo Depth Estimation and 3D Scene Understanding

A step-by-step guide from stereo images to metric 3D object location



Student Practice Edition

Lei Wang & Xuefei Yin

3006ICT Robotics and Computer Vision

What this practice book is for

Use this guide while doing the Week 6 lab. It explains what each part is trying to achieve, the concepts you need, the order in which to work, what to inspect after each step, common mistakes, and what evidence to save for your results file.

Important copyright reminder

The Week 6 lab materials are Griffith University course materials. Do not upload, share or publish the lab, solutions, or other protected course materials on public online platforms.

1. Week 6 at a glance

This workshop builds a complete stereo-based 3D perception pipeline. You start with two calibrated, rectified images. A pretrained UniMatch model estimates disparity. Camera geometry converts disparity into metric depth. The depth map is then converted into a coloured 3D point cloud, and the target mask is used to estimate the target's 3D location. Finally, you investigate how disparity errors affect depth and why stereo can fail.



Part	What you learn	Main output
1. Stereo inputs	Rectification, intrinsics, baseline, stereo geometry	Left/right inspection + camera parameters
2. UniMatch	Learned stereo correspondence → disparity	unimatch_left_disparity.npy + .png
3. Depth	Disparity → metric depth	week6_depth.png + depth statistics
4. Point cloud	Pixel + depth → 3D coordinates	week6_point_cloud.png + N×3 points
5. Object 3D location	Mask + depth → target XYZ	Centroid, median depth, (X,Y,Z)
6. Sensitivity/failure	Why depth is sensitive and stereo fails	±10% analysis + failure cases

The key idea

UniMatch gives you disparity. The calibrated stereo geometry turns disparity into metric 3D information. The pretrained model and the camera calibration therefore play different roles: learned correspondence estimates d ; geometry computes Z , X and Y .

1.1 What you should be able to explain after Week 6

- What a rectified stereo pair is and why rectification simplifies correspondence search.
- What the stereo baseline represents physically.
- What disparity means and why nearby objects normally have larger disparity.
- Why the raw numerical disparity array must be used for geometry.
- How $Z = f \times B / d$ converts disparity into metric depth.
- Why non-positive disparity values must be excluded.
- How (u,v,Z) is back-projected into camera-frame (X,Y,Z) .
- Why one valid depth pixel corresponds to one 3D point.
- Why a dense point cloud may need subsampling for visualisation.
- Why median target depth is more robust than using one centre pixel.
- Why the resulting 3D location is in the left-camera frame unless another transformation is supplied.
- Why stereo becomes more sensitive to disparity error for distant objects.
- Which scene properties make stereo correspondence difficult.
- Why invalid depth should be rejected rather than blindly used for robot actions.

2. Prerequisites and setup

The workshop assumes you are using the same Conda environment as previous labs. The provided solution checks whether PyTorch is already installed before installing anything. Do not reinstall a working PyTorch installation unnecessarily.

2.1 Setup checklist

1. Activate your existing Conda environment.

2. Check whether PyTorch and torchvision are already installed.
3. Install only missing packages needed by the workshop, such as matplotlib, pillow and numpy.
4. Clone the UniMatch repository once if it is not already present.
5. Make sure the Week 6 calibrated stereo data folder is accessible.
6. Make sure your working directory is the project directory containing the Part 1–6 Python files.

The solution script uses this general setup. Do not repeatedly reinstall packages that already work.

```
conda activate <environment_name>

python -c "import torch, torchvision; print(torch.__version__, torchvision.__version__)"

# Only if PyTorch is missing:
pip install torch torchvision

pip install matplotlib pillow numpy

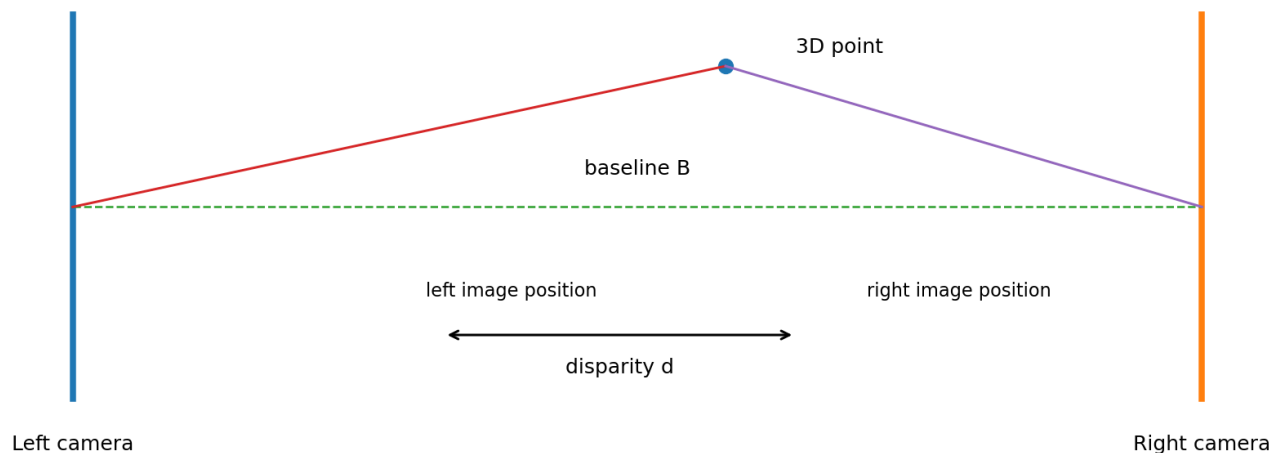
# Only once:
git clone https://github.com/autonomousvision/unimatch.git
```

Do not panic if you do not have a GPU

The provided UniMatch solution automatically chooses CUDA when available and otherwise uses CPU. The solution also uses a smaller maximum inference width on CPU. Your first check is simply to see which device the script reports.

3. Part 1 — Understand the stereo inputs

Before running a deep model, understand the data and camera geometry. Each scene contains a rectified left image, rectified right image, a target mask and camera parameters. The parameters include f_x , f_y , c_x , c_y and the stereo baseline B .



3.1 What the camera parameters mean

Symbol	Meaning	Unit / interpretation
f_x	Horizontal focal length	pixels
f_y	Vertical focal length	pixels
c_x	Principal point x-coordinate	pixels
c_y	Principal point y-coordinate	pixels
B	Distance between the two camera optical centres	metres
d	Horizontal disparity between corresponding points	pixels

Z	Depth in the left-camera frame	metres
---	--------------------------------	--------

```
import json

with open("week6_stereo_calibrated/scene01_indoor_chair_bottle/camera_parameters.json") as f:
    camera = json.load(f)

fx, fy = camera["fx_px"], camera["fy_px"]
cx, cy = camera["cx_px"], camera["cy_px"]
baseline = camera["baseline_m"]

print(fx, fy, cx, cy, baseline)
```

3.2 What does 'rectified' mean?

The two images have been geometrically aligned so that corresponding scene points lie approximately on the same image row. This means stereo matching can focus mainly on horizontal displacement rather than searching over the entire 2D image.

Visual check

Choose several visible scene points. Their horizontal positions should shift between the left and right images, while their vertical positions should be approximately aligned.

3.3 Baseline — physical meaning

The stereo baseline is the physical distance between the optical centres of the left and right cameras. It provides the second geometric viewpoint needed to infer depth.

3.4 Part 1 checkpoint

- ☐ Left and right images display correctly.
- ☐ Resolution is known.
- ☐ fx, fy, cx, cy and B are printed.
- ☐ You can identify horizontal disparity visually.
- ☐ You can explain why corresponding points have similar vertical coordinates.
- ☐ You can explain what the baseline represents.

4. Part 2 — Generate disparity with UniMatch

The goal is to estimate a dense disparity value for pixels in the left image. You are using a pretrained stereo model; there is no model training or fine-tuning in this workshop.

4.1 What UniMatch is doing conceptually

- Input: rectified left and right images.
- The model learns/uses visual correspondence to estimate where matching content appears in the other image.
- Output: a dense disparity field measured in pixels.
- Larger positive disparity generally corresponds to a closer point under the stereo convention used here.
- The numerical result is saved as a NumPy array for later geometry calculations.

4.2 Run the provided solution

The supplied solution checks the device, loads the pretrained checkpoint, runs inference, saves the raw disparity as `unimatch_left_disparity.npy`, and saves a coloured visualisation as `unimatch_left_disparity.png`.

```
python part2_unimatch_disparity.py
```

4.3 What to look for in the terminal

- Using device: cuda or cpu.
- Original image resolution.
- Inference resolution.
- Disparity array shape.
- Disparity data type.
- Minimum valid disparity.
- Median valid disparity.
- Maximum valid disparity.
- Confirmation that both .npy and .png outputs were saved.

4.4 Why can inference resolution differ from the original?

The supplied solution chooses a maximum inference width based on the available device and rounds the inference dimensions to a multiple of 32. If the model runs at a different resolution, the predicted disparity is resized back to the original image size and scaled consistently.

Critical distinction: .npy vs .png

The .png is a colourised display image. Its colours are for visual interpretation. The .npy file contains the actual numerical disparity values in pixels. Parts 3–6 must use the raw numerical disparity array, not the colours in the visualisation.

4.5 How to interpret the disparity image

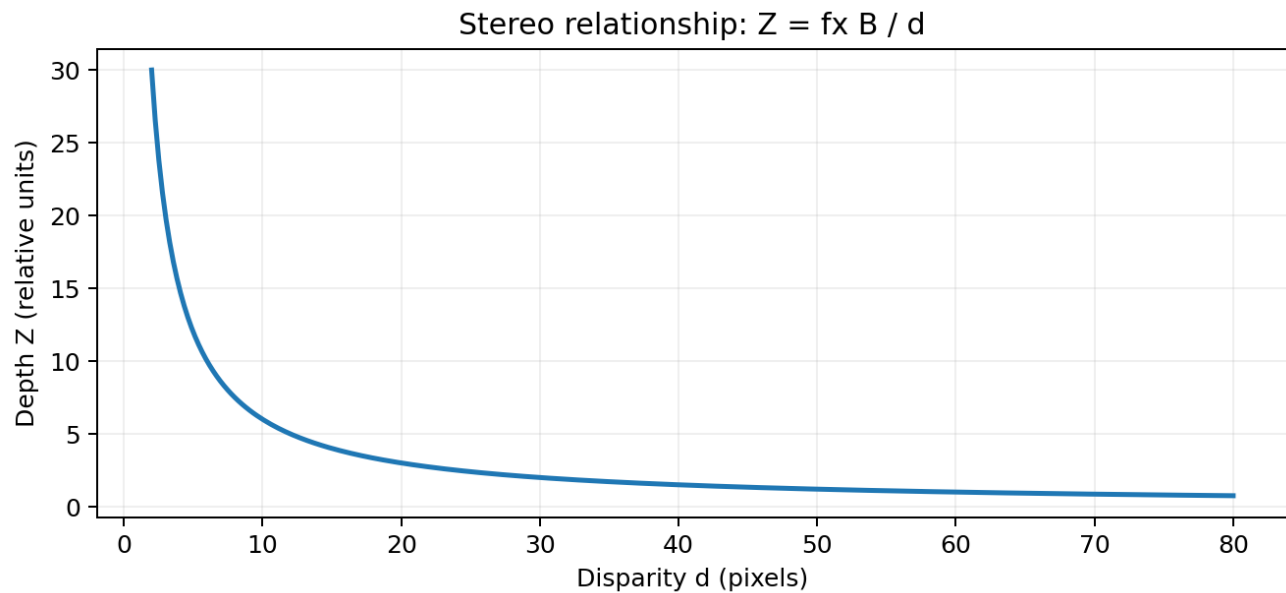
Visual observation	Likely interpretation
Foreground object has larger disparity	It is generally closer to the cameras.
Background has smaller disparity	It is generally farther away.
Very distant region	Disparity may be small and therefore more sensitive to errors.
Missing/invalid area	Correspondence may be unreliable or unavailable.
Sharp disparity change	May indicate an object boundary or depth discontinuity.

4.6 Part 2 checkpoint

- ☐ UniMatch runs successfully.
- ☐ You know whether it used CPU or CUDA.
- ☐ You know the inference resolution.
- ☐ The .npy file exists.
- ☐ The .png file exists.
- ☐ You inspected foreground/background structures.
- ☐ You can explain why the .npy file is required for Parts 3–6.

5. Part 3 — Convert disparity to metric depth

This is the central geometry step. Once disparity is known, the calibrated focal length and baseline allow you to estimate metric depth.



Core equation

$$Z = f_x \times B / d$$

If f_x is in pixels, B is in metres, and d is in pixels, Z is in metres.

5.1 Why depth is inversely proportional to disparity

A closer point creates a larger horizontal displacement between the two rectified images. A farther point creates a smaller displacement. Therefore, increasing disparity decreases estimated depth, while decreasing disparity increases estimated depth.

```
disparity = np.load("unimatch_left_disparity.npy").astype(np.float32)

valid = np.isfinite(disparity) & (disparity > 0)

depth = np.zeros_like(disparity, dtype=np.float32)
depth[valid] = fx * baseline / disparity[valid]
```

5.2 Why exclude non-positive disparity?

- Zero would cause division by zero.
- Non-positive values are not usable correspondences under the disparity convention in this workshop.
- Invalid values should remain excluded from statistics and downstream 3D reconstruction.

5.3 What should the depth image tell you?

The colour bar must be labelled in metres. Do not interpret the colours themselves as metres; use the colour bar. Foreground objects should generally have smaller depth values than distant background objects.

```
values = depth[valid]

print(f"Minimum depth: {values.min():.3f} m")
print(f"Median depth: {np.median(values):.3f} m")
print(f"Maximum depth: {values.max():.3f} m")

plt.imshow(np.where(valid, depth, np.nan), cmap="viridis")
plt.colorbar(label="Depth (m)")
plt.title("Metric Depth")
plt.axis("off")
plt.tight_layout()
```

```
plt.savefig("week6_depth.png", dpi=150)
plt.show()
```

5.4 Quick reasoning checks

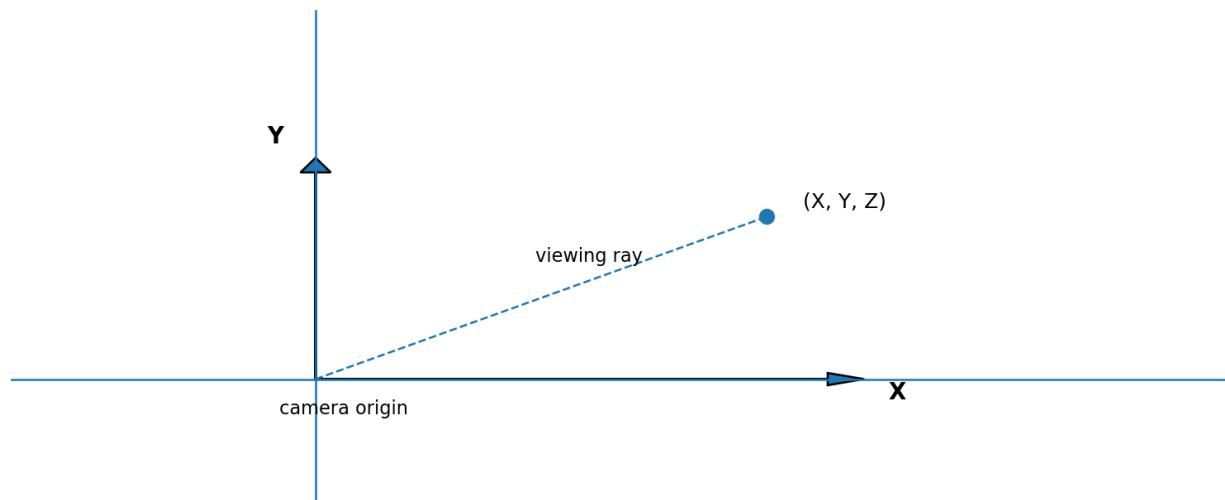
Question	Think this way
What if disparity is halved?	Because $Z \propto 1/d$, depth doubles.
What if disparity doubles?	Depth becomes half as large.
What if disparity is very small?	Estimated depth becomes large and more sensitive to error.
What if disparity is zero?	The formula is undefined, so exclude it.

5.5 Part 3 checkpoint

- ☐ Raw disparity is loaded from .npy.
- ☐ Valid values are defined using finite and >0.
- ☐ Depth is computed with $fx \times B / d$.
- ☐ Minimum, median and maximum valid depth are printed.
- ☐ week6_depth.png is saved.
- ☐ The colour bar is labelled in metres.
- ☐ You can explain the effect of halving disparity.

6. Part 4 — Build a coloured 3D point cloud

A depth map tells you how far each valid pixel is from the camera. Camera intrinsics allow you to turn each pixel and its depth into a 3D point.



6.1 Pixel-to-3D equations

Back-projection

$$X = (u - cx) Z / fx$$

$$Y = (v - cy) Z / fy$$

$$Z = \text{depth}(u,v)$$

Here u and v are image pixel coordinates. X , Y and Z are coordinates in the left-camera coordinate frame used by the workshop.

```
v, u = np.indices(depth.shape)
```

```
X = (u - cx) * depth / fx
```

```
Y = (v - cy) * depth / fy

points = np.stack((X, Y, depth), axis=-1)[valid]
```

6.2 Why does one pixel give one 3D point?

The pixel coordinate identifies a camera viewing ray. The depth value tells you how far along that ray the scene point lies. Together with the camera intrinsics, these determine one 3D location.

6.3 Add colour

```
colours = rgb[valid] / 255.0

points = points[:40]
colours = colours[:40]
```

The colours must correspond to the same valid pixels as the 3D points. Subsampling is for visualisation speed; it does not change the underlying depth map.

6.4 Visualise the point cloud

```
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection="3d")

ax.scatter(
    points[:, 0],
    points[:, 1],
    points[:, 2],
    c=colours,
    s=1
)

ax.set_xlabel("X (m)")
ax.set_ylabel("Y (m)")
ax.set_zlabel("Z (m)")
ax.set_title("Coloured 3D Point Cloud")
ax.invert_yaxis()

plt.tight_layout()
plt.savefig("week6_point_cloud.png", dpi=150)
plt.show()
```

6.5 What to inspect

- Rotate the 3D view.
- Identify obvious foreground structures.
- Identify background structures.
- Check that the cloud has sensible spatial structure rather than looking like random points.
- Notice where missing depth creates gaps.
- Check the axis labels and units.

Point cloud vs mesh

A point cloud is a collection of independent 3D points. A mesh additionally contains connectivity, such as edges and polygonal faces, to represent surfaces.

6.6 Part 4 checkpoint

- ☐ I can explain X, Y and Z.
- ☐ I know the point cloud is in the left-camera frame.
- ☐ I created an $N \times 3$ point array.

- ☐ I created matching RGB colours.
- ☐ I can explain why subsampling helps visualisation.
- ☐ week6_point_cloud.png is saved.

7. Part 5 — Estimate the target object's 3D location

Now combine the 2D target mask with the metric depth map. The important idea is to use many valid target pixels to estimate object depth, rather than trusting one pixel.

7.1 Step 1 — load the mask

```
mask = cv2.imread(
    str(SCENE_DIR / "target_mask.png"),
    cv2.IMREAD_GRAYSCALE
) > 0

target_valid = mask & (depth > 0)
```

7.2 Step 2 — compute the image centroid

```
ys, xs = np.where(mask)

u_c = xs.mean()
v_c = ys.mean()
```

This is the centroid of the mask in image coordinates. It is not automatically the 3D centre of the object.

7.3 Step 3 — estimate object depth robustly

```
z_c = float(np.median(depth[target_valid]))
```

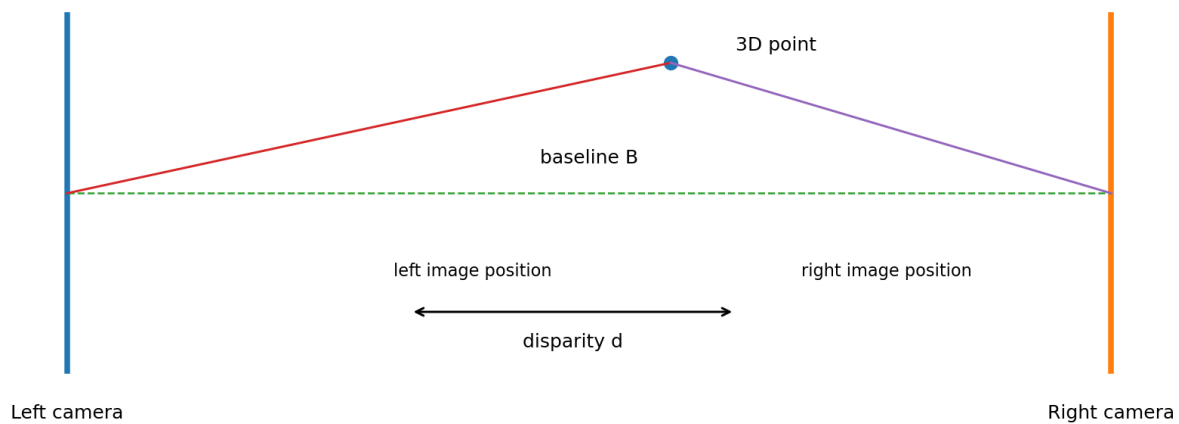
Why use the median?

The centre pixel can fall on an occlusion boundary, background, invalid disparity, or another noisy estimate. The median uses many valid target pixels and is less affected by a small number of extreme depth outliers.

7.4 Step 4 — back-project the centroid

```
Xc = (u_c - cx) * z_c / fx
Yc = (v_c - cy) * z_c / fy
Zc = z_c

print(f"3D location: X={Xc:.3f}, Y={Yc:.3f}, Z={Zc:.3f} m")
```



7.5 Step 5 — compare median depth with centre-pixel depth

```
centre_depth = depth[
    int(round(v_c)),
    int(round(u_c))
]

print(f"Median target depth: {z_c:.3f} m")
print(f"Centre-pixel depth: {centre_depth:.3f} m")
```

Do not assume the two values must match. The comparison is useful because it shows why using one pixel can be unstable.

7.6 Coordinate-frame warning

Very important

The calculated (X_c , Y_c , Z_c) is expressed in the left-camera coordinate frame. To express it in a robot or world coordinate frame, you would need a calibrated rigid transformation consisting of rotation R and translation t .

7.7 Part 5 checkpoint

- ☐ Target mask is overlaid on the left image.
- ☐ Mask centroid (u_c , v_c) is calculated.
- ☐ Median valid target depth is calculated.
- ☐ Centre-pixel depth is also inspected.
- ☐ 3D location is printed in metres.
- ☐ I know which coordinate frame the 3D location belongs to.
- ☐ I can explain why a robot/world transform would be needed.

8. Part 6 — Depth sensitivity and failure cases

Stereo depth is especially sensitive when disparity is small. This part turns that idea into a measurable experiment.

8.1 The $\pm 10\%$ experiment

```
d = float(np.median(disparity[target_valid]))

for scale in [0.9, 1.0, 1.1]:
    d_test = d * scale
    z_test = fx * baseline / d_test
    print(
        f"disparity={d_test:.2f} px, "
        f"depth={z_test:.3f} m"
```

Change in disparity	Expected effect on depth
-10% (0.9d)	Depth increases by more than 10%.
No change (d)	Reference depth.
+10% (1.1d)	Depth decreases by about 9.1%.

The relationship is nonlinear because depth is proportional to $1/d$. This is why simply saying '10% disparity error gives 10% depth error' is not generally correct.

8.2 Why distant objects are more sensitive

Distant objects have smaller disparity. The same absolute disparity error therefore represents a larger fraction of their true disparity, leading to a larger relative depth error.

8.3 Typical difficult regions

Scene condition	Why correspondence can be difficult
Textureless surface	Few distinctive visual patterns to match.
Repeated pattern	Multiple locations may look similar.
Occlusion	A point visible in one image may not be visible in the other.
Reflective surface	Appearance can change between viewpoints.
Transparent region	Image appearance may not correspond cleanly.
Very distant region	Disparity is small, so errors have a larger relative effect.

8.4 Connect depth error to robot behaviour

Ask what would happen if a target or obstacle were estimated to be closer or farther than it really is. An incorrect 3D position could affect navigation, obstacle avoidance or manipulation. Therefore, invalid or unreliable depth should be detected and handled rather than used blindly.

8.5 Part 6 checkpoint

- ☐ I measured the three $\pm 10\%$ cases.
- ☐ I can explain the nonlinear effect.
- ☐ I identified at least two unreliable regions in the disparity outputs.
- ☐ I can explain why those regions are difficult.
- ☐ I can give one realistic example of how wrong depth could affect robot behaviour.

9. The complete data flow — know what each file contains

File / object	Contains	Used for
stereo_left.png	Left RGB image	Input, colour, target visualisation.
stereo_right.png	Right RGB image	Stereo correspondence.
target_mask.png	Binary target mask	Object region and target 3D location.
camera_parameters.json	f_x, f_y, c_x, c_y , baseline	Depth and 3D geometry.
unimatch_left_disparity.npy	Raw numerical disparity	Parts 3–6 calculations.
unimatch_left_disparity.png	Colourised disparity visualisation	Human inspection only.
week6_depth.png	Colour-mapped metric depth	Result/evidence.
week6_point_cloud.png	3D visualisation	Result/evidence.

Dependency chain

camera parameters + disparity $\rightarrow$ depth $\rightarrow$ point cloud $\rightarrow$ object 3D location.

If an earlier stage is wrong, later stages can also be wrong. Always debug from the first incorrect output forward.

10. Recommended coding order

7. Run Part 1 and confirm the stereo images and camera parameters.
8. Run Part 2 and confirm UniMatch works before touching depth code.
9. Inspect the raw disparity statistics and visualisation.
10. Run Part 3 and verify that depth values are in metres.
11. Run Part 4 and inspect the point cloud from several viewpoints.
12. Run Part 5 and compare median target depth with centre-pixel depth.
13. Run Part 6 and record the $\pm 10\%$ sensitivity results.
14. Only then polish your result figures and report.

Do not debug all six parts at once

If Part 4 looks wrong, first check Part 3. If Part 3 looks wrong, first check Part 2. This dependency-based debugging approach is much faster than changing the point-cloud code immediately.

11. Common problems and how to diagnose them

Problem	Check first	Likely cause / fix
PyTorch import fails	Environment and installation	Activate correct Conda environment; install only if missing.
UniMatch repository not found	unimatch/unimatch.py	Run the repository clone command once in the correct directory.
Checkpoint missing	pretrained folder	Allow the provided solution to download it, or check the expected path.
CUDA unavailable	Printed device	CPU is acceptable; expect slower inference and smaller inference width.
Disparity statistics fail	valid mask	Check finite values and disparity > 0 .
Depth contains huge values	Very small positive disparity	This can be mathematically expected but may indicate unreliable far regions.
Point cloud is slow	Number of valid points	Subsample for visualisation.
Point cloud colours look wrong	RGB array alignment	Make sure colours use exactly the same valid pixels as points.
Target 3D location is strange	Mask, depth and coordinate convention	Check target_valid, median depth, centroid and camera intrinsics.
Centre-pixel depth is invalid	Depth at rounded centroid	The centre pixel may have invalid/noisy disparity; compare with median target depth.
Results use wrong units	fx/B/d units	fx and d are pixels; B is metres; resulting Z is metres.

Unit sanity check

fx: pixels | fy: pixels | cx/cy: pixels | disparity: pixels | baseline: metres | depth/XYZ: metres.

12. How to interpret your results

12.1 Do not just report numbers

A strong result section explains what the numbers and images mean. For example, if your median depth is ___ m, identify what scene regions correspond to smaller and larger depth. If your point cloud has ___ points, explain whether the density is useful for visualisation.

Weak statement	Better analysis
The disparity map looks good.	The foreground target has larger disparity than the background, consistent with closer objects having smaller metric depth.
The depth is 2 m.	The median valid depth is 2 m, and the depth image shows the foreground structure at smaller Z than the background.
The point cloud works.	The 3D cloud reconstructs recognisable foreground/background structures; rotating the view confirms spatial separation.
Median is better.	The median uses many target pixels and reduces sensitivity to isolated invalid or extreme depth estimates.
10% error changes depth.	Because $Z = fx_B/d$, the change is nonlinear; a -10% disparity change produces a depth increase larger than 10% .

13. Results file — what should be included

The lab requires a Result.pdf or Result.docx containing the results generated in Parts 2–6. The results file is uploaded separately from the code zip. Discussion-question answers are optional and not marking according to the workshop instructions.

13.1 Recommended structure

Section	Include
Part 2 — Disparity	Device, inference resolution, disparity image, array shape/dtype, min/median/max valid disparity.
Part 3 — Depth	Depth image with metre colour bar, min/median/max valid depth, short interpretation.
Part 4 — Point cloud	Point-cloud image, point-array shape, short interpretation of 3D structure.
Part 5 — Object 3D location	Mask/centroid visualisation, median depth, centre-pixel depth, X/Y/Z in metres.
Part 6 — Sensitivity	$\pm 10\%$ disparity table/results, ≥ 2 difficult regions, explanation of robot impact.

Make the results readable

Do not paste a huge amount of terminal output. Select the important values and figures, label them clearly, and add a short interpretation under each result.

14. Discussion questions — how to answer well

The lab lists discussion questions for each part. The solution files contain reference answers. For your own learning, try answering first without looking at the solutions.

Question type	What a good answer should contain
Why is rectification important?	Connect rectification to same-row correspondences and simpler horizontal search.
What is baseline?	Physical distance between the two optical centres.
Why is nearby disparity larger?	Use $Z = fx_B/d$ and the inverse relationship.
Why use .npz?	It preserves numerical disparity in pixels; the PNG is only a coloured visualisation.
Why pretrained model?	It avoids the data, optimisation and GPU cost of training a stereo network during the lab.
What if disparity is halved?	Depth doubles.
Why exclude non-positive disparity?	Zero causes division by zero; non-positive values are unusable under the workshop convention.
Why can one pixel be unreliable?	It may be background, boundary, occluded, noisy or invalid.
Why median?	Uses many pixels and reduces the effect of outliers.

What transform to robot/world?	A calibrated rigid transformation with rotation and translation.
Why distant depth is sensitive?	Small disparity means the same absolute error is a larger fraction of true disparity.
Why reject invalid depth?	Wrong 3D position can cause unsafe robot decisions.

15. Knowledge checklist — test yourself before finishing

- ☐ I can define disparity in one sentence.
- ☐ I can explain why rectification matters.
- ☐ I can explain what the stereo baseline represents.
- ☐ I know the difference between disparity and depth.
- ☐ I can write $Z = f_x B / d$ from memory.
- ☐ I know the units required for Z to be in metres.
- ☐ I know why $d \leq 0$ is excluded.
- ☐ I can explain why the colourised PNG cannot replace the .npy array.
- ☐ I can write the back-projection equations for X and Y.
- ☐ I know that the resulting XYZ is in the left-camera frame.
- ☐ I can explain why a point cloud is not a mesh.
- ☐ I can explain why subsampling is useful.
- ☐ I can explain why median target depth is robust.
- ☐ I can explain the difference between centroid and 3D object centre.
- ☐ I can explain why $\pm 10\%$ disparity does not mean $\pm 10\%$ depth.
- ☐ I can identify at least two difficult stereo regions.
- ☐ I can explain one safety consequence of wrong depth.

16. Final submission checklist

Item	Done?
Part 2 code runs and produces unimatch_left_disparity.npy	☐
Part 2 disparity visualisation saved	☐
Part 3 week6_depth.png saved with metre colour bar	☐
Part 4 week6_point_cloud.png saved	☐
Part 5 target centroid, median depth and XYZ recorded	☐
Part 5 centre-pixel depth compared with median depth	☐
Part 6 $\pm 10\%$ disparity experiment recorded	☐
At least two unreliable stereo regions identified	☐
Robot-impact explanation included	☐
Python implementation for Parts 2–6 included in code zip	☐
Result.pdf or Result.docx prepared separately and NOT zipped	☐
All important figures have captions/labels	☐
Units are correct and clearly stated	☐
Code uses the raw .npy disparity for downstream calculations	☐
No unnecessary generated files are included	☐

17. One-page mental model

Remember the five transformations

1. Two images $\rightarrow$ disparity
2. Disparity $\rightarrow$ metric depth
3. Pixel + depth $\rightarrow$ 3D point
4. Target mask + depth $\rightarrow$ target 3D location

5. Disparity error → depth sensitivity / robot risk

The most important equation

$$Z = f \times B / d$$

Everything else in the 3D reconstruction pipeline builds around this relationship and the camera intrinsics.

If you understand the dependency chain, Week 6 becomes much easier: UniMatch estimates correspondence; stereo geometry converts correspondence into metric depth; camera geometry converts pixels and depth into 3D; robust statistics help estimate an object's location; sensitivity analysis tells you when the estimate can be trusted.