Representing 3D spaces using acoustics

Exploring a prototype

What do we want to solve?

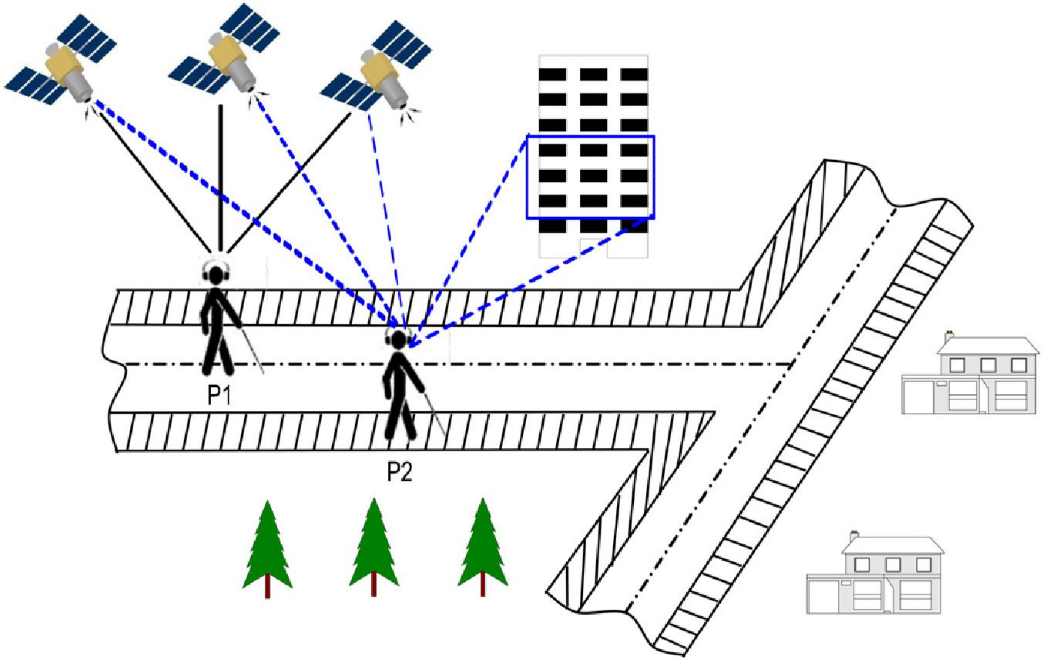
- The problem of navigating the environment without visual information

- Useful for visually impaired people

What has been tried so far?

- GPS-assistance coupled with object detection

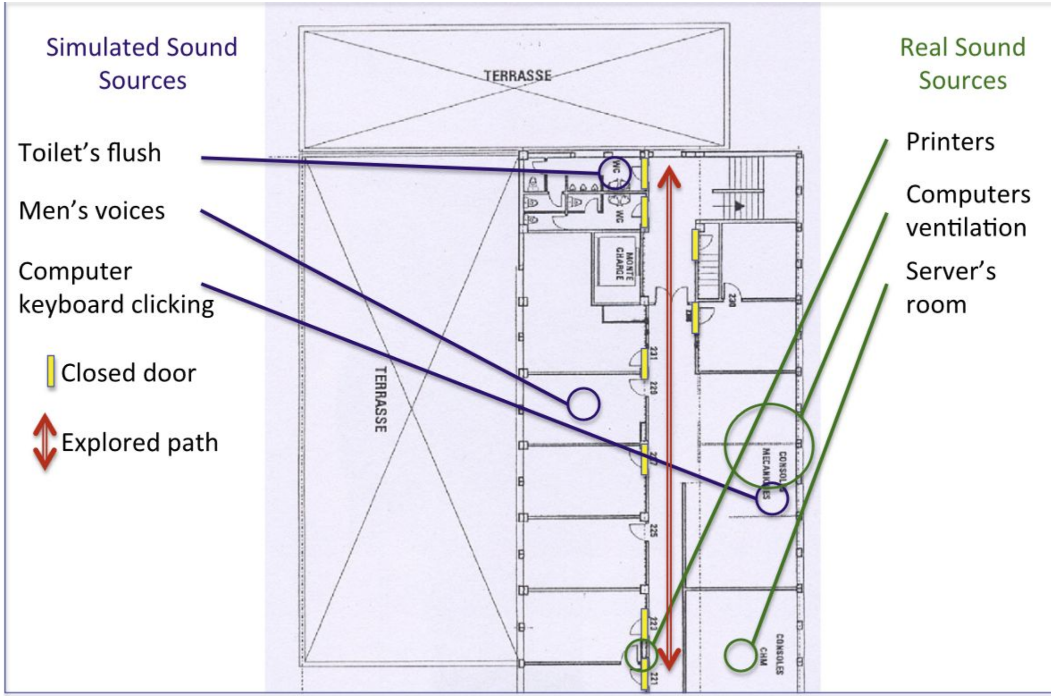
- Recognizing salient features of surroundings to increase accuracy



What has been tried so far?

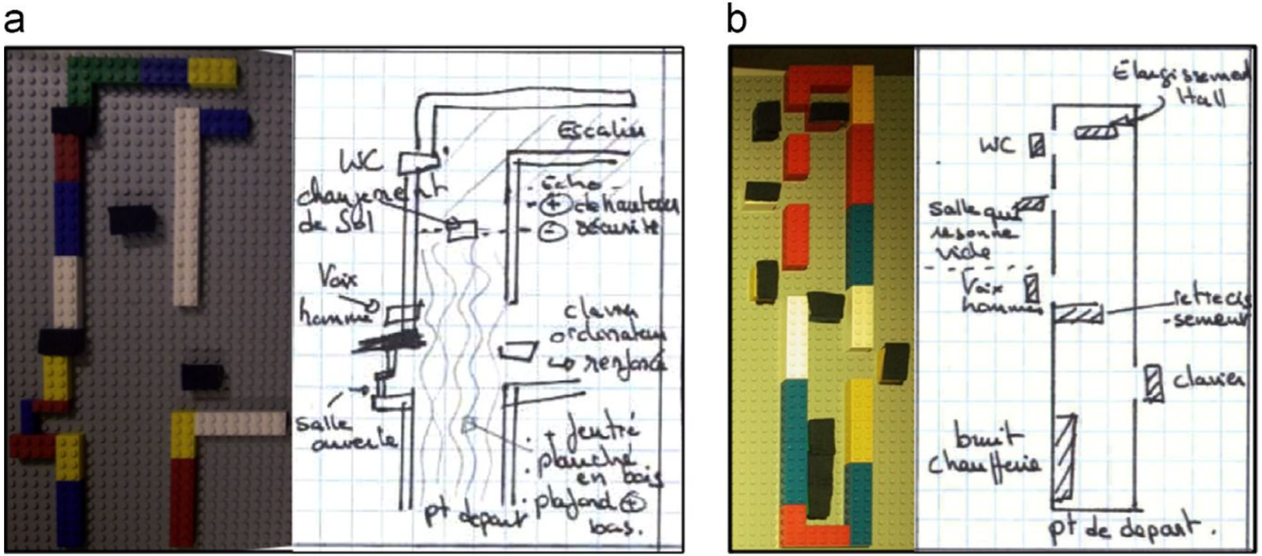
- Simulated soundscapes to make off-sight exploration possible

- Useful for exploring a virtual version of a place before visiting it (e.g. a new office)



What has been tried so far?

- Testers were able to map out building layouts after exploring them virtually

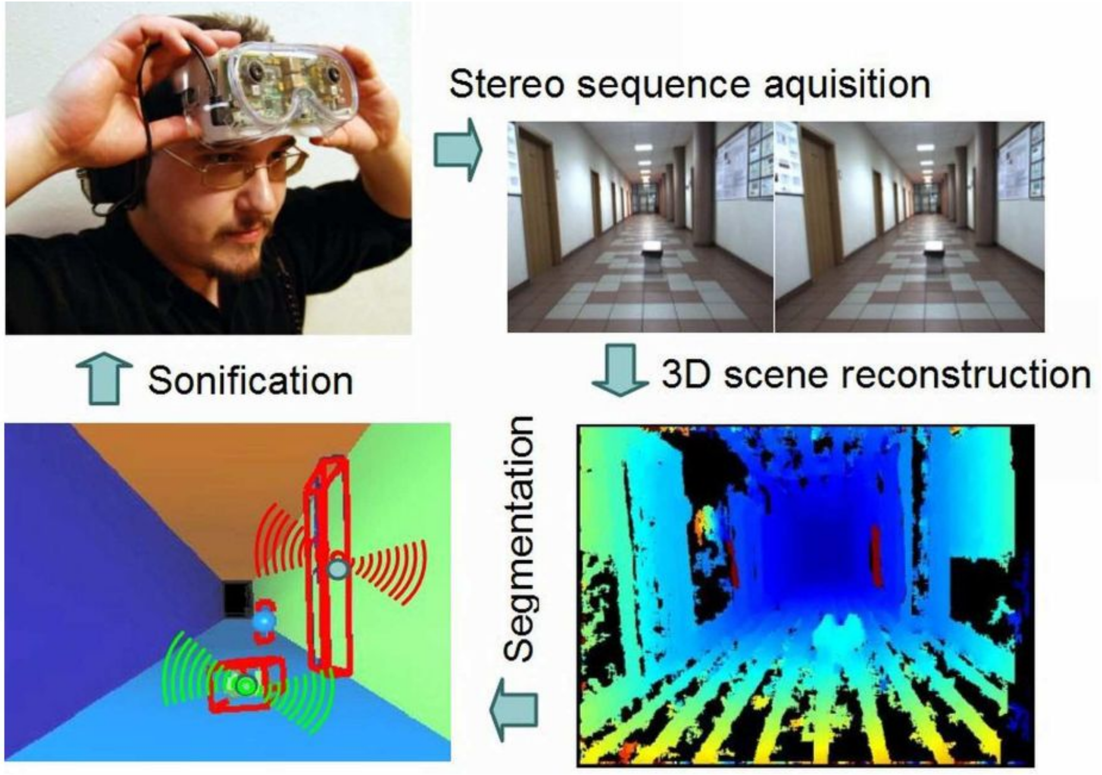


5. Examples of LEGO® reconstructions and annotated drawings for Environment 1 after real (a) and virtual (b) navigations

What has been tried so far?

- Sonification of the environment

- Use sounds with different frequencies, timbres, volume etc. to encode positions



What has been tried so far?

Other ideas are:

- Using object detection to help the user locate doors

What has been tried so far?

Other ideas are:

- Using object detection to help the user locate doors

- Equipping canes with sensors so they can give more nuanced feedback

What has been tried so far?

Other ideas are:

- Using object detection to help the user locate doors
- Equipping canes with sensors so they can give more nuanced feedback
- Using object detection and pass the information via text-to-speech to the user
- etc

Common Problems

- Extra equipment is needed (canes, cameras, sensors, VR-goggles…)
    - often impractical
    - expensive

Common Problems

- Extra equipment is needed (canes, cameras, sensors VR-goggles…)
    - often impractical
    - expensive

- Information is too inaccurate and passed too slowly
    - it takes long to describe an object and it’s position
    - many approaches try to convey information from the perspective of sight

Common Problems

- Extra equipment is needed (canes, cameras, sensors VR-goggles…)
    - often impractical
    - expensive

- Information is too inaccurate and passed too slowly
    - it takes long to describe an object and it’s position
    - many approaches try to convey information from the perspective of sight

- Usage is too distracting in day-to-day life
    - Most tools require full attention from the user
    - Just like our sensory perception it should be subconsciously processable

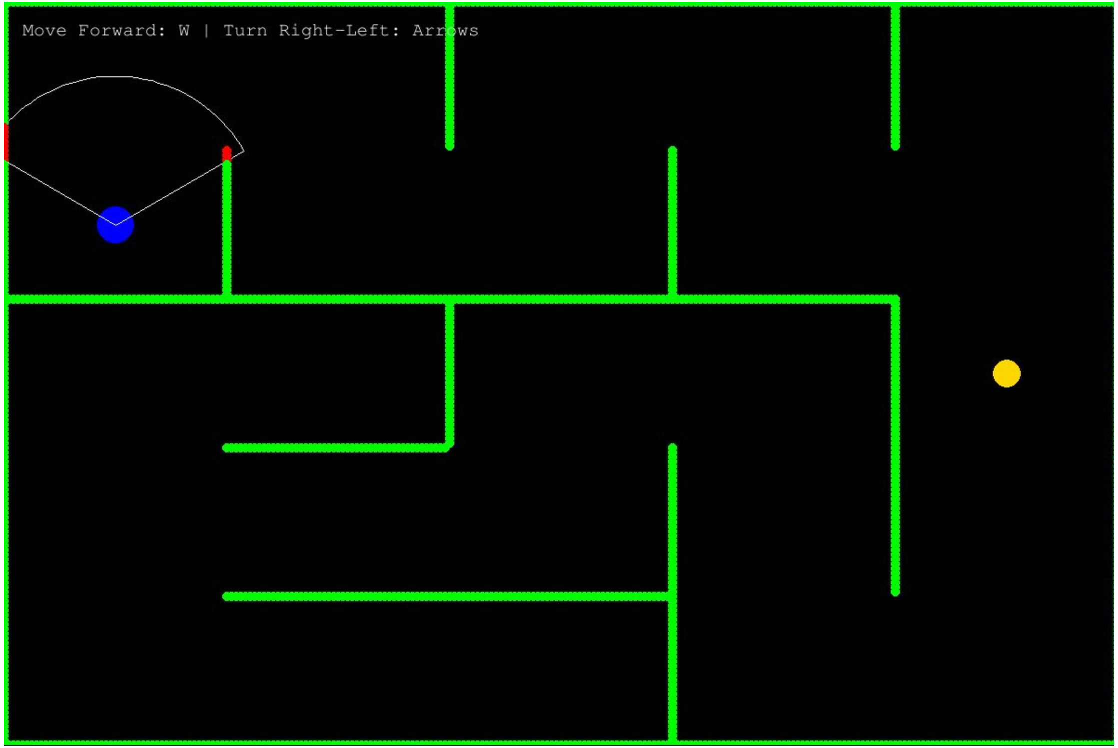
How would you solve this?

Demo game

- simple Python 2D prototype

- allows for simplified experiments

- might lead to an entry for the solution challenge in the future



Please clone and execute

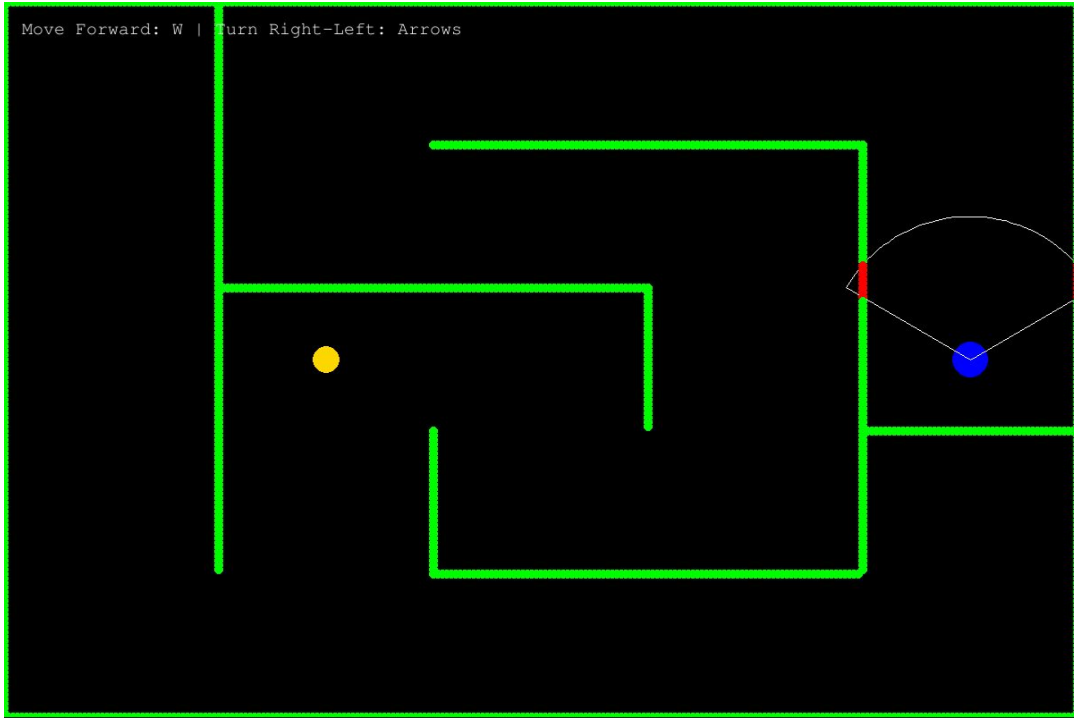
- Github link: https://github.com/GDSC-TU-Berlin/IntroEventTUMaster

- Open project and install necessary dependencies (pygame, numpy)

- Run program to see if it works (with headphones)

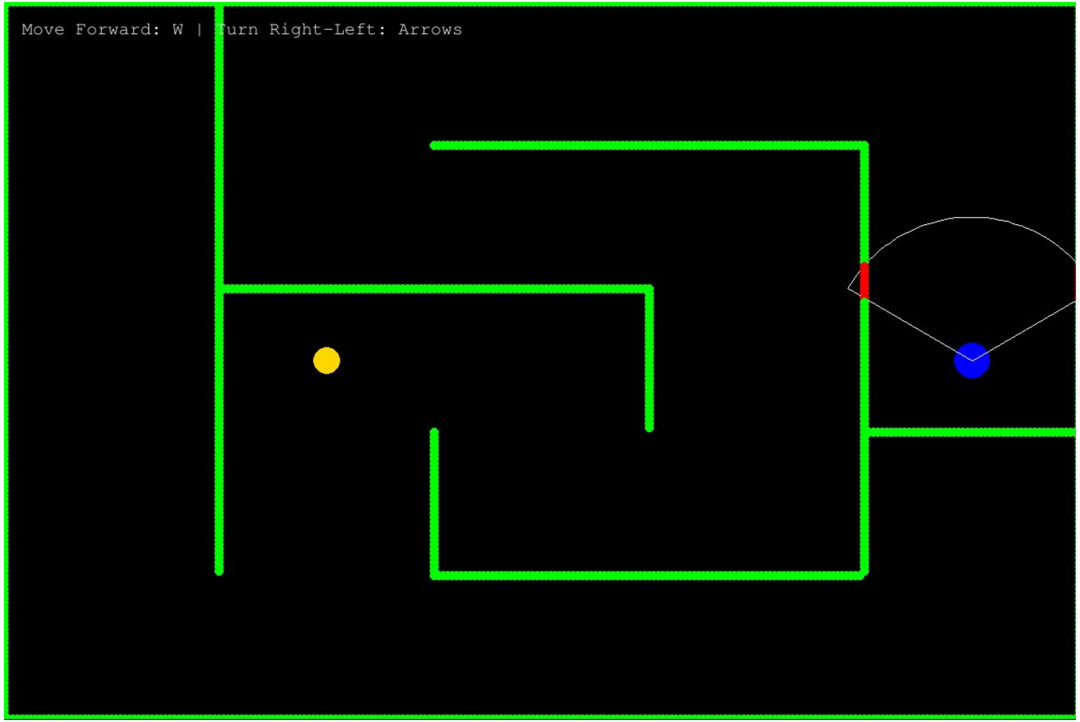
Rules of the game

- The game is played blindly (close laptop halfway)
- You control the 'player' (blue dot)
  - Move forward with W
  - Turn left and right with arrow keys
- Navigate the maze to reach the target (yellow dot)
- If you hit a wall you lose



Rules of the game

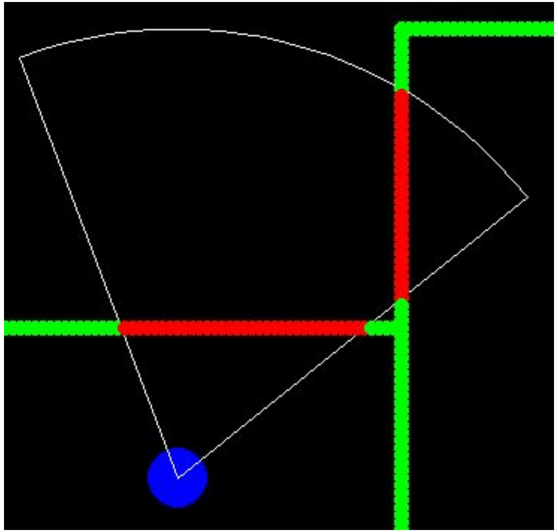
- A monotone sound from left, right or center indicates a wall in that direction (inside viewing bounds)
- A short beeping sound indicates the direction of the target
- Louder Volume means the object gets closer



Relevant code (Player.py)

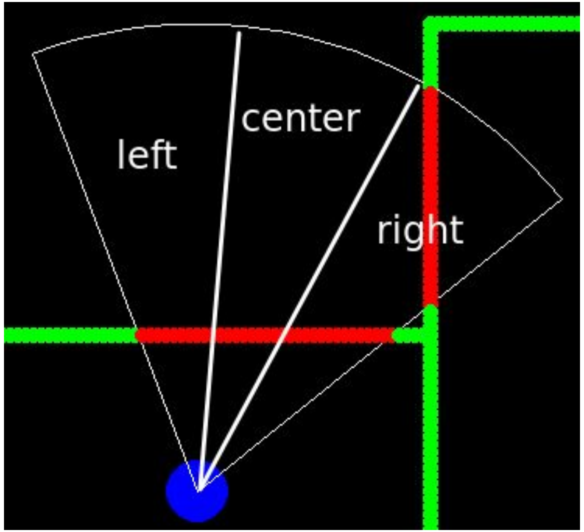
```
class Player:
    MOVEMENT_SPEED = 3
    TURNING_SPEED = Angle(1 / (32 * np.pi))
    """radius becomes radius * min(cell width, cell height) of maze, see Level.generate_obstacles"""
    VIEWING_BOUNDS = PolarCoordinate(Angle(np.pi / 5), radius: 1.5)
```

- VIEWING_BOUNDS control the width of the angle and the radius of the field of view



Relevant code (Player.py)

```
if point.angle.is_in_bounds(left_center_bound, left_bound):
    return Direction.Left
if point.angle.is_in_bounds(right_center_bound, left_center_bound):
    return Direction.Center
else:
    return Direction.Right
```



- Field of view is divided into 3 areas
- Closest point in each area decides volume of respective panned sound

Relevant code (AudioHandler.py)

```python
class Audio:
    MAX_VOLUME = .1
    C3_MAJOR_FREQUENCIES = {
        CMajorScale.C: 131,
        CMajorScale.D: 147,
        CMajorScale.E: 165,
        CMajorScale.F: 175,
        CMajorScale.G: 196,
        CMajorScale.A: 220,
        CMajorScale.B: 247,
    }
```

```python
def generate_sine_wave(frequency: int) -> np.ndarray:
    """
    Generate a monotone sound

    :param frequency: frequency of sound
    :return: sound buffer
    """
    return np.sin(2 * np.pi * np.arange(44100) * frequency / 44100).astype(np.float32)


def generate_sine_wave_beep(frequency: int) -> np.ndarray:
    """
    Generate a beeping sound

    :param frequency: frequency of sound
    :return: sound buffer
    """
    buffer = np.zeros(44100).astype(np.float32)
    buffer[:5000] = np.sin(2 * np.pi * np.arange(5000) * frequency / 5000).astype(np.float32)
    return buffer
```

Relevant code (AudioHandler.py)

```python
def set_panning(self, left, center, right):
    self.left_sound.set_panning(left)
    self.center_sound.set_panning(center)
    self.right_sound.set_panning(right)

1 usage  n1colas
def set_target_panning(self, panning):
    self.target_audio.set_panning(panning)
```

```python
def set_target_volume(self, volume):
    self.target_audio.set_volume(volume)

3 usages  n1colas
def set_volume(self, left, center, right):
    self.left_sound.set_volume(left)
    self.center_sound.set_volume(center)
    self.right_sound.set_volume(right)
```

Relevant code (Game.py, scan_surroundings)

- for all obstacles:
    - if player collides with it -> game ends
    - calculate if obstacle is to the left, center or right
    - adjust volume of respective audio channel accordingly

- if player reaches the target -> game ends

Relevant code (Level.py)

- MAZE_DIMENSIONS are the size of the maze

- The larger the dimensions, the harder it gets

- Start with dimensions of (2, 2) and go up once you are comfortable

```
class Level:
    DEFAULT_MAZE_DIMENSIONS = (4, 4)
    TARGET_COLOR = 'gold'
    TARGET_RADIUS = 30
```

Experiments

- Organize in groups of 3 or 4 (if possible)
- Get familiar with the game and change the code as you please

- Tasks:
  - What challenges arise?
  - What sound cues could be added / changed to make it easier?
  - Does the current 3-channel system make sense? Can it be improved?
  - How would you extend this to 3 dimensions? What else would we need?
  - What other information - apart from depth data - could be used in a real scenario?