Cyber Physical Systems
School of Computation, Information and Technology
Technical University of Munich

The VisionStick

A technical report on the development process

Advitya Chopra ✉, **Chitsidzo V. Nemazuwa** ✉, **Kanaya N. Ozora** ✉,
Oscar E. Navarro Banderas ✉, **Sofia Libman** ✉, **Syed R. Qamar** ✉, **and Süeda Özkaya** ✉

School of Computation, Information and Technology, Technical University of Munich

✉ adi.chopra@tum.de
✉ c.nemazuwa@tum.de
✉ kanaya.ozora@tum.de
✉ ge27zox@mytum.de
✉ sofia.libman@tum.de
✉ syed.qamar@tum.de
✉ sueda.oezkaya@tum.de

September 1, 2025

Abstract — This paper documents the creation of a VisionStick, a project developed as part of the curriculum of the Embedded Systems, Cyber-Physical Systems, and Robotics course at the Technical University of Munich. VisionStick is a smart system integrated with a mobility cane, aimed at helping the visually impaired navigate their surroundings. The device is designed to detect obstacles around the user. When an obstacle is detected, the user is alerted through an alert system of vibration motors.

1 Introduction

The city of Heilbronn is a beautiful place with constant ongoing construction, and as such, it might be particularly tough and dangerous to navigate for those with vision impairment; this is in fact such a common issue, that one of the members of our team even had an experience where they personally had to help out a visually impaired person navigate a place. After the rest of the team learned about this moving anecdote and further brainstorming, as well as a thorough interview, and even a field trip with the Association of the Blind and Visually Impaired Württemberg e.V. Heilbronn (Blinden- und Sehbehindertenverbandes Württemberg e.V. Heilbronn), we came up with the final idea for the device *VisionStick*.

VisionStick is a device that relies on computer vision and ultrasonic sensors to detect obstacles and their proximity and navigate the user to avoid them. At the core of our device lies:

- Raspberry Pi processor: Raspberry Pi 5
- Two video cameras: Raspberry Pi camera V2.1 and IMX 219-B,

- Two ultrasonic sensors: HC-SR04
- And two vibration motors B0DM7XSCQX.

To program the device's logic, we used several Python libraries:

- OpenCV
- Gpiozero
- Ultralitics
- Numpy

1.1 Prior research

1.1.1 Literature and Paper Research

In the early stages of the project, we conducted a short paper research phase to better understand existing approaches in assistive navigation. Each team member selected a set of relevant academic papers, covering topics such as:

- Using CV for obstacle recognition [13]
- Using Ultrasonic Sensors for additional obstacle recognition [13] [15] [2] [5]
- Partitioning the obstacle detection in upper and lower regions [13]
- Using vibration motors to communicate the navigation when an obstacle is detected [13]
- Creating an app for helping the user (option to request help from the caretaker or plot a route with Google Maps) [15] [2]
- Remote assistant-guided navigation via a web platform: a motorized, camera-equipped cane streams live video to remote helpers who guide the user in real time; onboard ultrasonic sensors provide obstacle alerts, with path guidance, security logging, and connection-loss fallback. [6]
- IoT + deep-learning smart assistant architecture: a smart cap (Raspberry Pi + camera, Mask R-CNN) and a multi-sensor smart stick (ultrasonic, IR, LDR, water, accelerometer) connect via Bluetooth/Wi-Fi to an Android assistant and IoT cloud for real-time detection, alerts, and caretaker monitoring [14]
- LiDAR + RGB-D smart cane that combines Cartographer SLAM algorithm for mapping/navigation with an improved YOLOv5 model to recognize common road features (e.g., crosswalks, traffic lights, vehicles); field tests report real-time indoor/outdoor guidance using an onboard Jetson-based drive module. [9]

1.1.2 User Study with BSVW Heilbronn

To make this device useful to real-life users, we decided to interview part of the society that is most likely to actually use it — the visually impaired people. We emailed the local association BSVW (Blinden- und Sehbehindertenverband Württemberg e.V., Heilbronn) and went to a meeting with them to discuss what they would find helpful and, additionally, to see how they use their canes to navigate the world.

Key observations included:

- The traditional cane uses a ceramic ball tip to sense different ground surfaces acoustically.
- Crossing streets without designated tactile markings or traffic lights is difficult; orientation is easily lost, especially near construction or shared pedestrian-bicycle paths.
- Upper-body obstacles are hard to detect with a traditional cane.
- Navigation in non-paved areas can be disorienting, sometimes requiring additional tools such as a compass.
- Many users rely on companions or guide dogs for support, but emphasized the importance of technological solutions for more independence.

From the discussion, we found out that the best way to communicate and warn the user of a potential obstacle is through the vibration motors, as loud environments rendered earpieces, which we originally intended to use, hard to understand and thus useless. We have also observed how the members of the association use their canes on a trip. Users also gave concrete suggestions on how the device should be detachable and potentially extend vibration feedback to the wrists or shoulders.

1.1.3 System Design

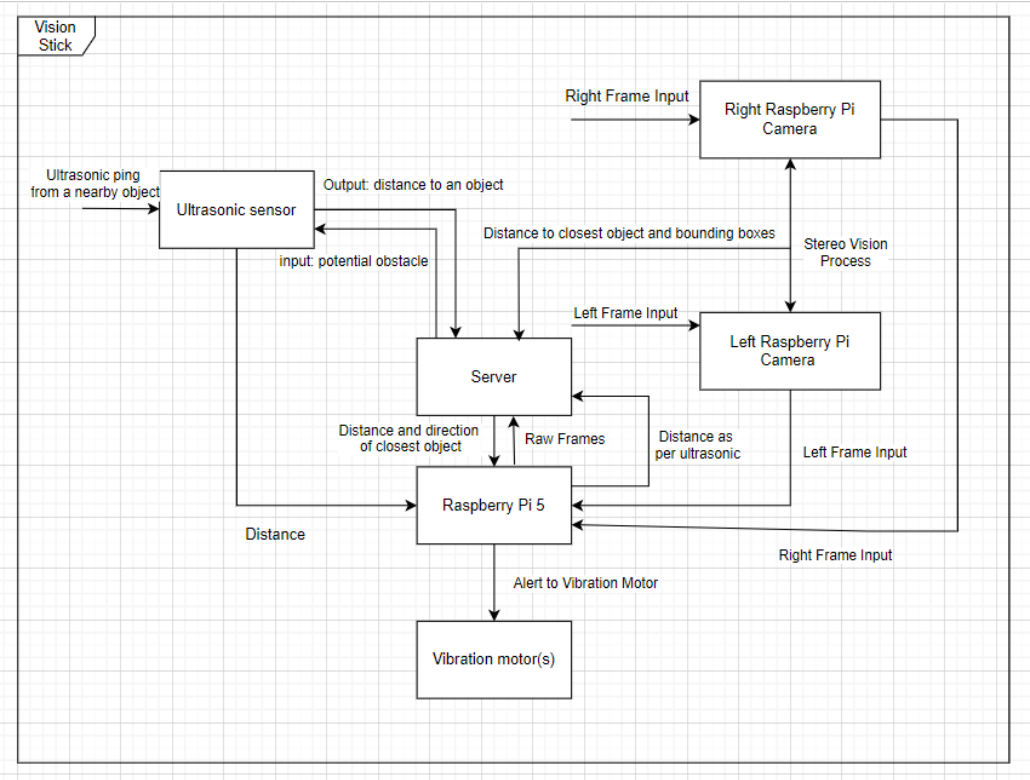

Figure 1 System design of the Vision Stick

From our interview with the Association of the Blind and Visually Impaired Württemberg e.V., Heilbronn, and our own research, we came up with a system diagram for our Vision Stick. Our Vision Stick is composed of 5 main components: a Raspberry Pi, a server, vibration motors, and cameras.

We also decided how and where the device should be positioned on the cane: somewhere in the beginning of the lower third of the cane — not too low to hit walls and floor when the user scans the environment with it, and not too high to miss some essential obstacles below it.

After that, we made a rough 3D model of how we want our VisionStick to look:

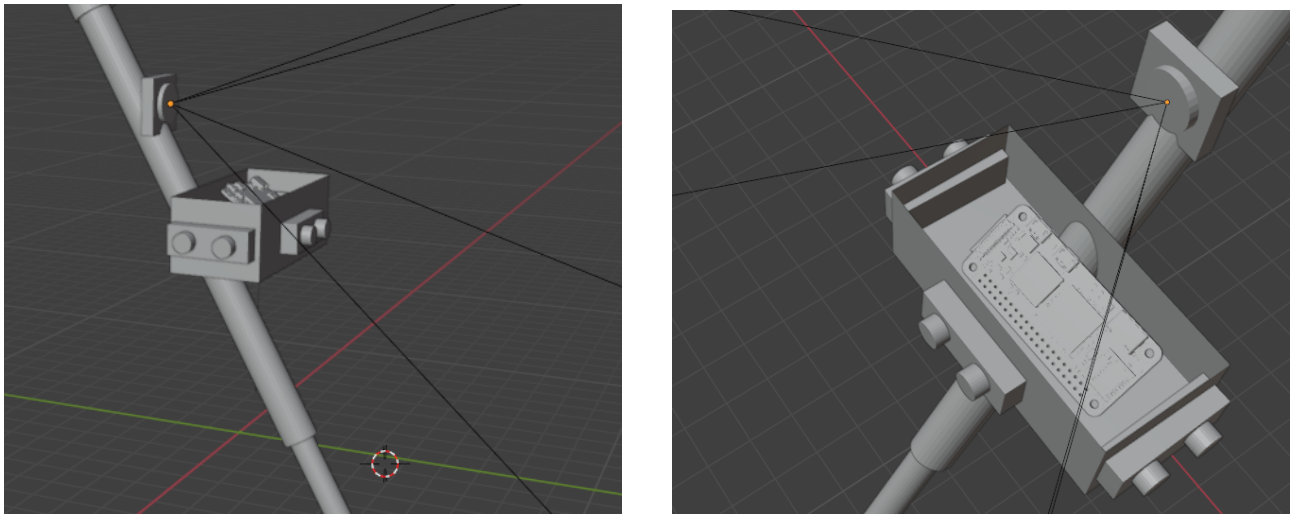

Figure 2 3D model of the VisionStick

Due to some hardware changes during the development phase, the finished piece now looks like this:

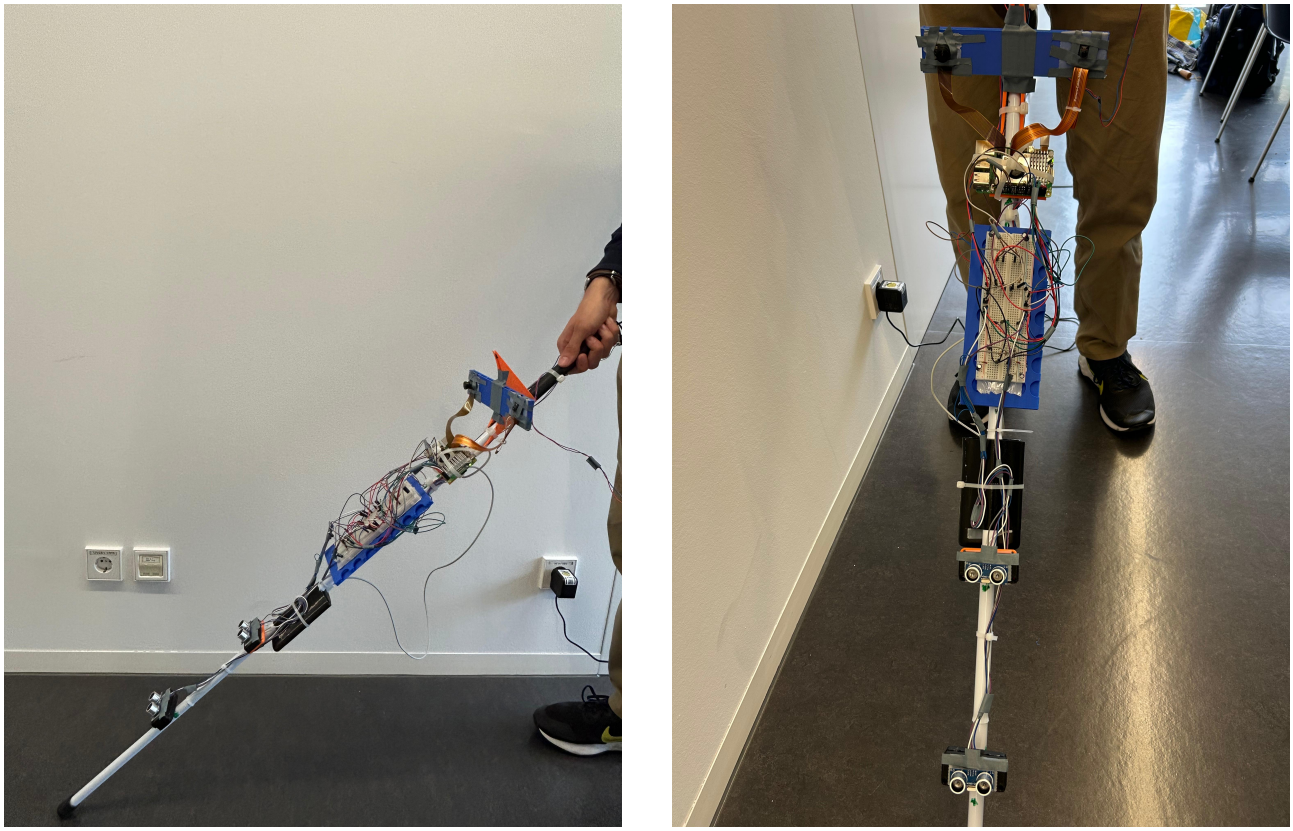

Figure 3 3D model of the VisionStick

2 Hardware and Software review

In this section, we will closely discuss the hardware and software we used for this project.

2.1.1 Raspberry Pi Microprocessor

Initially, we ordered and started working with Raspberry Pi Zero 2W, as it is recommended for mobile devices due to its small size and low power demand. Unfortunately, our project proved to require much more complex computations for computer vision that the Raspberry Pi Zero 2W could not handle. The CV team then found an elegant solution to use a cluster for the computations. Even though the cluster significantly boosted computational speed, the data transfer remained too slow due to the usage of the MicroUSB port and the relatively low RAM size of 512 MB. Because of these limitations, computer vision proved to be delayed, which is a big issue, since our system is time variant, and any delay may cause a lot of inconvenience for the user. Thus, our team collectively decided to switch to Raspberry Pi 5, which is a much more powerful machine with USB 3 ports and 8 GB of RAM, which provides a much smoother and faster data transfer.

2.1.2 Cameras

As mentioned in the introduction, we are using two camera modules in order to enable the usage of a Stereo-Vision for obstacle detection and distance estimation. For the left one, we used a Raspberry Pi Camera V2.1 and a *Sony IMX 219*-B camera model. Initially, we thought it would be sufficient to use only one camera; however, as proven by testing of the model, the accuracy was not high enough to accomplish our purpose of ensuring the safety of the visually-impaired user.

2.1.3 Ultrasonic Sensors

The model of the two ultrasonic sensors used in our project is *HC-SR04*. We use two units to identify the obstacles from two different angles and heights, hence helping the user navigate the streets. The method to do so is fairly easy - the sonar sends an ultrasound ping to the environment, which is then reflected by the surroundings and intercepted back by the sensor. Then the distance is calculated as:

$$D = (\Delta_t * c)/2$$

where
D - distance
Δ_t - is the time for the sound to hit the obstacle and get back
c - speed of sound propagation
If the distance calculated is below the critical distance, i.e., the object is close to

the user and can be deemed as a potential obstacle, the processor activates one of the vibration motors to alert the user of the danger and signify where to move in order to avoid it.

2.1.4 Vibration Motors

We used *B0DM7XSCQX* vibration motors in our project. As mentioned earlier, the motors serve the purpose of alerting the user to any interference in the way. The motors become a signal from the processor that indicates the intensity with which the motors should vibrate. High intensity on the right motor would signify that there is a large obstacle in front of the user, closer to their left side, and the user should move to the right to avoid it. Analogously, high intensity in the left motor would mean the user should immediately move to the left to avoid a big obstacle on the right. Low intensity would mean that the obstacle is small and the user should move, but does not have to move too far to avoid it. The moment the motor stops, the vibration signifies that the user may continue forward safely.

2.2 Software

2.2.1 Computer Vision

To implement computer vision, we used the OpenCV library and the COCO dataset, a dataset of commonly detected objects. Originally, we intended to use only one camera and calculate the focal length, which would allow us to calculate the distance to the object. The formula for calculating the distance with focal length is:

$$D = \frac{Fl \cdot S_a}{S_p}$$

where:
D - distance
Fl - focal length
S_a - actual size of the object
S_p - perceived size of the object

Unfortunately, that would include hard-coding the widths of a large number of objects. The width of each individual object may vary depending on many outside factors that we do not control. Thus rendering this method highly unreliable, which is an issue for us, as we need the computer vision to be somewhat accurate and not endanger the user.

After some brainstorming, the team came up with an ingenious idea of using Stereo Vision, a method that, although it uses two cameras and is not as easy to implement, gives decently reliable results for us to use.

In its core, stereo vision is quite similar to what humans have — a binocular vision.

The idea is simple: take two images made from two cameras positioned in the same plane and combine them together to achieve a proper 3D image from which an accurate distance to said object can be derived. The calculations can be more complex. Described below is an overview of the entire stereo vision process, from calibration to distance calculation.

1. Each of the cameras is positioned so that they're on the same plane, parallel to each other.

2. Each camera is calibrated separately from another with the help of chessboard pictures. To get the distance of an object to the camera, the intrinsic and extrinsic parameters of the camera must be calculated in the calibration step, based on the pinhole camera model. The intrinsic parameters are the optical center and focal length of one camera, while the extrinsic parameters describe the positioning of the two cameras relative to each other, i.e., the camera's baseline. The intrinsic and extrinsic parameters of each camera can be calculated by transforming an image vector to a vector containing 3D world points.

$$w \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = P \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

   where w is the scaling factor and P is the camera matrix containing both intrinsic and extrinsic parameters. Calibration obtains the matrix P from both left and right matrices. The P matrix contains the following components:

$$P = K[Rt]$$

   Where K is the camera intrinsic matrix; this will become useful in the following step.

3. After separate calibration of the two cameras to obtain the intrinsic parameters, stereo calibration is executed to obtain the extrinsic parameters relative to the two cameras. As a result, a reprojection matrix, also known as the Q matrix, is created. This matrix contains all the data needed to convert

the captured 2D points into 3D coordinates: baseline between the cameras (distance between left and right camera centers), focal length, principal point (image center offset), and image distortion.

$$Q = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f_x \\ 0 & 0 & -\frac{1}{t_x} & \frac{c_{x,\text{left}} - c_{x,\text{right}}}{t_x} \end{bmatrix}$$

Here, focal length is written as f_x, baseline is t_x, and the optical centers of the two cameras are c_x and c_y

4. A function `cv2.reprojectImageTo3D` is then applied to the Q matrix to get the 3D coordinates.

5. With the 3D coordinates the actual distance to the object is calculated:

```
for xyxy, confidence, class_id in zip(detections.xyxy, detections.
   confidence, detections.class_id):
    class_name = names[int(class_id)]
    x1, y1, x2, y2 = [int(v) for v in xyxy]
    cx = (x1 + x2) // 2
    cy = (y1 + y2) // 2

    # local window around center for robust median distance
    x_min = max(0, cx - win_size)
    x_max = min(points_3D.shape[1] - 1, cx + win_size)
    y_min = max(0, cy - win_size)
    y_max = min(points_3D.shape[0] - 1, cy + win_size)

    region = points_3D[y_min:y_max, x_min:x_max, :]
    X = region[:, :, 0].flatten()
    Y = region[:, :, 1].flatten()
    Z = region[:, :, 2].flatten()
    valid_mask = np.isfinite(Z)
    Xv, Yv, Zv = X[valid_mask], Y[valid_mask], Z[valid_mask]

    if len(Xv) == 0:
        d_this = None
    else:
        dists_vals = np.sqrt(Xv**2 + Yv**2 + Zv**2)
        d_this = float(np.median(dists_vals) / 1000.0)  # meters
```

2.2.2 Obstacle detection with echolocation

Our Vision Stick works similarly to echolocation. Ultrasonic signals are sent from an origin point, and thanks to the reflection caused by surfaces in the vicinity, these can be received back, thus allowing for estimation of current position and distance.

First, the CV-model detects potential obstacles, estimating information such as their size and the distance to the camera itself. This allows for a somewhat precise estimation of the position of potential obstacles and their distance from the stick.

Secondly, ultrasonic sensors are used to detect the precise distance to hazards and obstacles that are directly in front of the user.

Finally, the vibration intensity depends on how close the nearest obstacle is, and the closer it is, the more intense the vibration output.

2.3 Special additions

2.4 Server

The stereo vision server forms the computational core of a real-time obstacle detection system, processing dual camera streams from a Raspberry Pi 5 to provide distance measurements and directional guidance for navigation assistance. The core crux of the server is a Flask-based HTTPS API that utilises both YOLO object detection and stereo vision algorithms to deliver sub-second processing of incoming frame pairs.

1. Core Components
    - **StereoVisionProcessor** - This component initialises with automatic camera calibration using CalibrationModule, builds stereo matchers via StereoRuntime, and loads a YOLOv11 model for real-time object detection. GPU operations are serialised through thread locks to ensure safe concurrent processing across multiple Flask request handlers.
    - **DisplayManager** - A dedicated background thread that maintains an always-on-top OpenCV preview window. It is used to visualise and check whether the distance and object detection are implemented correctly.
    - **VisionCache** - A thread-safe caching mechanism that stores the most recent vision results for efficient polling by the Raspberry Pi client. The cache maintains temporal metadata, including server timestamps, distance measurements, bounding box coordinates, and frame dimensions required for client-side bias calculations.
2. Network Architecture
    - **POST/process frame** - Accepts Base64-encoded JPEG frame pairs, processes them through the stereo vision pipeline, updates the display preview, and returns comprehensive results including processing times and detection metadata.

- **GET/vision receiver** - Provides a lightweight polling endpoint for the Raspberry Pi to retrieve cached vision data without triggering new processing cycles.

3 Future Improvements

Even though we have achieved great results with the VisionStick being able to detect obstacles and communicate to the user how to avoid them, there are still a number of improvements to bring it to be a properly usable device:

- Currently, the stick is restricted to being tied to the same network to properly work. For the future usability of the stick, it is, of course, better to implement a central server for the processor to connect to every time the device is powered on. Thus attaining the freedom the user would need to navigate the city, and not a single room.
- We use a data set that is limited to a certain number of potential objects it can detect. Thus, for a better user experience, we should expand the dataset to allow for better obstacle evasion.
- We would like to add more interfaces to enable communication with the user, as simple buzzing may not always suffice; e.g., audio cues.
- Additional sensors may also prove useful for the project, for example, the wetness sensor, to find and avoid puddles and wet surfaces, or a GPS module for precise tracking of the location of the user by a caretaker / responsible person.

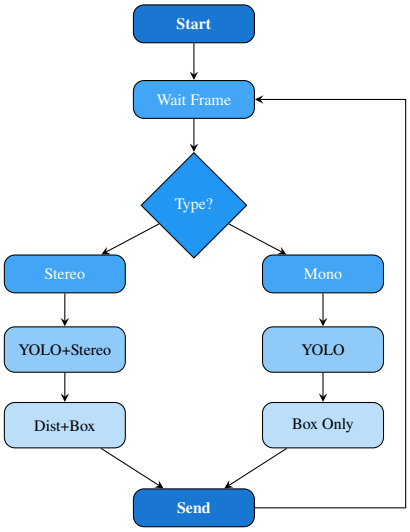

Figure 4 Server Overview

- In relation to the previous point, a care portal to see the live status of the user can also be added so that the caretaker of the user is aware whether the user might need any assistance.
- Implementation of an iOS/Android application to serve as a middleman between the Server and the stick for efficient/logged communication, sending, and receiving.
- Implementation of a central database for event logging and persistent feedback from all stick modules.
- Better hardware management for a cleaner and safer design with no exposed wires and a casing for the Raspberry Pi, as well as the power source.

4 Results

- We have developed and created an alpha prototype of a relatively cheap and low-consumption accessibility device to help visually impaired people navigate the city of Heilbronn.
- We have created an HTTPS server to process information from the Raspberry and poll information such as statistics from the stick.
- We have implemented a stereo vision model for obstacle detection and distance estimation.
- We have implemented sensors and actuators such as ultrasonic sensors and vibration motors in the Raspberry Pi 5 to communicate this information to the visually impaired user.

5 Conclusion

Through the development of this project, we have vastly deepened our knowledge in the field of embedded systems and applied robotics.

The creation of this prototype involved the extensive usage of principles in fields such as electrical engineering, signal processing, and computer architecture.

With a team composed of 7 students, we have all honed our teamwork skills, and we have thus also learned valuable core competencies such as conflict resolution, project management, and division of responsibilities.

Furthermore, this experience is not only valuable to our potential future careers as Information Engineers but also for our personal growth and quality as human beings by making us aware of the challenges experienced by visually-impaired

people, emphasizing to us the need for adopting accessibility solutions and practices in the field of engineering and computer science.

6 References

References

[1] IPB Bonn. “Camera Parameters: Extrinsics and Intrinsics”. In: (2021).

[2] Sandesh Chinchole and Samir Patel. “Artificial Intelligence and Sensors Based Assistive System for the Visually Impaired People”. In: *2017 International Conference on Intelligent Sustainable Systems (ICISS)*. 2017, pp. 16–19. DOI: 10.1109/ISS1.2017.8389401.

[3] Kornia Contributors. *Stereo Camera Documentation*. 2025.

[4] The Pi Hut. *Using HC-SR04 Ultrasonic Range Sensor on Raspberry Pi*. 2025.

[5] S. Krishnakumar et al. “Intelligent Walking Stick with Static and Dynamic Obstacle Detection Technology for Visually Challenged People”. In: *2021 Seventh International Conference on Bio Signals, Images, and Instrumentation (ICBSII)*. 2021, pp. 1–6. DOI: 10.1109/ICBSII51839.2021.9445155.

[6] K. M. Anand Kumar et al. “Remote Controlled Human Navigational Assistance for the Blind Using Intelligent Computing”. In: *Proceedings of the Mediterranean Symposium on Smart City Application (SCAMS ’17)*. Association for Computing Machinery, 2017. DOI: 10.1145/3175628.3175639.

[7] LearnTechWithUs. *Stereo Vision GitHub Repository*. 2025.

[8] LearnTechWithUs. *How to use two Camera Modules on Raspberry Pi 5*. 2025.

[9] Chunming Mai et al. “A Smart Cane Based on 2D LiDAR and RGB-D Camera Sensor—Realizing Navigation and Obstacle Recognition”. In: *Sensors* 24.3 (2024), p. 870. ISSN: 1424-8220. DOI: 10.3390/s24030870.

[10] MathWorks. *Camera Calibration Documentation*. 2025.

[11] Dr. Max. “Computer Vision: Stereo 3D Vision”. In: (2024).

[12] Newbiely. *Raspberry Pi and Ultrasonic Sensors Tutorial*. 2025.

[13] Md. Atikur Rahman and Muhammad Sheikh Sadi. “IoT Enabled Automated Object Recognition for the Visually Impaired”. In: *Computer Methods and Programs in Biomedicine Update* 1 (2021), p. 100015. ISSN: 2666-9900. DOI: 10.1016/j.cmpbup.2021.100015.

[14] Md. Wahidur Rahman et al. “The Architectural Design of Smart Blind Assistant Using IoT with Deep Learning Paradigm”. In: *Internet of Things* 13 (2021), p. 100344. ISSN: 2542-6605. DOI: 10.1016/j.iot.2020.100344.

[15] Aritra Ray and Hena Ray. “Smart Portable Assisted Device for Visually Impaired People”. In: *2019 International Conference on Intelligent Sustainable Systems (ICISS)*. 2019, pp. 182–186. DOI: 10.1109/ISS1.2019.8907954.

[16] Adrian Rosebrock. “Find Distance from Camera to Object using Python and OpenCV”. In: (2015).

[17] Ultralytics. “COCO Dataset Documentation”. In: (2025).