AI-SPY: A Real-Time Object Localization Framework for Cognitive Assistance

Sanjit Prakash

Seminole High School

Junior Science and Humanities Symposium

December 20, 2024

AI-SPY: A Real-Time Object Localization Framework for Cognitive Assistance

Misplacing household objects is a common issue for those with cognitive impairments such as dementia, which leads to reduced independence and frustration for caregivers. This study presents the design and implementation of an affordable, wearable, real-time object localization framework that uses multisensory techniques and artificial intelligence to address this issue. The framework integrates sensors such as an inertial measurement unit (IMU) and time-of-flight sensor (ToF) with machine vision algorithms, creating a lightweight object positioning system. The framework uses YOLO models for object detection, processing 320x320 binned images via the OpenVINO inference toolkit, which allows for optimized inferencing on resource-constrained hardware like the Raspberry Pi. For localizing objects, custom algorithms such as oblique cone tracking were created to determine the position of the user and misplaced objects relative to their environment, whose locations are then automatically updated whenever the objects are reseen. In experimental testing, the system achieved a frame rate of 8-10 FPS, which allows for sufficient overlap between frames for keypoint mapping during odometry at an average walking speed. The system was then placed in a lightweight, 3D printed enclosure, ensuring the device is comfortable to use, especially for elderly users. Empirical testing showed a mean localization accuracy of 19cm and a negligible mean drift error of 0.057m drift/m across various environments. Future improvements will include on transitioning to visual-inertial odometry for enhanced path mapping and more comprehensive object detection datasets such as ImageNet to improve object detection robustness.

Literature Review

The Urgency of an Object Localizing Support System

Dementia affects over 55 million people worldwide, with nearly 10 million new cases reported annually. It is also expensive to care for, with 1.3 trillion dollars spent solely on dementia care in 2019 (World Health Organization, n.d.). Cognitive impairments significantly reduce a person's ability to perform basic daily tasks to provide for themselves, due to symptoms such as memory loss, loss in judgement, difficulty speaking, loss of direction, but most importantly forgetting where everyday objects are kept (*What is dementia?*, n.d.). Imagine if a person with dementia lost essential items regularly, such as their phone, wallet, or more importantly – their medications (*Rummaging, hiding, and hoarding behaviors*, 2017). A system that could quickly and accurately locate objects can reduce these risks and potentially be lifesaving.

Current Means of Object Localization Frameworks

Current object localization frameworks use multisensory techniques and artificial intelligence to operate, but are designed for industrial or automotive context, and use complicated equipment such as LiDAR and radar for positioning (Syamal et al., 2024). These frameworks tend to be expensive and bulky, making them impractical for everyday use and for people of old age. While these systems are effective, it is rather impractical to carry around sophisticated sensors for this purpose. In contrast, the need for smaller, more affordable solutions is present, such as the proposed system for people with dementia. By utilizing more lightweight sensors like the IMU and ToF sensors previously mentioned, along with portable computing hardware such as a Raspberry Pi, it becomes possible to create systems that can be affixed to the body for all-day localizing without discomfort. The transition toward smaller positioning systems is key for allowing these vision systems to become viable options for those who would benefit from it.

Engineering Goals and Materials

The engineering goals that are set forth for this project are as follows: the system will be compact and lightweight, so it is as comfortable as possible for the wearer; it will be able to run fast enough to be able to have sufficient key point overlap between frames for accurate odometry; it will also be able to accurately localize objects and be cost effective (<\$300). Furthermore, it will integrate multiple sensors for more accurate and redundant positioning, and be able to work under various lighting conditions, indoors and outdoors. If all these goals are achieved, the overarching goal of creating a system that can help people with dementia will also be sufficiently met.

Materials

The materials that will be used in this project are as follows: a Raspberry Pi for portable computing (\$60), a Raspberry Pi Camera Module (v3 Wide) to allow for object detection (\$35), a BNO085 IMU for heading (\$25), a VL53L1X ToF to measure distance to the nearest object (\$15), a PiJuice Zero HAT for battery regulation (\$35), a 1200mAh battery (\$25), and a 3D-printer with filament for printing of the enclosure and necessitated parts (total <\$200).

Implementation

The implementation of this system is as follows: first, the intrinsic matrix of the camera is derived to allow for adjustment of each frame taken from the camera for its camera warps to allow for actually accurate odometry. Next, the visual odometry system is implemented. Following that, YOLO is utilized for object detection and to investigate possible optimizations. Finally, object tracking using a custom “oblique cone tracking” (more on its meaning in that section) algorithm is used for novel object positioning.

Derivation of the Camera's Intrinsic Matrix

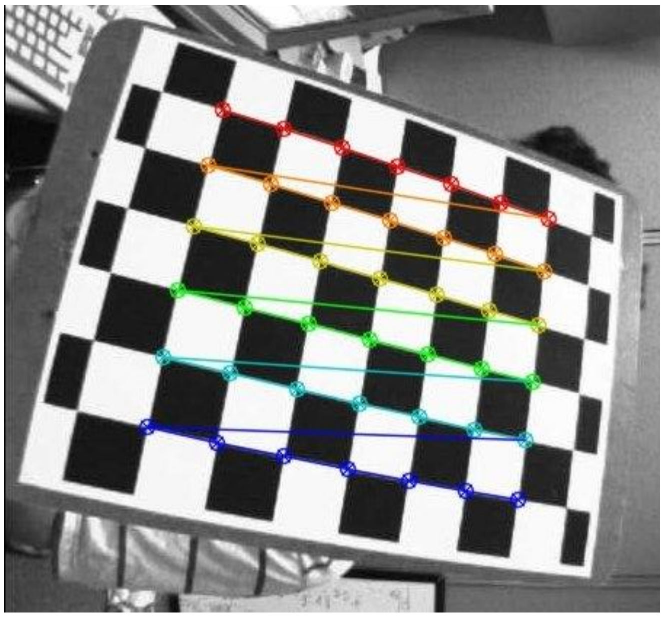
All pinhole cameras cause some level of distortion to the image. To adjust for these distortions, the intrinsic matrix of the camera needs to be calculated. The intrinsic matrix is unique to every camera and contains information like the focal length ($f_x,\ f_y$) and the optical camera centers (x_0, y_0) and is expressed as a 3x3 matrix as shown in (1) (Simek, 2013).

$$intrinsic = \begin{bmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

A common method of calculating the intrinsic matrix for a camera (simply, "calibration") is to use a checkerboard image as shown in Figure 1 to map out what the camera sees to a predefined pattern (*Camera Calibration*, n.d.).

Figure 1

Example of a calibration image

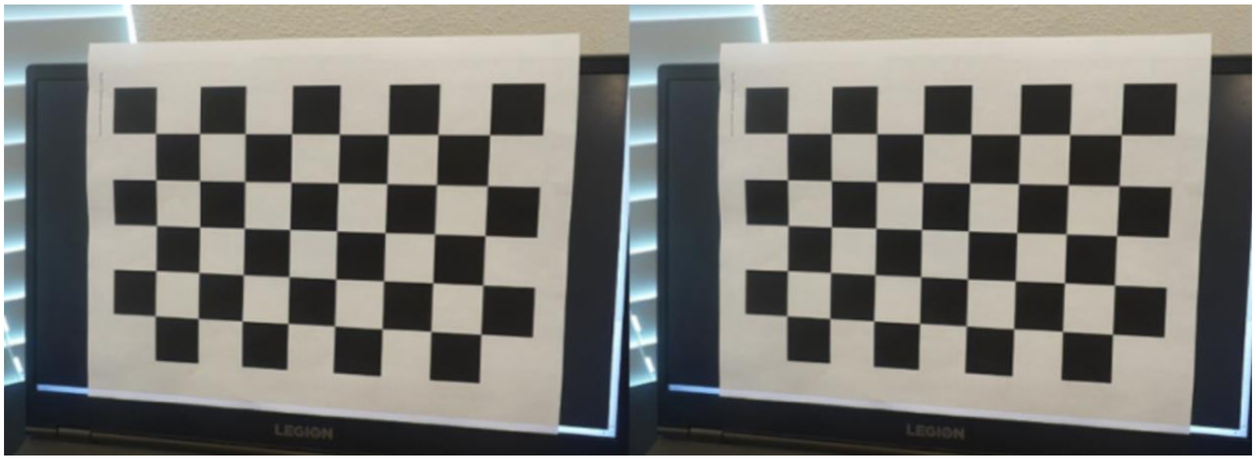


Note. See how the checkerboard pattern is identified (*Camera Calibration*, n.d.).

For camera calibration OpenCV provides many functions to aid in the intrinsic matrix calculations. The `cv2.findChessboardCorners` function was used to identify checkerboard patterns in images and the `cv2.getOptimalNewCameraMatrix` function was used to compute the intrinsic matrix. In this case, $(f_x,\ f_y)$ was approximately (568, 562) and $(x_o,\ y_0)$ was approximately (329, 245). Figure 2 show the changes due to distortion adjustment.

Figure 2

Checkerboard image before/after distortion adjustments



Note. Notice how the checkerboard on the left is more distorted around the edges than the image on the right.

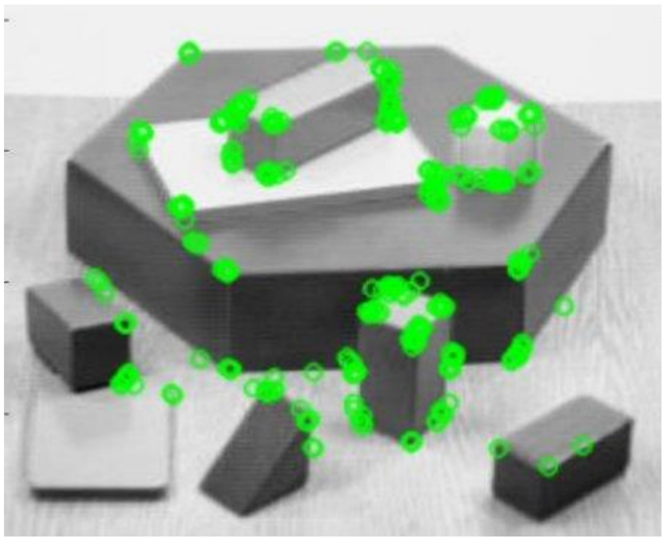
After corrections, the checkerboard is a more refined image with less warping around the outside edges. With these intrinsic parameters, it is now possible to conduct visual odometry.

The Visual Odometry Algorithm and Scale Referencing

Visual odometry works by using a feature tracker to match points between pairs of images and then using geometry and triangulation to estimate the displacement from the first frame to the second without knowledge of scale or distance (Nister et al., 2004). In this device, point mapping is done through the ORB (Oriented FAST and Rotated Brief) technique provided by OpenCV Labs, as shown in Figure 3 (ORB, n.d.). To ensure accurate displacement calculations, a minimum key point requirement of eight was put in place for the device.

Figure 3

Illustration of ORB feature tracking



Note: See how the key points of the image are highlighted in green (ORB, n.d.).

For this implementation, there is only have the liberty of using one camera for the sake of reducing processing power needed and to make the device more compact. The downside to having only one camera is that only relative distance can be calculated through odometry due to lacking depth information (Kitt et al., 2011). To mediate this issue, the absolute distance known when moving in the depth-axis and during rotations can be referenced using the IMU and ToF sensors. For the following equations, let d_o be the displacement vector from the odometry, Δy_t be the displacement from the ToF sensor, d_{scaled} for the final displacement vector, and θ for the angle between the original position and final position (if any). Given a displacement vector as shown in (2) from the odometry, d_{scaled} can be calculated using (3).

$$d_o = \begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} \tag{2}$$

$$d_{scaled} = d_o \cdot \frac{\Delta y_t}{y_o \cos\theta} = \begin{bmatrix} x_o \cdot \dfrac{\Delta y_t}{y_o \cos\theta} \\ y_o \cdot \dfrac{\Delta y_t}{y_o \cos\theta} \\ z_o \cdot \dfrac{\Delta y_t}{y_o \cos\theta} \end{bmatrix} \tag{3}$$

Solving for d_{scaled}, an estimate of the user's movement that is scaled to the environment can be calculated, which will allow for construction of a map based on real-world scale. However, the reasonable assumption can be made that the person using the device will not move directly in the x-axis or z-axis, as the average person does not shuffle sideways to move nor can they fly.

Object Detection using YOLO through OpenVINO inferencing framework

This device uses the latest generation of YOLO object detection models for identification. However, even while running the nano model using the standard PyTorch framework consumed around 800ms per frame for inference. This would not align with the goal of being able to run fast enough to capture a big enough overlap of key points during odometry, so a faster solution is needed. Therefore, benchmarks were run on all possible export methods to see what would yield the lowest inference time at imgsz=640 (640x640 images). One thing to be noted is that the Raspberry Pi only supports CPU-based object detection, and if practically any CUDA-based GPU was used, the inference would be significantly faster.

Table 1

Benchmarking of all CPU-based export frameworks for yolo11n.pt at imgsz=640

Format	metrics/mAP50-95	Inference time (ms)	FPS
PyTorch	0.6100	683.15	1.46
TorchScript	0.6082	869.01	1.15
OpenVINO	0.6115	148.51	6.73
NCNN	0.6106	161.65	6.19

As shown in Table 1, OpenVINO and NCNN are within a margin of error to each other. The rationale for using OpenVINO is that it allows for scaling of images down to imgsz=320, which allows for faster runtime when accounting that odometry will also take around 50-100ms per frame depending on the number of key points.

Target Object Distance Detection through Oblique Cone Tracking

Simply pointing a ToF sensor at an object is not ideal, as there is a high possibility that the field of view ("FoV") of the ToF sensor will include other objects. Luckily, the VL53L1X sensor provides a method of manipulating its original 27-degree FoV down to a minimum of 15 degrees, with the original FoV represented as a 16x16 grid of boxes and the minimum FoV being represented as a 4x4 grid of boxes. This range of interest ("ROI") can be moved to anywhere inside its original 16x16 bounds. It is also given that an 8x8 ROI is equivalent to 20 degrees (STMicroelectronics, 2024). Using this information, the degrees that a certain FoV covers from its ROI, can be determined with a logarithmic function (4), and consequently, also its inverse (5).

$$ToF_{FOV} = \frac{\log(ToF_{ROI})}{\log(1.25)} + 2 \tag{4}$$

$$ToF_{ROI} = 1.25^{(ToF_{FOV} - 2)} \tag{5}$$

The object detection algorithm previously described returns the bounding boxes of known objects, along with their coordinates in the form of (x_1, y_1, x_2, y_2) in pixels. The maximum area of the ToF ROI can also be calculated in pixels if a function could convert FoV degrees into a ratio of screen area. Hence, see (6), and its inverse, (7), with $camera_{FOV}$ being the FoV of the Raspberry Pi camera, which is 107 degrees, and $camera_{ratio}$ being the ratio between the maximum FoV of the ToF sensor and the maximum FoV of the screen.

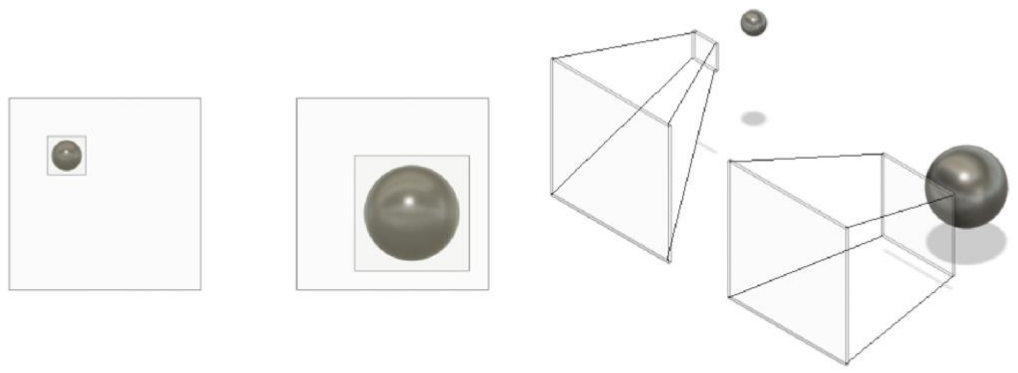
$$camera_{ratio} = \frac{ToF_{FOV}^2}{camera_{FOV}^2} \tag{6}$$

$$ToF_{FOV} = camera_{FOV} \cdot \sqrt{camera_{ratio}} \tag{7}$$

Now that there are means to calculate the area of the maximum FoV in pixels, the intersection of the nearest bounding box with the maximum FoV can be taken to create an overlap surface with bounds (ox_1, oy_1, ox_2, oy_2) and ensure that the area of the surface is positive. By iterating through all possible ROI sizes from 16x16 to 4x4, if possible, the largest ROI size that can fit inside the overlap box is chosen. This method of object distance tracking can be described as oblique cone tracking due to the irregular conic shapes created by the ToF emitters, as shown in Figure 4.

Figure 4

Visualization of the field of view of the ToF sensor's emitters



If such an ROI size exists, let it be called ToF_{size}, and let the center of it is defined as shown in (8), where ToF_{center} is the center coordinate of the ROI, and $camera_{length}$ is the length (in pixels) of the view of the camera.

$$ToF_{center} = \left(\frac{\frac{ox_1 + ox_2}{2} \cdot 16}{camera_{length}}, \frac{\frac{oy_1 + oy_2}{2} \cdot 16}{camera_{length}} \right) \tag{8}$$

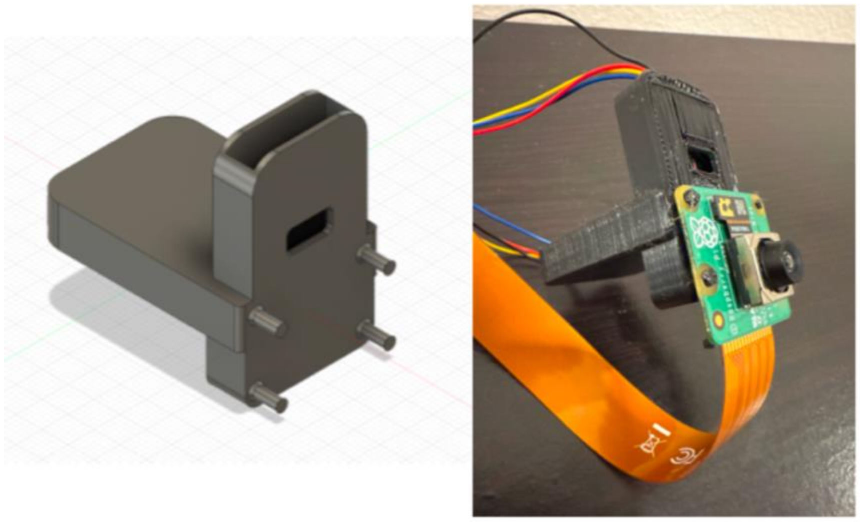
Now that both ToF_{size} and ToF_{center} have been calculated, the VL53L1X library can be used to write these values to the sensor itself and allow for more accurate detections of the distance between the user and the object (Adafruit, n.d.).

Hardware and Sensor Structure

To ensure that every measurement of the ToF sensor and every frame from the camera can be registered at a certain heading, all three sensors must be perpendicular to each other. This is achieved through a custom mount that will house all three sensors, as shown in Figure 5.

Figure 5

3D CAD Model of Sensor Housing Compared to Real-World Image of Sensor Housing



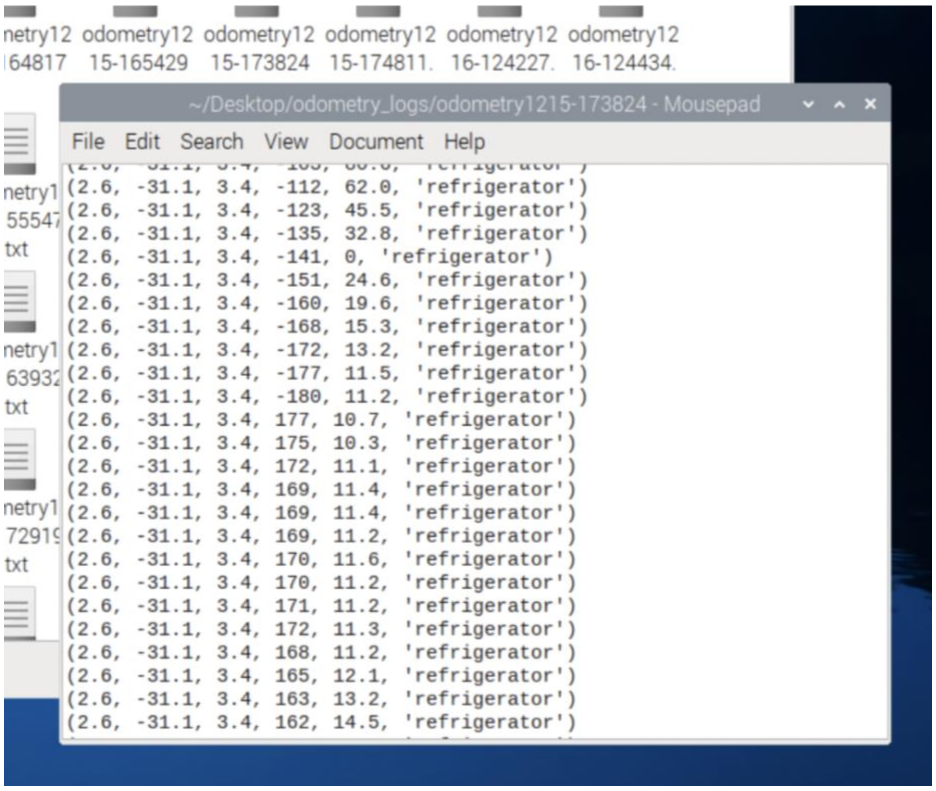
Now that all the sensors are either facing the same direction or perpendicular to each other, a program can be designed for validation of the framework.

Results

The performance of the system can be determined by two factors: the difference between actual and calculated user position, and the distance between the system's predicted object location and the true location. To visualize this data, the positioning data from the odometry and object localization algorithms were exported, as shown in Figure 6, and a parser program was created using PyGame.

Figure 6

Example odometry and object localization data export



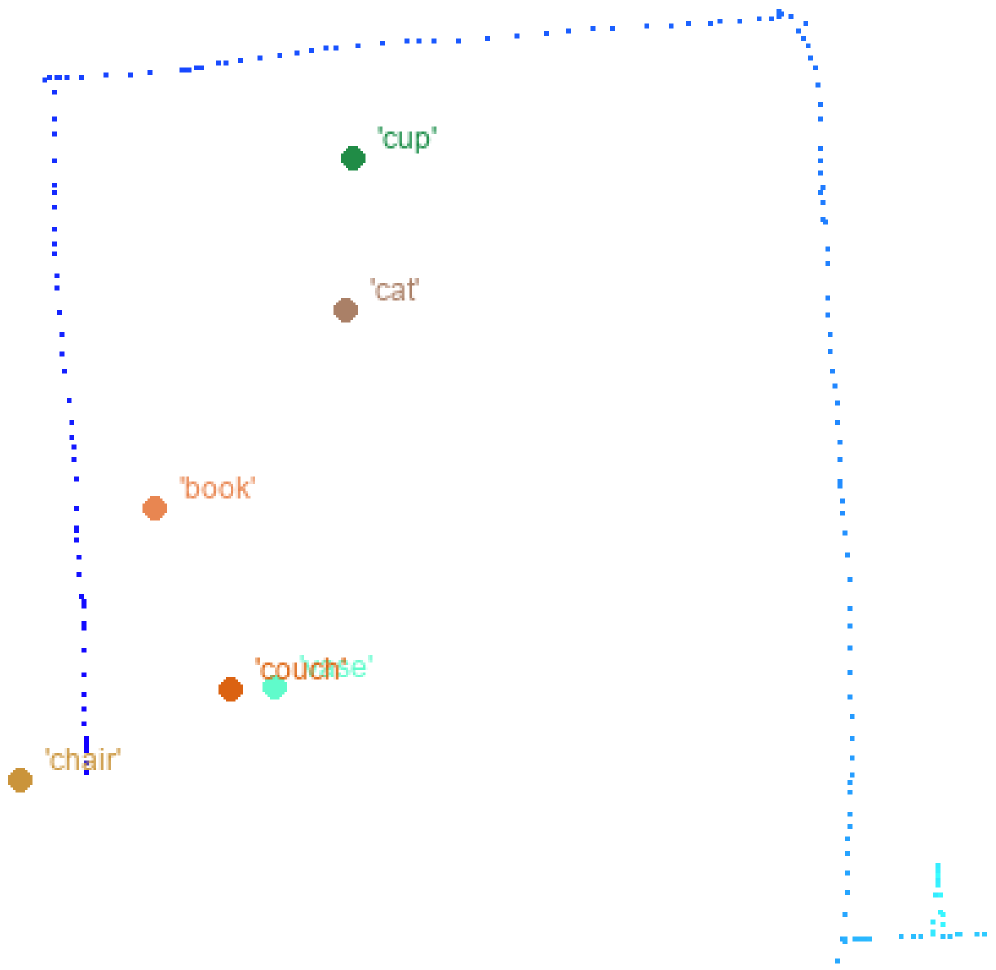
```
(2.6, -31.1, 3.4, -112, 62.0, 'refrigerator')
(2.6, -31.1, 3.4, -123, 45.5, 'refrigerator')
(2.6, -31.1, 3.4, -135, 32.8, 'refrigerator')
(2.6, -31.1, 3.4, -141, 0, 'refrigerator')
(2.6, -31.1, 3.4, -151, 24.6, 'refrigerator')
(2.6, -31.1, 3.4, -160, 19.6, 'refrigerator')
(2.6, -31.1, 3.4, -168, 15.3, 'refrigerator')
(2.6, -31.1, 3.4, -172, 13.2, 'refrigerator')
(2.6, -31.1, 3.4, -177, 11.5, 'refrigerator')
(2.6, -31.1, 3.4, -180, 11.2, 'refrigerator')
(2.6, -31.1, 3.4, 177, 10.7, 'refrigerator')
(2.6, -31.1, 3.4, 175, 10.3, 'refrigerator')
(2.6, -31.1, 3.4, 172, 11.1, 'refrigerator')
(2.6, -31.1, 3.4, 169, 11.4, 'refrigerator')
(2.6, -31.1, 3.4, 169, 11.4, 'refrigerator')
(2.6, -31.1, 3.4, 169, 11.2, 'refrigerator')
(2.6, -31.1, 3.4, 170, 11.6, 'refrigerator')
(2.6, -31.1, 3.4, 170, 11.2, 'refrigerator')
(2.6, -31.1, 3.4, 171, 11.2, 'refrigerator')
(2.6, -31.1, 3.4, 172, 11.3, 'refrigerator')
(2.6, -31.1, 3.4, 168, 11.2, 'refrigerator')
(2.6, -31.1, 3.4, 165, 12.1, 'refrigerator')
(2.6, -31.1, 3.4, 163, 13.2, 'refrigerator')
(2.6, -31.1, 3.4, 162, 14.5, 'refrigerator')
```

Note. The data is in the form (x distance, y distance, z distance, yaw, ToF reading, object)

Three controlled test environments were created with different objects and lighting conditions and the autonomous object localization algorithms were run 20 times in the same path on each of the environments. Figure 7 shows the result for a trial in one of these environments, with the small blue dots indicating odometry path, and the colored circles indicating the location of detected objects, and Table 2 shows the odometry and object localization statistics.

Figure 7

Example visualization of calculated path and detected objects



Note. Some objects (such as the cat and vase) were mistakenly identified.

Table 2

Statistics for odometry (path) mapping and object localization accuracy

System	Mean Absolute Error (MAE)	Root Mean Squared Error (RMSE)	Standard Deviation (SD)	False Positive Rate
Odometry (Path)	0.057m drift/m	0.093m drift/m	0.078m drift/m	N/A
Object Localization	0.19m	0.207m	0.083m	21.2% (87/411)

The MAE of 0.057m drift/m for the odometry indicates that most of the time, the path mapping is within a 6% margin of error, although there are times where the final pose can vary by as much as 10% as indicated by the RMSE and SD of 0.093m drift/m and 0.078m drift/m, respectively. This could be attributed to the lack of key points in some frames, such as when facing a blank white wall or an area with very few objects. For object localization, the MAE is 0.19m, which is sufficient for every day object finding, and is more reliable than the path mapping, with an RMSE closer to the MAE of 0.207m and a low SD of 0.083m. One issue with the system is that it has a relatively high false positive rate for object detection (21.2%), often positioning objects that do not exist, such as the nonexistent cat and vase shown in Figure 7. This could be attributed to using the nano YOLO model instead of the larger, more accurate models, or because the images are being downsized to 320x320 for faster processing. Another reason could be the patterned carpet that is in the test environment, which is being mistaken for a cat.

Conclusion

This project presents a functional and feasible implementation of a real-time object localization framework to assist individuals with cognitive impairments. Novel algorithms including oblique cone tracking were developed, and other established algorithms such as visual odometry were adapted to address this issue. Empirical testing showed its ability to localize objects with sufficient accuracy (often within 20cm), identify them correctly almost 80% of the time, while maintaining a high enough framerate for accurate spatial positioning, often within only a 6% drift error. Unlike existing solutions, which are often bulky and expensive, this system prioritizes affordability and portability with a price of under $200, which make it a unique candidate for scalable deployment in settings from homes to healthcare facilities.

Further Recommendations

While this study demonstrates a working framework, there are notable areas for future improvement. This study implemented visual odometry with corrections from a ToF sensor, but incorporating IMU data into a visual-inertial odometry (VIO) framework could improve positional accuracy. Although, to achieve this, a more powerful computing device than a Raspberry Pi 5 would be needed to maintain a high enough framerate for accurate key point mapping. In addition, this system currently employs YOLO, which is trained on the COCO dataset, which is limited to 80 objects. In future iterations, the ImageNet dataset can be used to expand the knowledge of the system to 1000 objects.

References

Adafruit. (n.d.). Adafruit/adafruit_circuitpython_vl53l1x: CircuitPython module for interacting with the VL53L1X distance sensor. GitHub. https://github.com/adafruit/Adafruit_CircuitPython_VL53L1X

Camera Calibration. OpenCV. (n.d.). https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html

Kitt, B., Rehder, J., Chambers, A., Schönbein, M., Lategahn, H., & Singh, S. (2011). *Monocular visual odometry using a planar road model to ...* Robotics Institute Carnegie Mellon University. https://www.ri.cmu.edu/pub_files/2011/9/ECMR2011.pdf

Simek, K. (2013). Dissecting the camera matrix, part 3: The intrinsic matrix ←. https://ksimek.github.io/2013/08/13/intrinsic/

Nielsen, N. (2022). *LiveCameraTrajectory*. GitHub. https://github.com/niconielsen32/ComputerVision/tree/master/LiveCameraTrajectory

Nister, D., Naroditsky, O., & Bergen, J. (2004). Visual Odometry | IEEE conference publication | IEEE xplore. IEEE Xplore. https://ieeexplore.ieee.org/document/1315094

ORB (Oriented FAST and Rotated BRIEF). OpenCV. (n.d.). https://docs.opencv.org/4.x/d1/d89/tutorial_py_orb.html

Rummaging, hiding and hoarding behaviors. Alzheimer's Association. (2017). https://www.alz.org/media/greatermissouri/rummaging_hiding_and_hoarding_behaviors.pdf

STMicroelectronics. (2024). *Datasheet - VL53L1X*. https://www.st.com/resource/en/datasheet/vl53l1x.pdf

Syamal, S., Huang, C., & Petrunin, I. (2024). Enhancing object detection and localization through multi-sensor fusion for Smart City Infrastructure. 2024 IEEE International Workshop on Metrology for Automotive (MetroAutomotive), 41–46. https://doi.org/10.1109/metroautomotive61329.2024.10615661

What is dementia? symptoms, types, and diagnosis | National Institute on Aging. National Institute on Aging. (n.d.). https://www.nia.nih.gov/health/alzheimers-and-dementia/what-dementia-symptoms-types-and-diagnosis

World Health Organization. (n.d.). *Dementia*. World Health Organization. https://www.who.int/news-room/fact-sheets/detail/dementia