Depth Panorama Generation on an iOS application for Fall Prevention

Katelyn Elisa Chen, Ken Wang, and Monroe Kennedy III
Department of Mechanical Engineering, Stanford University
Assistive Robotics and Manipulation Laboratory

Introduction to the SmartBelt for Trajectory Prediction

Falls caused by external perturbation have the possibility of being prevented through the prediction of a person's path using information on past trajectory, torso motion, and surrounding scene (Fig. 1).

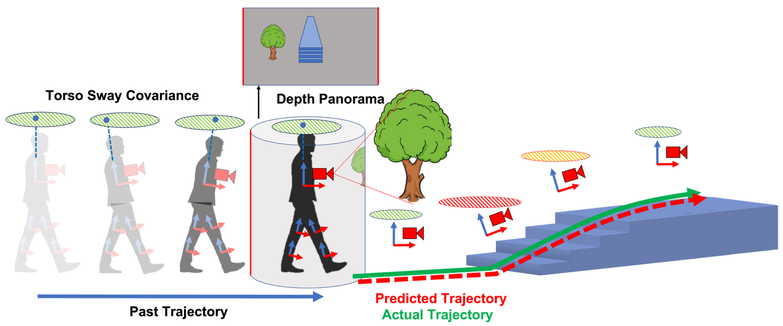

Figure 1: A person's predicted path and movement can be determined using information on path, motion, and environment

- The Sway Covariance Ellipse is a new proposed metric that quantifies the frequency of pose adjustment while walking. Frequent adjustments correspond to higher fall risk [1].
- The surrounding scene is represented by a depth panorama that stores information about the environment as point clouds.
- Using visual data from the panorama, the prediction model generates the predicted trajectory that a person will take (Fig. 2).

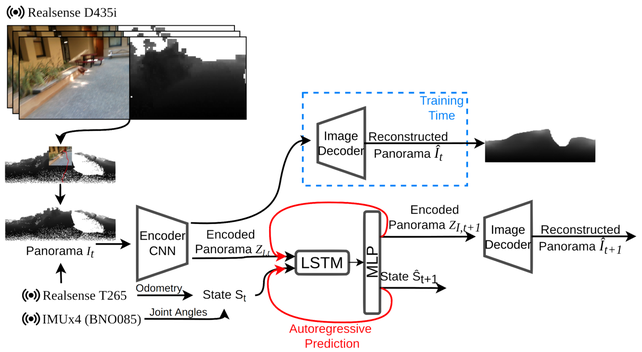

Figure 2: The prediction model uses a variational autoencoder to encode past trajectories and a long short-term memory network to predict future paths

Optimizing the Data Collection Setup

The original physical setup for the real-world walking data collection study consisted of two Intel Realsense cameras, four Inertial Measurement Units (IMUs), a step counter, and a laptop (Fig. 3).

The unwieldy setup was counterproductive to fall prevention, so the use of a phone was proposed. A single iPhone is capable of collecting all the necessary data given the following prerequisites:

- LiDAR capabilities to accurately detect depth data
- Compatibility with Apple's ARKit

Figure 3: Original setup with two Realsense cameras (green) and four IMUs (red)

Transforming Depth Data to 3D Space

Scene depth information from the LiDAR sensor is stored as a CVPixelBuffer by ARKit. To match the depth values, Z, to their corresponding positions in 3D space, the following transformations are applied:

- Pixel coordinates to positions in the camera frame using the inverse of the LiDAR intrinsic parameters, which is scaled from the camera intrinsic parameters using the resolution ratio $k_{x,y}$:

$$\begin{bmatrix} X_C \\ Y_C \\ Z \end{bmatrix} = \lambda K_{lidar}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}, \text{ where } K_{lidar} = \begin{bmatrix} f_x/k_x & 0 & \sigma_x/k_x \\ 0 & f_y/k_y & \sigma_y/k_y \\ 0 & 0 & 1 \end{bmatrix} \text{ and } \lambda = Z$$

- Positions in the camera frame to the world frame:

$$\begin{bmatrix} X_W \\ Y_W \\ Z \\ 1 \end{bmatrix} = {}^{W}_{C}T \begin{bmatrix} X_C \\ Y_C \\ Z \\ 1 \end{bmatrix}$$

The positions in the world frame are used for visualization in ARKit. For use in the depth panorama, further transformations are needed.

Visualizing the Point Cloud with ARKit

The unprojected depth data correspond to real positions in 3D space. Using SceneKit, a rendering framework integrated with Swift APIs and ARKit, the points are displayed as a "particle system" and overlaid on the camera view (Fig. 4). The point cloud session updates at 60 FPS and can be toggled to display depth-based color (Fig. 5).

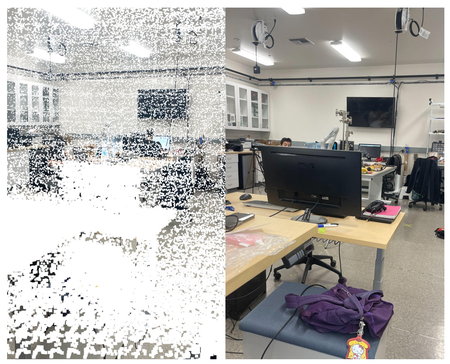

Figure 4: Points are placed in 3D space using depth data (left: point cloud view, right: camera view)

Figure 5: Point cloud colored based on depth

Adapting Python Code to Swift and ARKit

The original panorama generation code was written with a coordinate frame (Fig. 6) different from Swift's (Fig. 7) so an additional rotation is required.

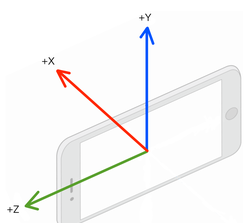

Figure 6: Python coordinate frame

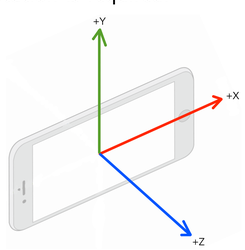

Figure 7: Swift coordinate frame (landscape right in ARKit)

Generating the Panorama

The previously gathered 3D position values are used to generate the depth image, as shown in Fig. 8.

- The points are transformed back to world frame using the camera pose of the current panorama frame (point gathering and panorama generation run on different threads).
- A surround image of shape 180x360x1 is built from the panorama projection algorithm.
- The last channel represents RGBA values such that points fade to white as depth increases.

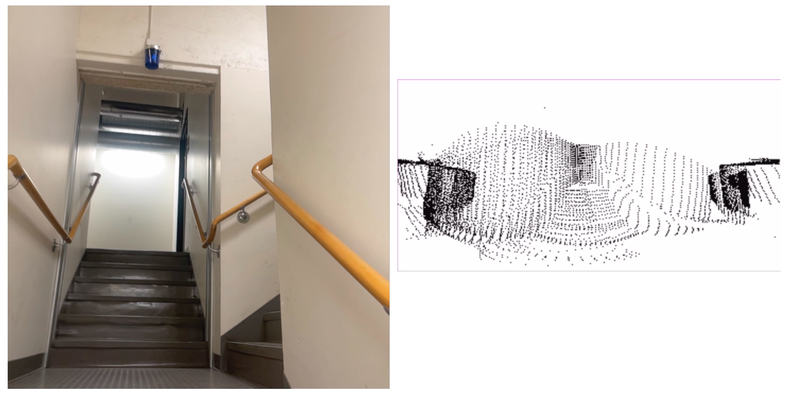

Figure 8: Video frame (left) and the corresponding depth image (right)

PyTorch iOS Deployment and Next Steps

Using PyTorch Mobile, the trajectory prediction model will be deployed to the iPhone and run in real time using the new panorama as the visual input.

- The states to be monitored include time, position, sway covariance, step count, camera pose, and velocity.
- Data logging methods must be explored to determine which states and training data to store.
- Predictions from the phone deployment will be compared to the original model to confirm result correspondence.
- The chest harness for the phone will be updated for increased comfort, stability, and usability.

Scan for video documentation:

Acknowledgements

This work is made possible by the ARMLab and ME SURI program. Thank you to Ken Wang and Monroe Kennedy III for their guidance and mentorship, Shivani Guptasarma and Rafael Sonderegger for their crash courses on robotics, Ahmed Muhammad for his assistance and support, and all ARMLab members for the welcoming community.

References

[1] Weizhuo Wang, Michael Raitor, Steve Collins, C. Karen Liu, and Monroe Kennedy III au2. Trajectory and sway prediction towards fall prevention, 2023.