PART 1

PROCEDURE

Step 1. Data Acquisition and Initialization

- Download the example data (TUM RGB-D benchmark).
- Create an imageDatastore object to examine the RGB images.
- Initialize the 3D world points and establish initial ORB feature point correspondence using matchFeatures between images.
- Utilize homographic transformation and fundamental matrix for map initialization.
- Apply triangulation to determine 3D point locations between image pairs.

Step 2. Key Frame and Map Point Storage:

- Store key frames and map points using imageviewset and worldpointset respectively.
- imageviewset stores key frames while worldpointset stores 3D map point positions.
- Initialize the place recognition database and perform loop detection using the bag of words approach.
- Build a database for loop closure based on ORB features.

Step 3. Refinement and Tracking:

- Refine and visualize the reconstructed image using bundleAdjustment.
- Perform tracking for every frame:

    1. Extract ORB features for each new frame and match them with features in the last key frame.
    2. Estimate camera pose using the Perspective-n-Point algorithm.
    3. Project map points observed by the last key frame into the current frame and search for feature correspondences.
    4. Refine camera pose using motion-only bundle adjustment.
    5. Project local map points into the current frame, refine camera pose again.
    6. Determine if the current frame is a new key frame, proceed to Local Mapping if so.

Step 4. Local Mapping:

- Perform local mapping for every key frame.
- Update map points' attributes observed in at least 3 key frames.
- Add new features using triangulation and adjust local bundles to refine frame poses.

Step 5. Loop Closure Detection:

- Detect and close loops by finding frames similar to the current one.

- Determine relative pose between the loop candidate and the current frame.
- Update relative pose and add loop connection.
- Perform a similarity pose graph over the essential graph to correct drift.

Step 6. Comparison with Ground Truth:

- Compare optimized camera trajectory with ground truth from the provided .txt file.
- Optionally, calculate root mean square error of trajectory estimates for further evaluation.

RESULT

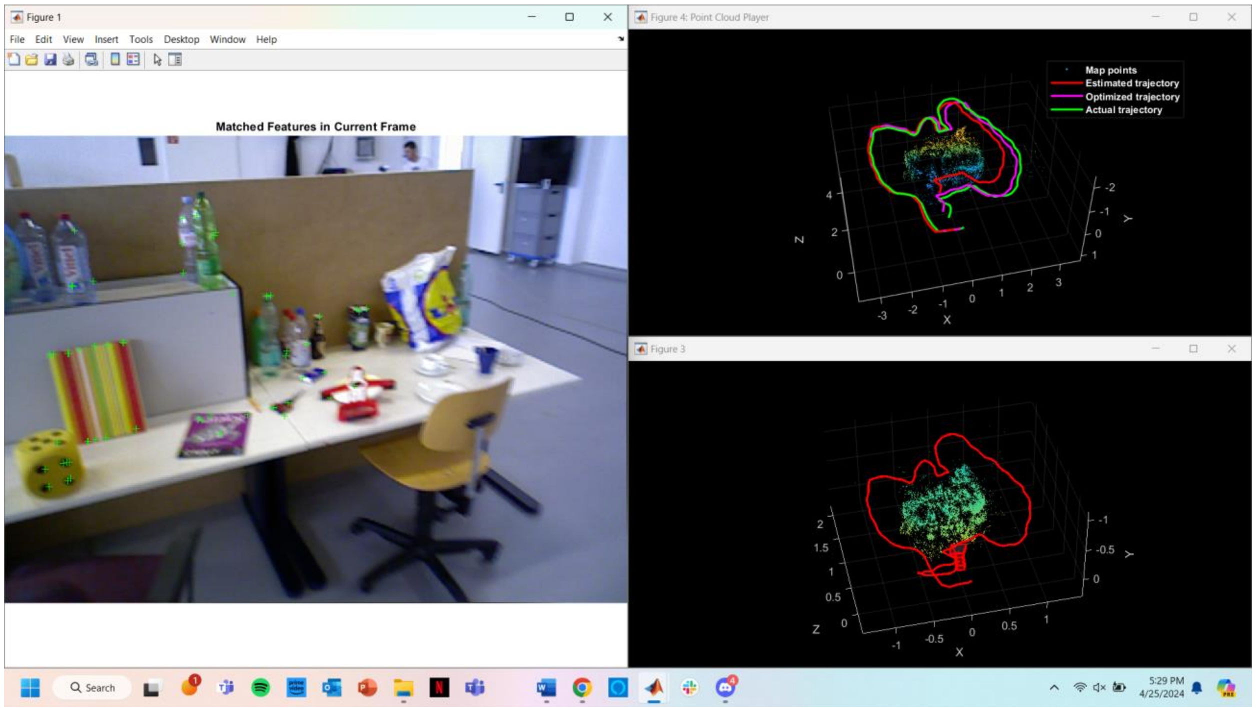


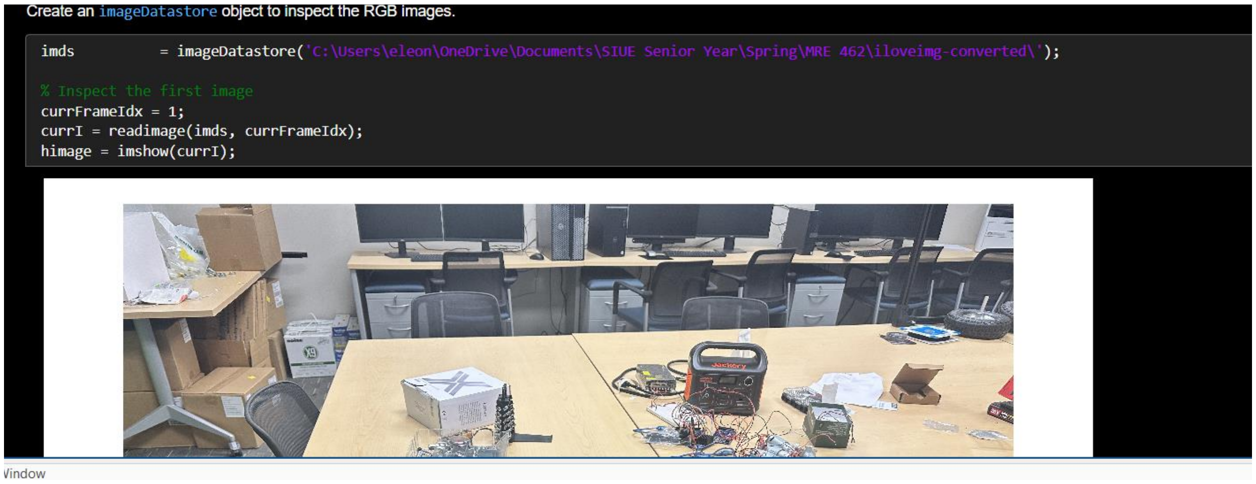
Figure 1
File Edit View Insert Tools Desktop Window Help
Matched Features in Current Frame
Figure 4: Point Cloud Player
Map points
Estimated trajectory
Optimized trajectory
Actual trajectory
X
Y
Z
Figure 3
Search
5:29 PM
4/25/2024

IMPLEMENTATION

The first step I would use in implementing a visual-SLAM application is to choose an appropriate framework, such as ROS or OpenCV. Next, adjust the camera's calibration and look for features in a series of frames. Start the system by setting the map's initialization and estimating the camera's pose. Triangulate new features, optimize structure and camera poses, and update the map continuously. Use feature matching and bundle adjustment to estimate the position of the camera in real time with respect to the map. Closing loops will fix drift. For robustness, incorporate odometry or IMU data. Analyze outcomes visually and assess system performance. Scalability and efficiency should be prioritized.

PART 2

I was able to read into MATLAB the folder containing the images

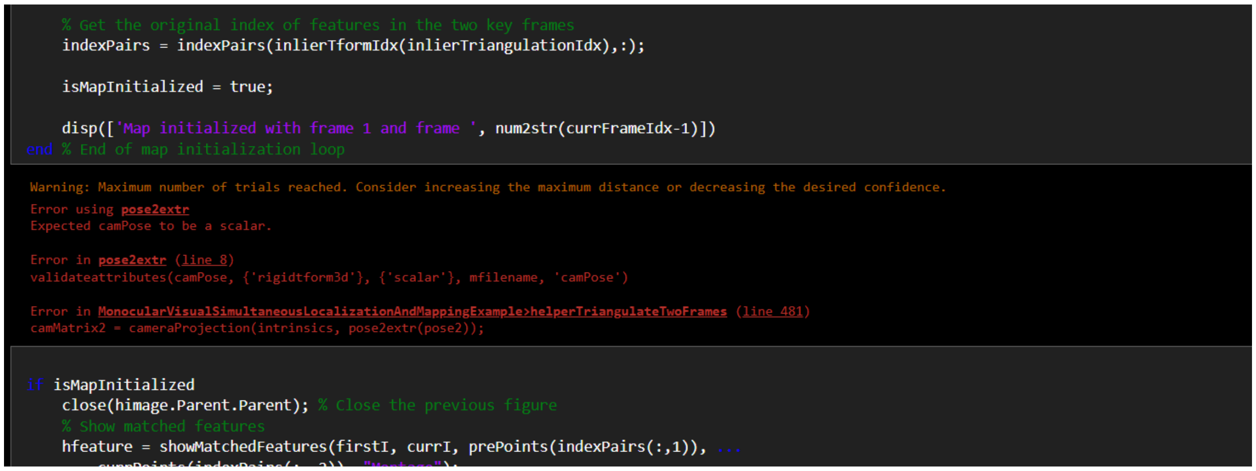


Create an imageDatastore object to inspect the RGB images.

```
imds          = imageDatastore('C:\Users\eleon\OneDrive\Documents\SIUE Senior Year\Spring\MRE 462\iloveimg-converted\');

% Inspect the first image
currFrameIdx = 1;
currI = readimage(imds, currFrameIdx);
himage = imshow(currI);
```

I ran into issues with the code not being optimized for my dataset and it would not proceed



```
    % Get the original index of features in the two key frames
    indexPairs = indexPairs(inlierTformIdx(inlierTriangulationIdx),:);

    isMapInitialized = true;

    disp(['Map initialized with frame 1 and frame ', num2str(currFrameIdx-1)])
end % End of map initialization loop
```

```
Warning: Maximum number of trials reached. Consider increasing the maximum distance or decreasing the desired confidence.
Error using pose2extr
Expected camPose to be a scalar.

Error in pose2extr (line 8)
validateattributes(camPose, {'rigidtform3d'}, {'scalar'}, mfilename, 'camPose')

Error in MonocularVisualSimultaneousLocalizationAndMappingExample>helperTriangulateTwoFrames (line 481)
camMatrix2 = cameraProjection(intrinsics, pose2extr(pose2));
```

```
if isMapInitialized
    close(himage.Parent.Parent); % Close the previous figure
    % Show matched features
    hfeature = showMatchedFeatures(firstI, currI, prePoints(indexPairs(:,1)), ...
```