Aerial Robotics Kharagpur

Help luna ! (task - 1)

Sandeepkumar, 24MA10009

***Abstract*— This work addresses the problem of depth perception for a robot by generating a depth map from stereo images. The implementation calculates depth by analyzing disparities between corresponding points in left and right camera images, effectively reconstructing the depth map and understanding the 3d environment for the robot luna.**

Using the given images i computed the disparity matrix by matching the template on the same horizontal line. The featureful points in the matrix were saved and the depth was calculated using the relation (the depth is inversely proportion to the disparity).

Once the depth was calculated, the matrix was plotted using opencv to create a depth map which helps luna to visualize the 3d environment from the 2d images provided. FIELDS OF USE : Autonomous Vehicles-to navigate through the 3d environement 3D Modeling and Scanning:for generating 3d models from stereo images

I. INTRODUCTION

The problem is to help a robot named Luna, equipped with two parallel cameras, understand its surroundings. Luna is unsure whether to move forward due to objects in front of her. The task is to develop a Python code that takes images from both cameras and generates a depth map visualizing the distance of objects, with closer objects appearing red and farther ones blue. breif it even smaller

Approach: The basic idea behind solving the problem was to get the good points and calculate the disparity map.To do the same i used the Shi thomasi and Harris algorithm built into opencv to select the points.This attempt failed because the number of points was extremely low and unable to plot a dense depth with it. The next approach was computing the disparity matrix and then clipping of the points with less disparity as they tend to be flat regions . The problem is the computation time is very high (30-40 seconds)

II. Problem Statement

This should cover one full column of page.

The task focuses on enabling a robot, Luna, to navigate its environment effectively. Luna, equipped with two cameras, struggles to process visual information for maneuvering, requiring a system to ”teach her to see.” Given stereo images from Luna’s cameras (”left.png” and ”right.png”), the objective is to develop a Python code that generates a depth map, a visual representation of the environment’s depth. This depth map will use red for closer objects and blue for farther ones. The implementation should primarily involve creating the depth map generation from scratch, using basic library functions for support, by measuring the shift of elements between the images. The final output includes the depth map (”depth.png”), code, and a detailed report.

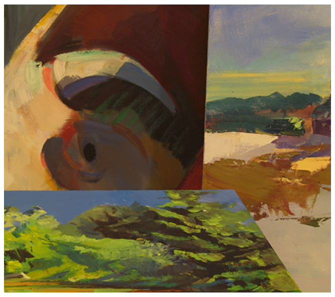


Fig. 1. righ.png

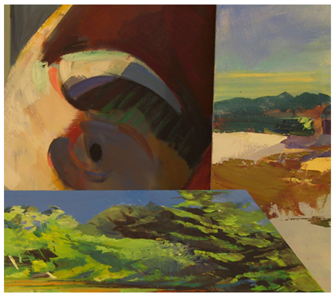


Fig. 2. left.png

III. INITIAL ATTEMPTS

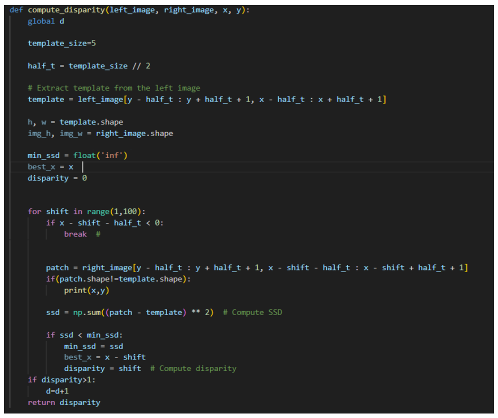
Shi thomasi: Using shi thomasi edge detection the left image i got 7,764 points and my tuning parameter it went upto 10,254 points.however the total points is 1,63,000.this type of edge detection detects very few points and tend to be meaningless in this case.However this can be used in other fields such as optical path and field path detection using lucas kannada algorithm.

Harris edge detection: The problem remains the same.This leads to the need of proper pixel detecting algorithm using which points can be selected and the integrity of data remains.

Window size: Intuitively large window size is bound to give better result.however using this resulted in loss of small features and edge boundaries leading to illogical depth maps.Also the computational cost for using 7*7 matrix window is very high and takes about 120 secs.// // // //

IV. Final Approach

1) Libraries used - numpy and opencv
   Initially, the two given images are read, and a function for disparity is coded.
2) For computing disparity matrix, a window of size 5 is taken for each pixel and matched with another window in the same horizontal axis. The shift in distance is then stored in (i, j)th position of the disparity matrix.



```python
def compute_disparity(left_image, right_image, x, y):
    global d

    template_size=5

    half_t = template_size // 2

    # Extract template from the left image
    template = left_image[y - half_t : y + half_t + 1, x - half_t : x + half_t + 1]

    h, w = template.shape
    img_h, img_w = right_image.shape

    min_ssd = float('inf')
    best_x = x
    disparity = 0


    for shift in range(1,100):
        if x - shift - half_t < 0:
            break  #


        patch = right_image[y - half_t : y + half_t + 1, x - shift - half_t : x - shift + half_t + 1]
        if(patch.shape!=template.shape):
            print(x,y)

        ssd = np.sum((patch - template) ** 2)  # Compute SSD

        if ssd < min_ssd:
            min_ssd = ssd
            best_x = x - shift
            disparity = shift  # Compute disparity
    if disparity>1:
        d=d+1
    return disparity
```

Fig. 3. Disparity

3) After computing the disparity matrix, the points with extremely low disparity are clipped off as these points tend to be flat or featureless. I chose the clipping point to be (2-100) after detailed analysis with percentiles and plotting depth map multiple times.

4) The depth was computed using the formula:

$$Z = \frac{B \times f}{d}$$

Where:

- **Z**: Depth of the point.
- **B**: Distance between cameras (baseline).
- **f**: Camera focal length (in pixels).
- **d**: Horizontal difference of corresponding points in the images (disparity).

But however, we only need the relative depth for computing the depth map, and hence we assume $B = f = 1$. We note that depth varies from 0-0.49 but however the important thing to note is that 93rd percentile is at 0.16 so the points above that are of little use and hence i am clipping at a suitable point.

```
depth=np.clip(depth,0,0.14)
```

5) using this ,the depth map was normalised using minmax normalisation and plotted using the following commanda

```
colored_depth = cv.applyColorMap(normalized_depth, cv.COLORMAP_JET)

kernel = np.ones((3,3), np.uint8)
dilated_depth = cv.dilate(colored_depth, kernel, iterations=1)
print(d)
cv.imshow("Depth Map", colored_depth)
```

6)minmaxnormalisation -A suitable normalisation method were the essence of the data isnt lost and only converted to a different scale. $x_{normalized} = \frac{x - x_{min}}{x_{max} - x_{min}}$

V. Results and Observation

The depth map represents the distance of objects from the robot. In this representation:

- Objects that are **close** to the robot are depicted in **red**.
- Objects that are **far** from the robot are depicted in **blue**.

This color scheme allows for a clear and intuitive visualization of depth, where the transition from red to blue indicates increasing distance.

comparision between my approach and the opencv predefined stereo function

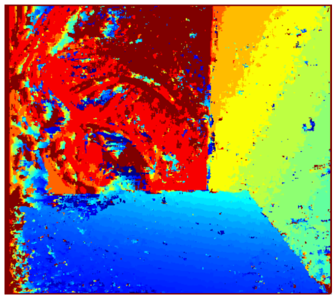


Fig. 4. manual implementation

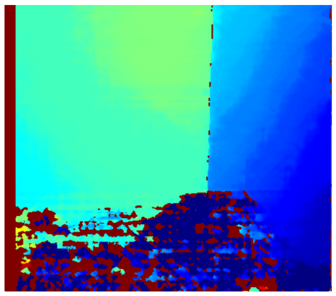


Fig. 5. stereobgm after parameter tuning

we can clearly see that the implementation done manually involves many trial or error and hence has much better accuracy.

However,it is to be noted that the depth map is not properly distributed as my approach is mainly based on trial and error and proper clipping.there is 3 different images and each have difference range for depth.Clipping at a any points results in loss of accuracy over the other images.

VI. Future Work

article

Limitations and Future Work

The resulting depth map failed to accurately depict the relative distances of objects and exhibited significant disturbances. This is largely attributed to the constraints under which the code was developed, namely:

- The code was written in Python from scratch.
- Very few optimized library functions were utilized.

Scope for Improvement:

The results obtained have great scope for improvement. Several limitations were encountered during this task, and the lack of inbuilt functions presented a significant drawback.

Specifically:

- An attempt was made to create new methods for selecting good feature points using eigenvalues, but this approach did not yield satisfactory results.
- The input images exhibited excessive noise and blurring, which negatively impacted the depth map quality.

Future Directions:

To improve the depth map generation, the following methods and approaches could be explored:

- Implementing logarithmic solutions for depth map computation.
- Investigating more effective methods for selecting feature-rich points.
- Using gausian blur in the image to reduce the noise.

i will be working on this in the coming days and work on improving the depth map.

CONCLUSION

This problem is key for robotics and ARK in general.Understanding the 3-d environment plays a vital role in computer vision.My work may not be a absolute solution or thereby anywhere even close. But trying out new approaches will eventually lead us to better solutions. In this problem i might have tried many things which may or may not have worked but trying new approaches contributes to the ongoing research and development in this vital area

References

[1] Khatib, O., ”Real-time Obstacle Avoidance for Manipulators and Mobile Robots,” International Journal of Robotics Research, Vol. 5, No. 1, pp 90-98, 1986.

[2] Write all the papers and links you have referenced to complete your project.One example for format is given above, Format followed should be ::

[3] Author Names, ”Paper Name”, Conference / Journal where the paper was published , Year of Publication