Problem Set 4, Due: Sunday March 12th 2023

Bryan Olisaemeka Mbanefo
Computer Science
bmbanefo@stanford.edu

1 Extended Kalman Filter with a Nonlinear Observation Model (30 points)

Consider the scenario depicted in Figure 1 where a robot tries to catch a fly that it tracks visually with its cameras. To catch the fly, the robot needs to estimate the 3D position $\mathbf{p}_t \in \mathbb{R}^3$ and linear

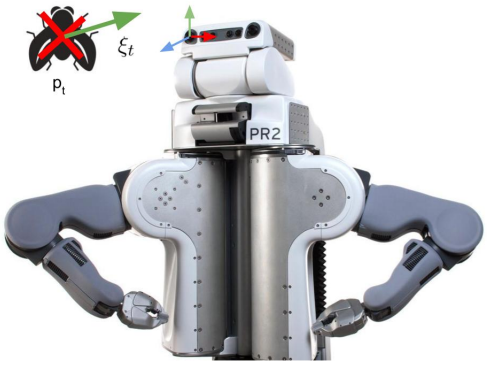


Figure 1

velocity $\xi_t \in \mathbb{R}^3$ of the fly with respect to its camera coordinate system. The fly is moving randomly in a way that can be modelled by a discrete time double integrator:

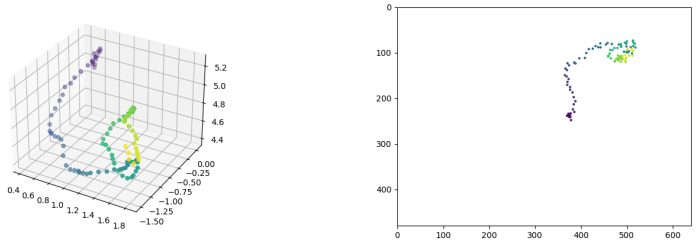
$p_{t+1} = \mathbf{p}_t + \Delta t \xi_t$
$\xi_{t+1} = 0.8\xi_t + \Delta t \mathbf{a}_t$ where the constant velocity value describes the average velocity value over Δt and is just an approximation of the true process. Variations in the fly's linear velocity are caused by random, immeasurable accelerations $\mathbf{a}_t$. As the accelerations are not measurable, we treat it as the process noise, $\mathbf{w} = \Delta t \mathbf{a}_t$, and we model it as a realization of a normally-distributed white-noise random vector with zero mean and covariance Q: $\mathbf{w} \sim N(0, Q)$. The covariance is given by $Q = diag(0.05, 0.05, 0.05)$

The vision system of the robot consists of (unfortunately) only one camera. With the camera, the robot can observe the fly and receive noisy measurements $\mathbf{z} \in \mathbb{R}^2$ which are the pixel coordinates (u, v) of the projection of the fly onto the image. We model this projection mapping of the fly's 3D location to pixels as the observation model h: $z_t = h(\mathbf{x}_t) + \mathbf{v}_t$

where $\mathbf{x} = (\mathbf{p}, \xi)^T$ and $\mathbf{v}$ is a realization of the normally-distributed, white-noise observation noise vector: $\mathbf{v} \sim N(0, R)$. The covariance of the measurement noise is assumed constant and of value, $R = diag(5, 5)$.

We assume a known 3x3 camera intrinsic matrix: K = $\begin{bmatrix} 500 & 0 & 320 & 0 \\ 0 & 500 & 240 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

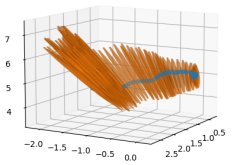
1. Let $\Delta t = 0.1s$. Find the system matrix A for the process model, and implement the noise covariance functions (Implement your answer in the $system\_matrix$, $process\_noise\_covariance$, and $observation\_noise\_covariance$ functions in Q1.py). **[5 points]** see code submission
2. Define the observation model h in terms of the camera parameters (Implement your answer in the $observation$ function in Q1.py). **[5 points]** see code submission
3. Initially, the fly is sitting on the fingertip of the robot when it is noticing it for the first time. Therefore, the robot knows the fly's initial position from forward kinematics to be at $\mathbf{p}_0 = (0.5, 0, 5.0)^T$ (resting velocity). Simulate in Python the 3D trajectory that the fly takes as well as the measurement process. This requires generating random acceleration noise and observation noise. Simulate for 100 time steps. Attach a plot of the generated trajectories and the corresponding measurements. **[5 points]**



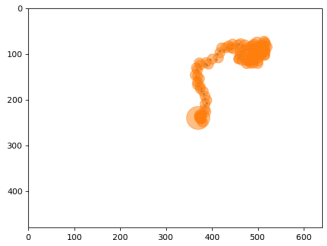
(a) Simulated Trajectory (b) 2D Measurements

Figure 2: Trajectory and 2D Measurements

4. Find the Jacobian H of the observation model with respect to the fly's state $\mathbf{x}$. (Implement your answer of H in function $observation\_state\_jacobian$ in Q1.py.)**[5 points]** see code submission
5. Now let us run an Extended Kalman Filter to estimate the position and velocity of the fly relative to the camera. You can assume the aforementioned initial position and the following initial error covariance matrix: $P_0 = diag(0.1, 0.1, 0.1)$. The measurements can be found in `data/Q1E_measurement.npy`. Plot the mean and error ellipse of the predicted measurements over the true measurements. Plot the means and error ellipsoids of the estimated positions over the true trajectory of the fly. The true states are in `data/Q1E_state.npy` **[5 points]**



(a) Mean and Error Ellipsoids of the estimated positions over the true trajectory of the fly.



(b) Mean and Error Ellipse of the predicted measurements over the true measurements

Figure 3: Mean and Error Ellipsoids

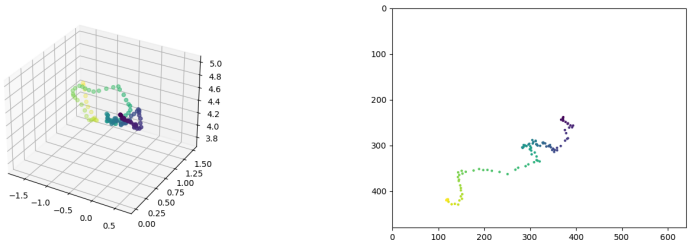
6. **Discuss the difference in magnitude of uncertainty in the different dimensions of the state. [5 points]** From figure 3 we can see that the uncertainty in the x and y dimensions is much smaller than in the z dimension, this means that our measurement model is losing information about the z dimension during 3D to 2D transformation. This phenomenon introduces ambiguities when determining the fly's position in the z dimension which causes uncertainty to increase as time progresses.

2 From Monocular to Stereo Vision (30 points)

Now let us assume that our robot got an upgrade: Someone installed a stereo camera and calibrated it. Let us assume that this stereo camera is perfectly manufactured, i.e., the two cameras are perfectly parallel with a baseline of $b = 0.2$. The camera intrinsics are the same as before in Question 1.

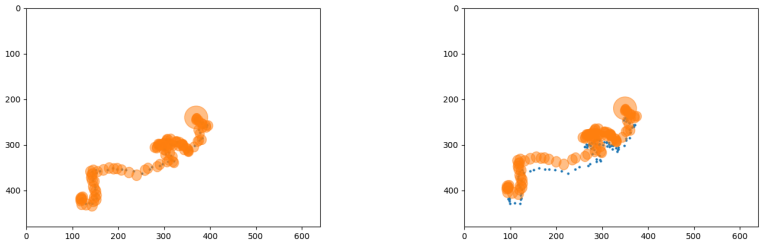
Now the robot receives as measurement $\mathbf{z}$ a pair of pixel coordinates in the left image (u^L, v^L) and right image (u^R, v^R) of the camera. Since our camera system is perfectly parallel, we will assume a measurement vector $\mathbf{z} = (u^L, v^L, d^L)$ where d^L is the disparity between the projection of the fly on the left and right image. We define the disparity to be positive. The fly's states are represented in the left camera's coordinate system.

1. Find the observation model h in terms of the camera parameters (Implement your answer in function $observation$ in Q2.py). **[6 points]** see code submission
2. Find the Jacobian H of the observation model with respect to the fly's state x. (Implement H in function $observation\_state\_jacobian$ in Q2.py) **[6 points]** see code submission
3. What is the new observation noise covariance matrix R? Assume the noise on (u^L, v^L), and (u^R, v^R) to be independent and to have the same distribution as the observation noise given in Question 1, respectively. (Implement R in function $observation\_noise\_covariance$ in Q2.py). **[6 points]** see code submission
4. Now let us run an Extended Kalman Filter to estimate the position and velocity of the fly relative to the left camera. You can assume the same initial position and the initial error covariance matrix as in the previous questions. Plot the means and error ellipses of the predicted measurements over the true measurement trajectory in both the left and right images. The measurements can be found in `data/Q2D_measurement.npy`. Plot the means and error ellipsoids of the estimated positions over the true trajectory of the fly. The true states are in `data/Q2D_state.npy` Include these plots here.**[6 points]**



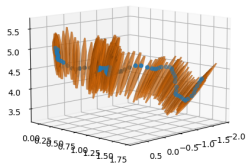
(a) Simulated trajectory of the fly. (b) True trajectory of the fly

Figure 4: Fly Trajectories



(a) Left Image Mean and Error Predicted Ellipses. (b) Right Image Mean and Error Predicted Ellipses

Figure 5: Mean and Error Predicted Ellipses



(a) Mean and Error Predicted Ellipsoid.

Figure 6: Ellipsoid

5. In this Question, we are defining $\mathbf{z} = (u^L, v^L, d^L)^T$. Alternatively, we could reconstruct the 3D position $\mathbf{p}$ of the fly from its left and right projection (u^L, v^L, u^R, v^R) through triangulation and use $\mathbf{z} = (x, y, z)^T$ directly. **Discuss the pros and cons of using (u^L, v^L, d^L) over (x, y, z)! [6 points]**

   We could construct the 3D position $\mathbf{p}$ of the fly from its left and right projection (u^L, v^L, u^R, v^R) through triangulation and use $\mathbf{z} = (x, y, z)^T$ which would have some advantages over using $\mathbf{z} = (u^L, v^L, d^L)^T$. It will simplify the Kalman Filter by reducing the non-linear measurement function to a linear function. This means we can track the fly's 3D position without having to compute the Jacobian and get a linear estimate of the measurement function.

3 Linear Kalman Filter with a Learned Inverse Observation Model (20 points)

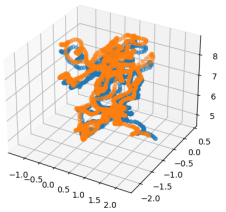
Now the robot is trying to catch a ball. So far, we assumed that there was some vision module that would detect the object in the image and thereby provide a noisy observation. In this part of the assignment, let us learn such a detector from annotated training data and treat the resulting detector as a sensor.

If we assume the same process as in the first task, but we have a measurement model that observes directly noisy 3D locations of the ball, we end up with a linear model whose state can be estimated with a Kalman filter. Note that since you are modifying code from previous parts and are implementing your own outlier detection for part C, there is no autograder for this problem - we will be grading based on your plots.

1. In the folder `data/Q3A_data` you will find a training set of 1000 images in the subfolder `training_set` and the file `Q3A_positions_train.npy` that contains the ground truth 3D position of the red ball in the image. We have provided you with the notebook LearnedObservationModel that can be used to train a noisy observation model. As in PSET 3, use this notebook with Google Colab to do this – note that you'll need to upload the data directory onto a location of your choosing in Drive first. **Report the training and test set mean squared error in your write-up. [5 points]**

   Final training loss: 0.9459299
   Final testing loss: 0.5851323

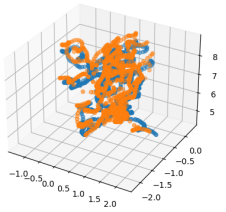
2. In the folder `data/Q3B_data` you will find a set of 1000 images that show a new trajectory of the red ball. Run your linear Kalman Filter using this sequence of images as input, where your learned model provides the noisy measurements (the logic for this is provided in `LearnedObservationModel.ipynb`). Now you can work on using the model by completing `p3.py`. Tune a constant measurement noise covariance appropriately, assuming it is a zero mean Gaussian and the covariance matrix is a diagonal matrix. **Plot the resulting estimated trajectory from the images, along with the detections and the ground truth trajectory (the logic for this is provided in the starter code). [5 points]**



(a) Estimated Trajectory (Orange), True (blue) for Q3B data

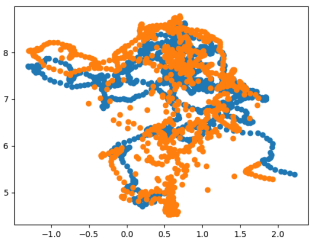
Figure 7: Estimates Q3B data vs Ground Truth

3. Because the images are quite noisy and the red ball may be partially or completely occluded, your detector is likely to produce some false detections. In the folder `data/Q3D_data` you will find a set of 1000 images that show a trajectory of the red ball where some images are blank (as if the ball is occluded by a white object). Discuss what happens if you do not reject these outliers but instead use them to update the state estimate. Like in the previous question, run your linear Kalman Filter using the sequence of images as input that are corrupted by occlusions (this is also provided in the notebook). textbfPlot the resulting estimated trajectory of the ball over the ground truth trajectory. Also plot the 3-D trajectory in 2-D (x vs. z) and (y vs. z) to better visualize what happens to your filter. **[5 points]**

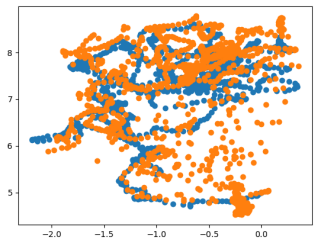


(a) Estimated Trajectory (Orange), True (blue) for Q3D data

Figure 8: Estimates Q3D Data vs Ground Truth



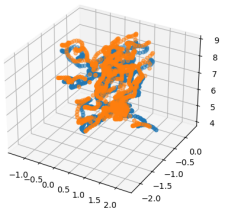
(a) 3D Trajectory in 2D X vs Z.



(b) 3D Trajectory in 2D Y vs Z

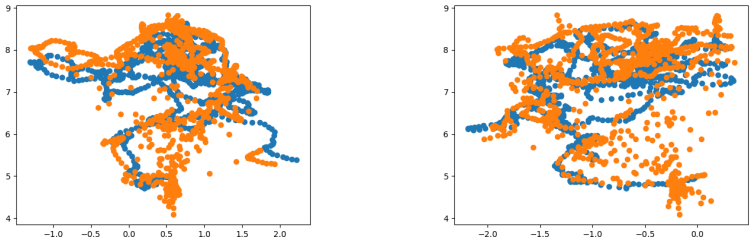
Figure 9: 3D Trajectories in 2D

4. Design an outlier detector and use the data from `data/Q3D_data`. Provide the same plots as in part c with `filter_outliers=True`. **Explain how you implemented your outlier detector and add your code to the report. Hint: Your observation model predicts where your measurement is expected to occur and its uncertainty. [5 points]**



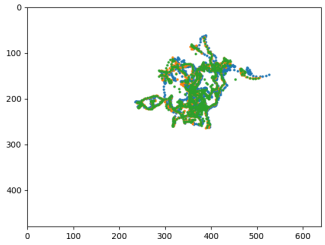
(a) Estimated Trajectory (Orange), True (blue) for Q3D data

Figure 10: Estimates Q3D Data vs Ground Truth Filter Outliers



(a) 3D Trajectory in 2D X vs Z. (b) 3D Trajectory in 2D Y vs Z

Figure 11: 3D Trajectories in 2D Filter Outliers



(a) Estimated Trajectory (Orange), True (blue) for Q3D data

Figure 12: 2D Projection of Estimated and Ground Truth Trajectory

I used Mahalanobis distance to compute the deviation. Mahalanobis distance is a measure of the distance between a point and a distribution which takes into account the covariance structure of the variables and can be used to identify outliers or to measure the similarity between two observations. If this distance is greater than a threshold (25) then it is considered an outlier and the updated state is set to the expected state.

4 Tracking with Optical and Scene flow (20 points)

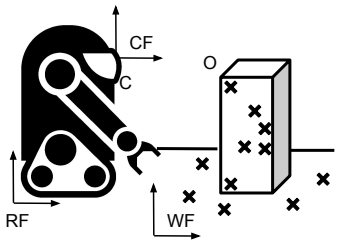


Figure 13

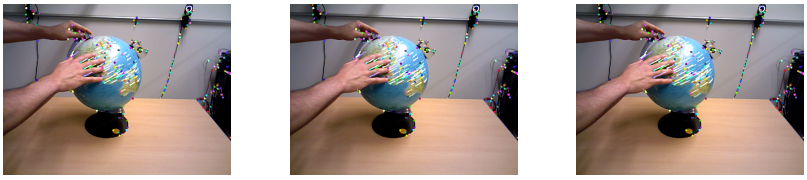
Lastly, a problem having to do with optical and scene flow. Consider the scenario depicted in Fig. 13, where an object O is being interacting by a person, while the robot observes the interaction with its RGB-D camera C. Files `rgb01.png`, `rgb02.png`, ..., `rgb10.png` in the `globe1` folder contain ten RGB frames of the interaction as observed by the robot. Files `depth01.txt`, `depth02.txt`, ..., `depth10.txt` contain the corresponding registered depth values for the same frames.

1. Detect and track the $N = 200$ best point features using Luca-Kanade point feature optical flow algorithm on the consecutive images, initialized with Tomasi-Shi corner

features. Use the opencv implementation (look at this tutorial for more information: `https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html`). For the termination criteria use the number of iterations and the computed pixel residual from *Bouget*, 2001. Use only 2 pyramid levels and use minimum eigen values as an error measure. **What is the lowest quality level of the features detected in the first frame? Provide a visualization of the flow for window sizes 15×15 and 75×75. Explain the differences observed. Is it always better a larger or a smaller value? What is the trade-off? [5 points]**

Note: You will need the opencv-python package. Please make sure that you have this installed.

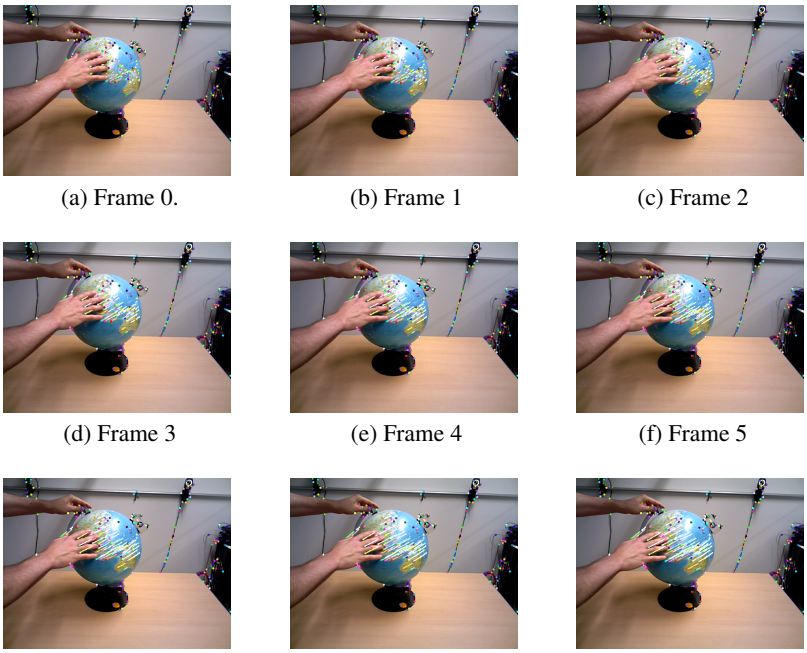
(a) Frame 0. (b) Frame 1 (c) Frame 2



(d) Frame 3 (e) Frame 4 (f) Frame 5

(g) Frame 6 (h) Frame 7 (i) Frame 8

Figure 14: Window Size 15 x 15



(a) Frame 0. (b) Frame 1 (c) Frame 2

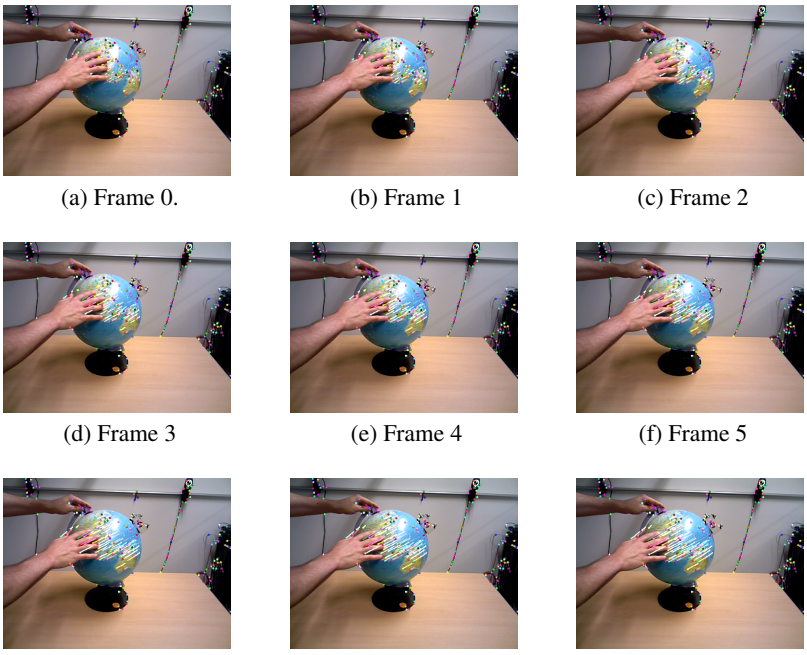
(d) Frame 3 (e) Frame 4 (f) Frame 5

(g) Frame 6 (h) Frame 7 (i) Frame 8

Figure 15: Window Size 75 x 75

- Minimum quality level: 2.0 Minimum point : [63. 2.]
- We see slightly worse accuracy with small window size of 15x15 this could be due to noise associated with small window size. Whereas we have longer computation time with window size of 75x75 but better accuracy.
- Larger or smaller value, one is not necessarily always better than the other as they are situation dependent. For example, larger window size can provide more accurate flow estimation by capturing more spatial information in the neighborhood of each pixel but this can introducing smoothing of pixels and make it difficult to detect fast motion. Smaller window size can provide more precise flow estimates by focusing on a smaller region around each pixel which is good for small objects but this can be noisy. So window size will depend on the characteristics of the scene being captured. In some cases a large window size might be better while in other cases a small window size is best.
- Tradeoff is computation-time vs accuacy

2. Use the registered depth maps and the camera intrinsics provided in the starter code to compute the sparse scene flow (i.e. 3D positions of the tracked features between frames). (Note that depth maps contain NaN values. Features that have NaN depth value in any of the frames should be excluded in the result.). **[5 points]**



(a) Frame 0. (b) Frame 1 (c) Frame 2

(d) Frame 3 (e) Frame 4 (f) Frame 5

(g) Frame 6 (h) Frame 7 (i) Frame 8

Figure 16: Sparse Scene Flow Globe

3. Now let's switch to another scenario of a person opening a book. We have the same format of data as in part(a); the ten RGB frames and the corresponding depth files are available in the `book` folder. Run the algorithm you implemented for part(a) in this new setting and provide a visualization of the flow for block sizes 75×75. **How does the result look qualititavely, compared to the visualization from part(a)? How many features are we tracking in this new scenario and how does it compare to part(a)? Give a brief explanation why it is the case.[5 points]**

    - Qualitatively there is not as much motion as in A and we also have a lower quality of features detected in the first frame Minimum quality level: 1.0 Minimum point : [578. 1.].
    - We are tracking 192 features for this section vs 200 for part A. Number of features in scene can vary based on several factors such as the complexity of the scene or the texture and contrast of objects in the scene. We have less features because the texture and contrast of the book is less than that of the globe therefore distinctive features which are more difficult to detect and track.

(a) Frame 0. (b) Frame 1 (c) Frame 2

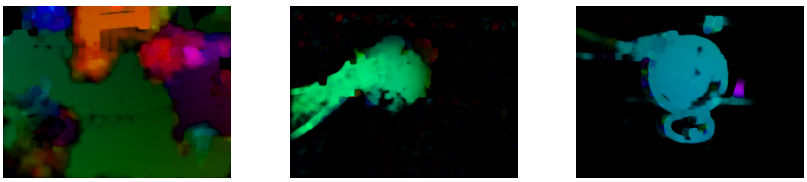
(d) Frame 3 (e) Frame 4 (f) Frame 5

(g) Frame 6 (h) Frame 7 (i) Frame 8

Figure 17: Optical Scene Flow Book

4. Next, we will be comparing dense optical flow tracking results from two different methods: Gunnar Farneback's algorithm, and FlowNet 2.0. First, run the provided p4.py script to get the dense optical flow for two frames from the set used in part b, and two frames from the Flying Chairs dataset. The resulting dense optical flow images should be saved to a file titled farneback-(imagename).png; include this in your report.

   Describe any artifacts you find between farneback-globe.png and farneback-globe2.png. Does one type of image appear have a better optical flow result? What differences between the image pairs might have caused one optical flow result to appear clearer than the other? [2 points]



(a) Chairs (b) Globe 1 (c) Globe 2

Figure 18: Farneback outputs

   - Some artifacts are noise it seems globe has a higher brightness level than the globe2 image
   - Globe 2 has a better optical flow result.
   - A major difference is the partial occlusion by the globe holders right hand in the globe 1 image. The hand covers part of the image in globe 1. Wheras in globe 2 the whole globe is visible.

5. Next, go to Google Colab and upload the included `FlowNet.ipynb` notebook. Once loading the notebook, follow the included instructions to start a GPU runtime, and upload the respective images from the `p4_data` folder used in the last step (`p4_data/globe2/rgb01.png`, `p4_data/globe2/rgb02.png`, `p4_data/globe1/rgb04.png`,

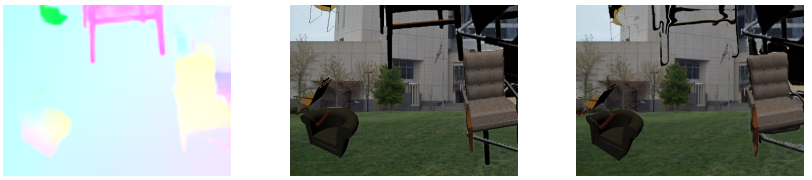
`p4_data/globe1/rgb06.png`, `p4_data/chairs/frame_chairs_1.png`, `p4_data/chairs/frame_chairs_2.png`) to the notebook files tab. Run the cells in the notebook to produce the optical flow between the pairs of images, and include these flow images (and the reconstructed images) in your report. **Note any qualitative differences you see between the optical flow results for the different pairs of images. Did one of the optical flow results and reconstructions from a specific pair of the images look less noisier than the others? Describe why you think this might be the case, and if there are any ways in which the performance on the other pairs of images could be improved with the FlowNet model. [3 points]**

(a) Optical Flow Globe 1 (b) Globe 1 Original (c) Globe 1 Reconstructed

Figure 19: Globe 1 Optical Flow Outputs

(a) Optical Flow Globe 2 (b) Globe 2 Original (c) Globe 2 Reconstructed

Figure 20: Globe 2 Optical Flow Outputs



(a) Optical Flow Chairs (b) Chairs Original (c) Chairs Reconstructed

Figure 21: Chairs Optical Flow Outputs

- Globe 2 optical flow results seems to have a lot less noise than the other 2 pairs of images.
- Globe 2 seems to have better lighting conditions which can lead to reduction in noise and the scene complexity is low. The chairs image has a high scene complexity which exposes it to more noise.
- The noise can be reduced by doing some image pre-processing, for example to adjust the brightness level to be equal across images. Another method of improvement is regularization which can be used to smooth out the flow vectors in flownet and reduce noise. This can be done by adding a penalty to the optimization function that encourages smoothness in the flow field.