1 Estimate the helicopter angles

Task 1.1

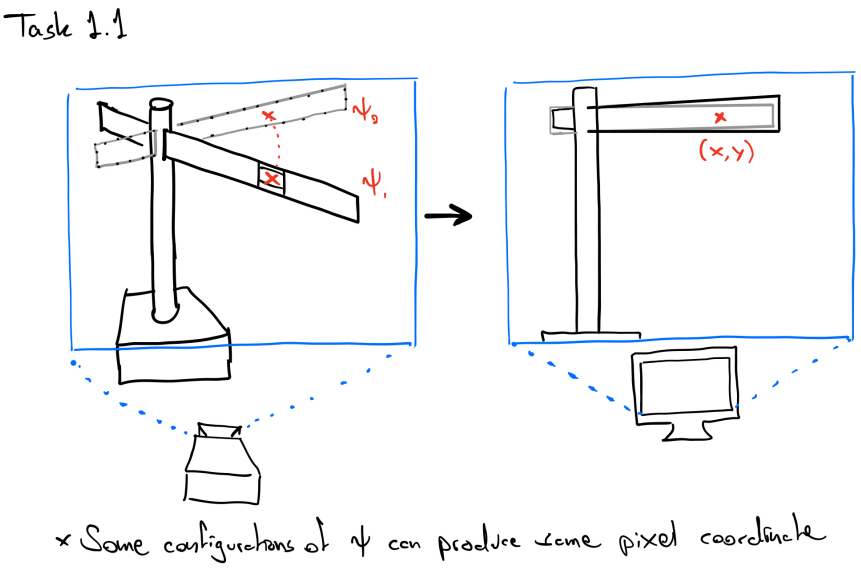


Figure 1: Projection into camera plane

As you can see in the above sketch, two points from the 3D world coordinate system can correspond to one in the camera view. This is due to the loss of one dimension (depth), what makes it impossible to distinguish single points that only differ in their distance to the camera.

Task 1.2

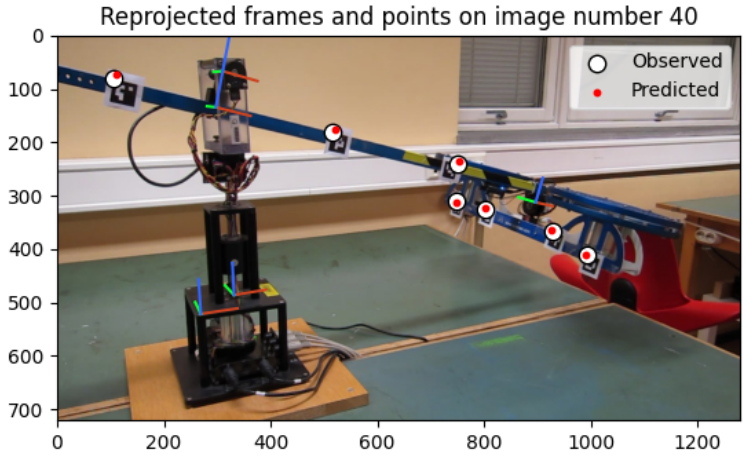
The linearization of absolute values is probably much less efficient. Calculating horizontal and vertical differences is a very trivial task and preserves the uniqueness of configurations (+-u,v).

Task 1.3

Optimal angles (residuals) on image 0:

5.24245604	4.80020124	1.19885845	-4.99317996	-4.47286417	0	0.04965743
-8.87519959	-5.02385631	-6.06002405	-2.97156827	0.43079638	0	-6.26710301

Reprojection errors at solution:
Marker 1: 10.60 px
Marker 2: 7.26 px
Marker 3: 5.06 px
Marker 4: 2.70 px
Marker 5: 1.85 px
Marker 6: 3.57 px
Marker 7: 1.90 px
Average: 4.71 px
Median: 3.57 px



Task 1.4

$\boldsymbol{p_0}$	steps	Avg.Err.	Med.Err
$\begin{bmatrix}0.0 & 0.0 & 0.0\end{bmatrix}^T$	3	4.71 px	3.38 px
$\begin{bmatrix}0.5 & 0.5 & 0.5\end{bmatrix}^T$	3	4.71 px	3.61 px
$\begin{bmatrix}1.0 & 1.0 & 1.0\end{bmatrix}^T$	4	4.71 px	3.80 px

As can be seen in the above table, the initial value $\boldsymbol{p_0}$ does not change the amount of steps nor the reprojection error significantly.

Task 1.5

We get this warning because we are trying to solve an under-determined system of linear equations what results in having infinite many solutions. The "numpy.linalg.solve()" function cannot handle this.

Task 1.6

For simply extending the dimensions $\boldsymbol{p} \in \mathbb{R}^4$ we will get a Jacobian $\boldsymbol{J} \in \mathbb{R}^{n\times 4}$ and an approximate Hessian $\boldsymbol{J}^T\boldsymbol{J} \in \mathbb{R}^{4\times 4}$. The modification of the helicopter model might result in not having unique configurations for one specific model state, e.g. ϕ and ψ are nearly indistinguishable for straight upwards pointing helicopter.

Task 1.7

(a) The maximum error is 19.4877 pixels.

(b) This maximum error occurs in image 104. The minimum error occurs in image 152.

(c) Parameter :
- Yaw: max = 0.7999 , image: 187
- Pitch: max = 0.2431, image: 327
- Roll: max = 0.0748, image: 350

The minimum errors are 0 at image 0 because of the offset synchronization we use.

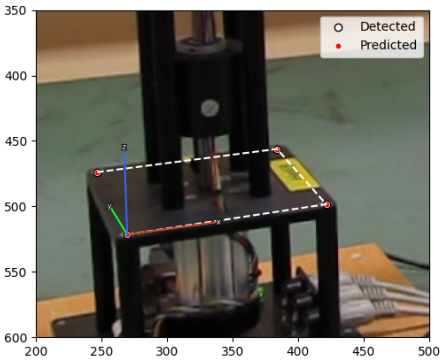
Task 1.8

(a) Because of the way the residue function has been implemented, with the weights for valid entries, the residue of unobserved markers are set to zero. Followingly, the corresponding index of the jacobian matrix J will be zero too - leading to a potentially singular matrix, which prevents solving for a unique solution. The Levenberg-Marquardt method has circumvented this problem by fixing the step length to 1 and adding a term $\mu * I$ to the normal equation. This guarantees that the equation remains solvable and that the estimation algorithm can move past the points with insufficient observational data.

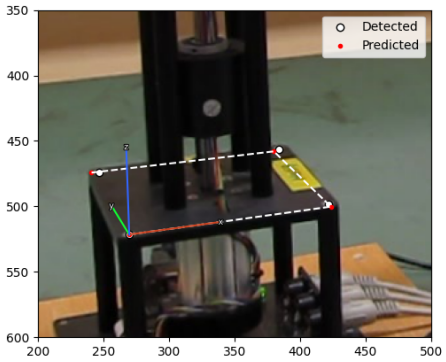
(b) The added term μI adds $\mu 1$ to the diagonal elements of $J^T J$, with μhaving a variable magnitude. The Levenberg-Marquardt method is set up in such a way that the μ term decreases when approaching the minimum, and increases otherwise. This means that the $J^T J$ term will dominate when close to the minimum, leveraging the benefits of Newton-Gauss, while having the '+1' term dominate further from the minimum - where Newton-Gauss struggles. The downside of this method, is the fixed step length of 1. While Levenberg-Marquardt is robust in ensuring that each step of the optimization remains solvable, it suffers from a sort of 'maximum accuracy' - with no guarantees beyond 'minimum + 1'. This must be taken into account when determining the stopping criterion.

2 Estimate the platform pose

Task 2.1



(a) Output equation $\tilde{\boldsymbol{u}} = \boldsymbol{K}\boldsymbol{H}\begin{bmatrix} X & Y & 1 \end{bmatrix}^T$



(b) Output equation $\tilde{\boldsymbol{u}} = \boldsymbol{K}\begin{bmatrix}\boldsymbol{R}\boldsymbol{t}\end{bmatrix}\begin{bmatrix} X & Y & 0 & 1 \end{bmatrix}^T$

In (a) we are getting all zero reprojection errors because can use the direct linear transformation $\boldsymbol{H}$ which is the most accurate we can get. Whereas in (b) we decompose this $\boldsymbol{H}$ into translation and rotation, which we have to estimate for every point after the starting point. Therefor estimation errors add up, and we loose more and more accuracy as can be nicely seen in the above figure. (The error values are: 0.000 2.798 4.286 6.168)

Task 2.2

Reprojection errors:
- all: 0.122 0.131 0.145 0.136
- mean: 0.133 px
- median: 0.133 px

The errors look much better and more equal than in 2.1 b.

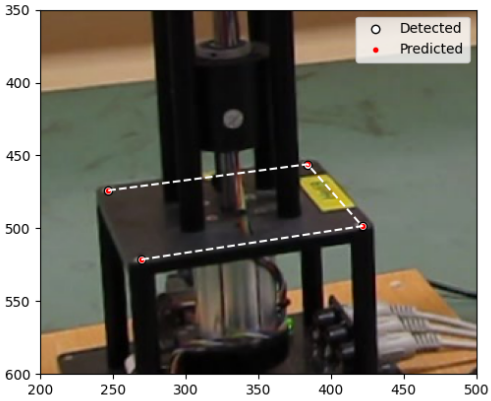


Figure 3: Pose estimation with Levenberg-Marquardt

3 Calibrate the model via batch optimization

Task 3.1

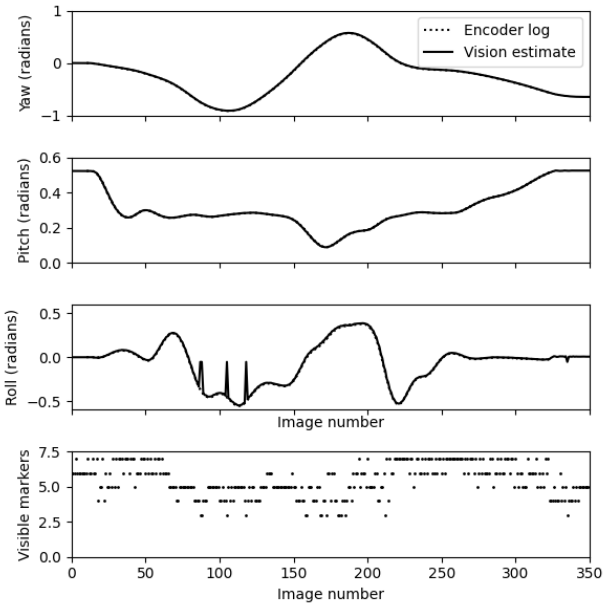


Figure 4: Plotted reprojection errors of the 3 angels and number of detected markers per image (model A)

For model A we get the following reprojection errors:
- Maximum: 1.1296 pixels
- Average: 0.1690 pixels
- Median: 0.1510 pixels
30 iterations

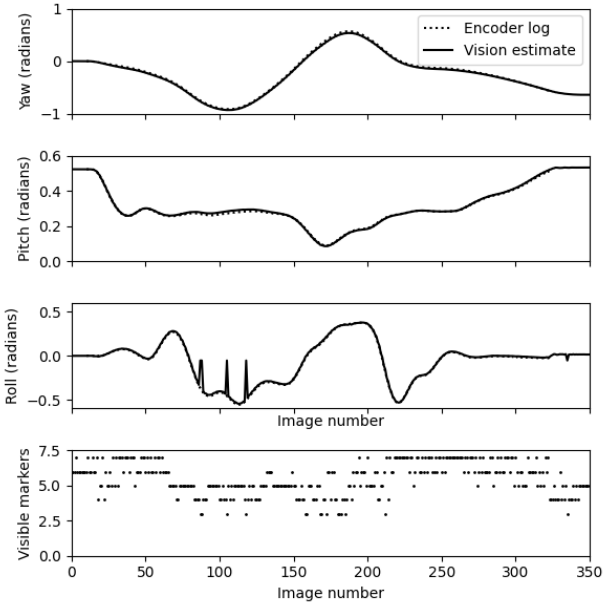


Figure 5: Plotted reprojection errors of the 3 angels and number of detected markers per image (modle B)

For model A we get the following reprojection errors:
- Maximum: 1.1199 pixels
- Average: 0.1519 pixels
- Median: 0.1328 pixels
150 iterations
For model C we get the following reprojection errors:

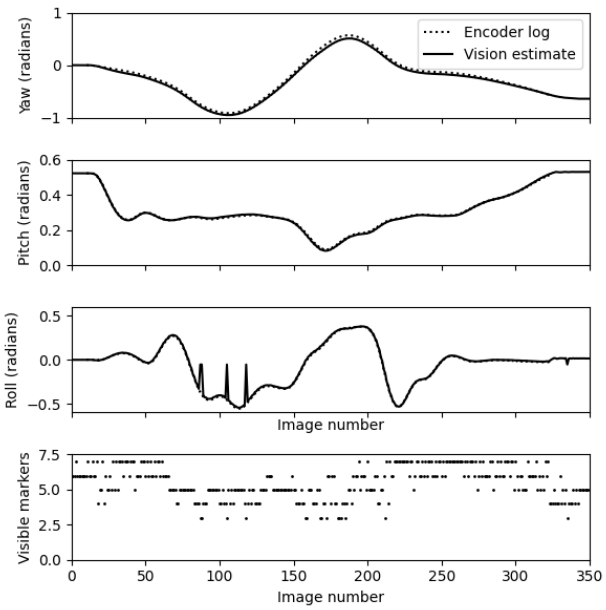


Figure 6: Plotted reprojection errors of the 3 angels and number of detected markers per image (model C)

- Maximum: 1.1152 pixels
- Average: 0.1525 pixels
- Median: 0.1327 pixels
600 iterations

Task 3.2

As a standard starting point you can always try zero-initialisation what has proven to work well enough in many cases. But if this doesn't work you can always define a simpler model and use the best estimates of that as your new starting parameters.

Task 3.3

No. If the '5cm length' had been 0, and the 32.5cm and 65cm ones were shorter, you could still end up with the same marker positions.