UNIVERSITEIT VAN AMSTERDAM

PROJECT ARTIFICIAL INTELLIGENCE

# Visual Odometry with the Ricoh Theta

*Students:*
O. MUNTEANU (10656901)
R. PRONK (10121897)

*Supervisor:*
Arnoud VISSER

February 5, 2014

**Abstract**

*In this paper, we highlight the idea of visual odometry using the Ricoh Theta camera. A brute-force based method is presented which aims at finding the best matching transformation between two consecutive frames. By computing motion fields a distance measure between frames is calculated which is used to extract the estimation of two transformations, namely rotation and translation. We show that using this method will yield good estimations for the transformations but is still too computationally expensive for real-time use.*

## 1 Introduction

There are many research papers regarding Visual odometry (VO) for omnidirectional cameras [3], however VO for a full 360 degrees spherical camera like the Ricoh Theta is still a fairly new research topic. In order to get an estimation of the ego-motion of this sensor, we study the effect of optical flow between consecutive frames given the spherical domain. Based on these motion fields we estimate two main transformations such as rotation and translation. For this we use a brute-force approach where we try to match the best transformation and so get a close enough estimation of the actual transformation.

## 2 Visual Odometry

Visual odometry[1] is the process of estimating the motion (i.e., position and orientation) of an agent (i.e., robot, vehicle, human) by analyzing various images used as inputs for the camera. The appliance of visual odometry covers several fields such as robotics, automotive, augmented reality, and wearable computing.
The visual odometry works properly when there is sufficient illumination in the environment and a static scene with enough texture to allow apparent motion to be extracted. Furthermore, consecutive frames should be captured by ensuring that they have sufficient scene overlap.
The advantage encounted by visual odometry compared to wheel odometry is that visual odometry is not affected by wheel slip when taking into consideration some adverse conditions leading to a more accurate trajectory estimates.

The problems of visual odometry concern motion drift due to the errors accumulated over time from the occurance of displacements among consecutive poses. This drift becomes evident after a few hundred meters and the results rely on the features in the environment, the resolution of the camera, the presence of moving objects such as people or vehicles, and the illumination conditions. The problem of motion drift can be solved if the agent revisits the place that has been already observed previously.
Regarding the usefulness of visual odometry, this can be used in several domains, namely GPS, inertial measurement units (IMUs), laser odometry, wheel odometry.

## 3 The Ricoh Theta

The Ricoh Theta camera[1] is a full 360 degrees spherical camera with two lenses, where the images obtained by the two lenses are automatically stitched together. We assume that the images outputted by the camera are direct polar which is also known as *spherical* or *equirectangular projection*. This assumption is based on the fact that a simple wrapping around a sphere in order to create the unit sphere (see figure 1) already provides a good result, which for now should be enough to make an estimation for the ego-motion.

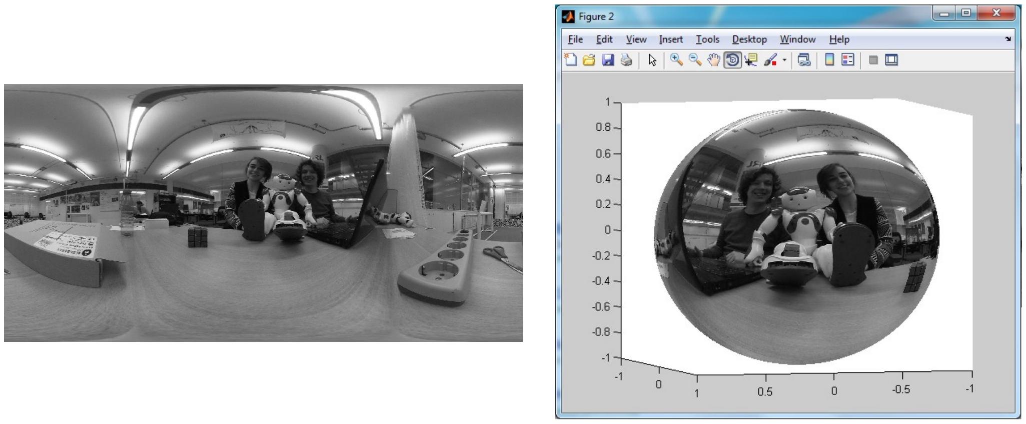


Figure 1: Simple wrapping around a sphere

Below we provide the code for wrapping an image in Matlab:

[1]https://theta360.com/en/

```
I = imread('theta1.JPG');
[x,y,z] = sphere(100);
figure;
warp(x,y,z,flipud(I));
```

### 3.1 Camera calibration

In order to obtain depth information and achieve better results, the camera needs to be calibrated[2]. At this point, however, we were unable to use pre-existing calibration toolboxes for this sensor. Due to this, we provide an visual odometry solution for this camera where calibration is not required.

## 4 Ego-motion estimation

### 4.1 Optical flow

For achieving more insights regarding the approach of visual odometry, we focus on the optical flow for obtaining the scene's motion field. The optical flow is a method that provides a measure of the apparent motion within a sequence of images.

Our optical flow is implemented using Lucas Kanade algorithm that uses gradient information obtained from consecutive frames. In order to improve the method's ability to track fast movements, Gaussian pyramids are used. When using Gaussian pyramids, we look at different layers of detail of the images made by using Gaussian smoothing and rescaling. Using these different layers also more general movements (derived from the smoothed images) can be extracted. For our computations, we use the Lucas Kanade algorithm[2] using Gaussian pyramids while taking three pyramid levels into consideration when computing the optical flow.

[2]We use an implementation written by Sohaib Khan and can be downloaded from http://crcv.ucf.edu/source/optical

## 4.2 Data-set

In order to be able to accurately test our system, we started by making a data-set consisting of basic transformations made with the Ricoh Theta camera. For testing our methods with respect to rotational transformations, we took pictures under different angle degrees alternating every 5 degrees. For translational transformations, we created a data-set for sideways and forward translations. Where for each image taken, we considered different scales interchanging every 1 cm.

## 4.3 Extracting the transformations

### 4.3.1 Rotation

When looking at the direct polar representation of the image we receive from the camera, we can see the full 360 degree vision and most importantly we can see that the horizontal axis has a range of 360 degrees (see figure 2). In our approach we exploit this property and use this property to apply simple rotational transformations on the image.

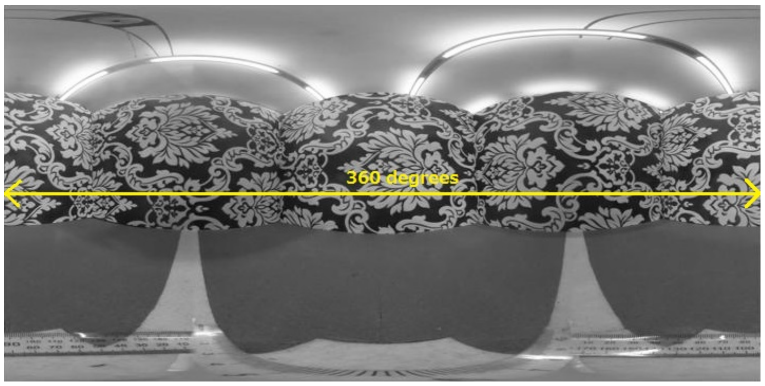


Figure 2: Range horizontal axis

In order to extract a close approximation of the rotation, we rotate the new image (applying a circular shift) and see at which degree of rotation it gives the lowest error compared to the original image. The error in this case is given by the sum of vector lengths given the motion field between the two images. For calculating this error we use an implementation of Lucas Kanade algorithm using Gaussian pyramids where for each angle $i$ assigned, we calculate the corresponding sum of the motion vector lengths. We use the build-in Matlab function *circshift* for the circular shifting of the image in the horizontal axis. Therefore, an estimation of the degree of rotation can be calculated by:

$$\textit{Shift image in horizontal direction} \quad \rightarrow \quad \frac{\text{SizeImageX}}{360} \times \text{i} \tag{1}$$

$$\textit{Calculate Error} \quad \rightarrow \quad \text{Error} = \text{sum of motion vectors lengths} \tag{2}$$

$$\textit{Degree of rotation} \quad \rightarrow \quad \text{Argmin(Error(i))} \tag{3}$$

Below we provide a part of the Matlab's code regarding the computation of errors according to the angle $i$:

```
for i = 1:10
    [u,v,~] = HierarchicalLK(image1, ...
        circshift(image2,[0 -round((size(image2,2)/360)*i)]),
        3, 4, 2, 0);
     errors(i) = sum(sum(abs(u))) + sum(sum(abs(v)));
end
```

### 4.3.2 Translation

In order to extract the translational transformations, we have to look at specific parts of the image. Therefore the first step is to extract important regions which indicate translational movement (in this case regions indicating sideways movement see figure 3).

In order to get scale of translation in the sideways direction, we take the same approach as with the rotational transformation. This time however instead of applying a circular shift to the image, we zoom in/out on the interest regions. Where we zoom in on one interest region and zoom out on the other interest region, which region to zoom in on depends on the direction of the agent. This direction can also be found by comparing the

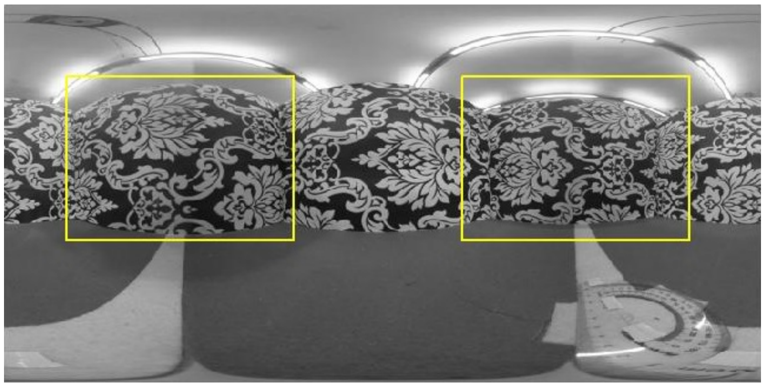


Figure 3: Important regions for sideways movement

error of zooming in and out, in the same way we do throughout this paper. To find the scale of translation, we zoom in/out with scale $i$ and see which $i$ yields the lowest error. The estimation of scale can therefore be found as:

$$\textit{Zoom in/out on interest regions} \quad \rightarrow \quad \text{Zoom with scale i} \tag{4}$$

$$\textit{Calculate Error} \quad \rightarrow \quad \text{Error = sum of motion vectors lengths} \tag{5}$$

$$\textit{Scale of translation} \quad \rightarrow \quad \text{Argmin(Error(i))} \tag{6}$$

Below we present a part of the Matlab's code regarding the computation of errors for extracting the scale of translation:

```
% Extract interest regions
IRLeft1 = image1(IRLX1:IRLX2,IRLY1:IRLY2);
IRRight1 = image1(IRRX1:IRRX2,IRRY1:IRRY2);

numrows = size(IRLeft1,1);
numcols = size(IRLeft1,2);
offset = 1;
```

```
% Try for 10 zoom magnifications
errors = zeros(1,10);
for i = 1:10
  IRLeft2 = image2((IRLX1+(i-1)*offset):(IRLX2-(i-1)*offset),
                (IRLY1+(i-1)*offset):(IRLY2-(i-1)*offset));
  IRLeft2 = imresize(IRLeft2 ,[numrows, numcols]);
  IRRight2 = image2((IRRX1-(i-1)*offset):(IRRX2+(i-1)*offset),
                (IRRY1-(i-1)*offset):(IRRY2+(i-1)*offset));
  IRRight2 = imresize(IRRight2 ,[numrows, numcols]);
  [u1,v1,~] = HierarchicalLK(IRLeft1, IRLeft2, 3, 4, 2, 0);
  [u2,v2,~] = HierarchicalLK(IRRight1, IRRight2, 3, 4, 2, 0);
  errors(i) = sum(sum(abs(u1))) + sum(sum(abs(v1))) + ...
              sum(sum(abs(u2))) + sum(sum(abs(v2)));
end
```

## 5 Results

### 5.1 Rotation

An example regarding the results obtained after implementing the rotation is given considering *image1 (original position)* and *image2 (rotated 5 degrees).*

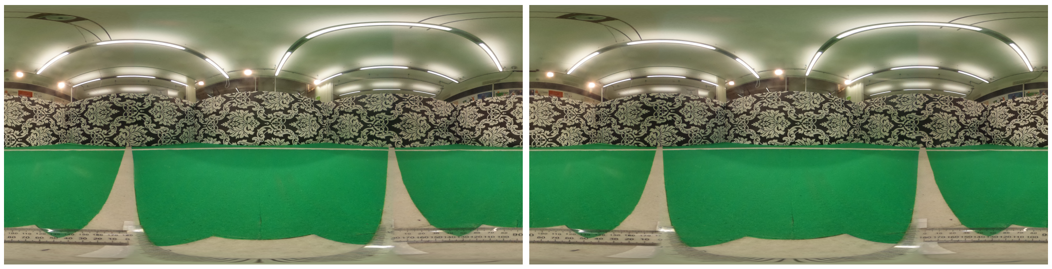


Figure 4: Image for 0 degree (Left Side) and Image for 5 degrees (Right Side)

From Figure 5 we can observe that the lowest error of the motion vectors lengths is provided by the angle i=5 degrees, which was also the degree of rotation for image2. Given this result we can see that our method is able to make close estimations of the actual degree of rotation and is in this case spot

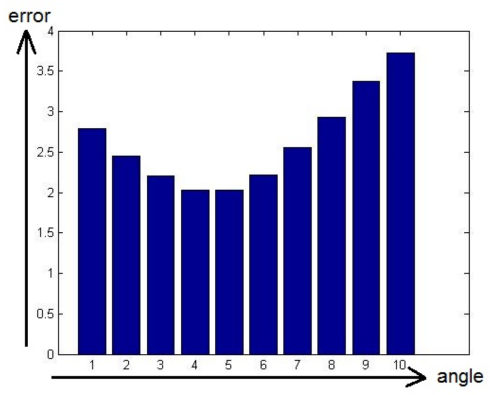


Figure 5: Errors plotted per degree of rotation

on. For a more accurate measure however, a larger resolution image should be used (now downscaled to save computational expenses) and shorter angle intervals should be checked. More results regarding the estimation of degree of rotation can be found in appendix A. Here can be seen that larger rotations do become harder for the system, however using a bigger (original) resolution for the image would yield higher accuracy given these big rotations.

## 5.2 Translation

After running the implementation for the sideways translation, we obtain the lowest error for i=5 as can be seen in figure 6. However, this scale is something relative and all we can conclude based on this is that there is a certain translation in the sideways direction. However, how many centimeter this translations actually consists of is unknown. For comparison we also added results for other scale of translations (see appendix B) and what can seen from these results is that relativity between translations is still maintained. Meaning that a bigger translation yields a bigger scale and vice versa.

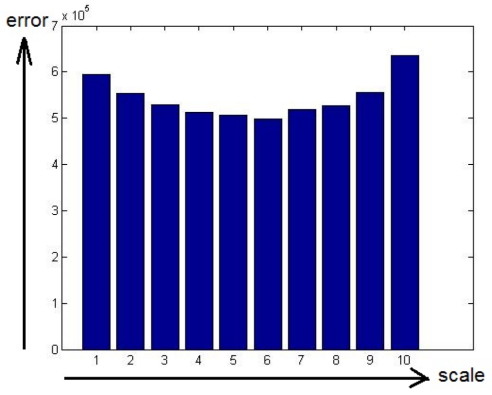


Figure 6: Errors plotted per scale of translation (sideways movement)

## 6 Conclusion

Shown is that rotational transformations can be estimated quite well, where even higher accuracy could be achieved by using higher resolution images (which were now downscaled due to the computational expenses). Also the translational transformations can be estimated but will only give relative values for the measure of translation. This relative values however, can certainly be used during the VO process as the relativity between translations is maintained. The brute-force method is however still quite computationally expensive where running this in real time is still an unfeasible task.

## 7 Future Work

With our current work we are able to estimate the degree of rotation and a relative translation measure. These estimations are however still done separately, it would be more convenient to make this into a single operation which finds the rotation and translation at the same time. Another improvement would be to change the relative translation measure to something more concrete, which could be achieved by first calibrating the camera. But also taking a more mathematical approach instead of brute-force estimation and so creating a method which can also run real-time will be a great improvement.

## References

[1] Roland Siegwart, Illah L. Nourbakhsh & Davide Scaramuzza *Introduction to Autonomous Mobile Robots, 2nd Edition.* The MIT Press, Cambridge, Massachusetts London, England. 2011.

[2] Peter Corke *Robotics, Vision and Control - Fundamental Algorithms in MATLAB*, The MIT Press, Cambridge, Massachusetts London, England. Spring.

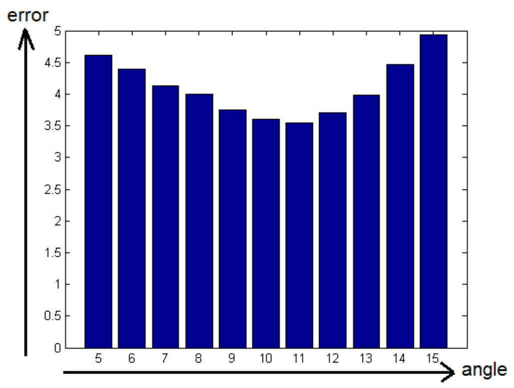
[3] Chris McCarthy, Nick Barnes & Mandyam Srinivasan *Real Time Biologically-Inspired Depth Maps from Spherical Flow*

[4] Sohaib Khan *Hierarchical Lukas Kanade*, downloaded from: *http://crcv.ucf.edu/source/optical*,2003.

## Appendix

## A Rotation

*Errors plotted per degree*



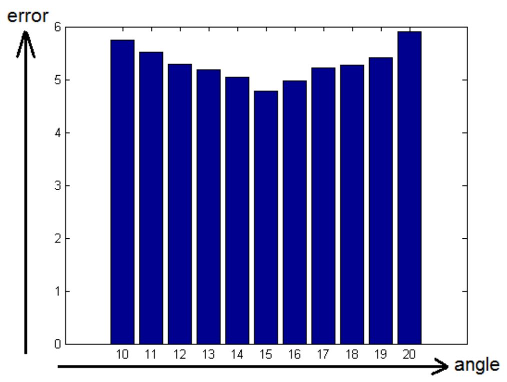
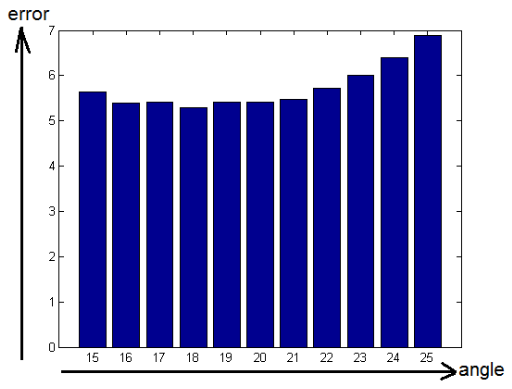


Figure 7: Testing 0-10 degrees (Left) & 0-15 degrees (Right)



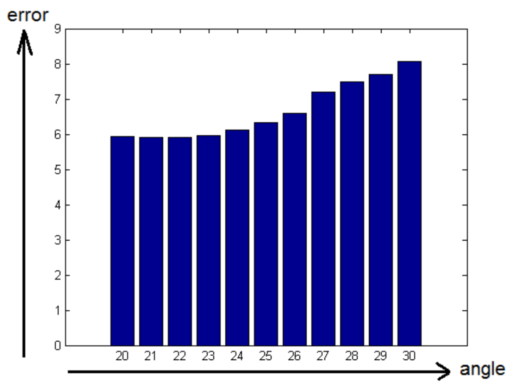


Figure 8: Testing 0-20 degrees (Left) & 0-25 degrees (Right)

## B Translation

*Errors plotted per scale*

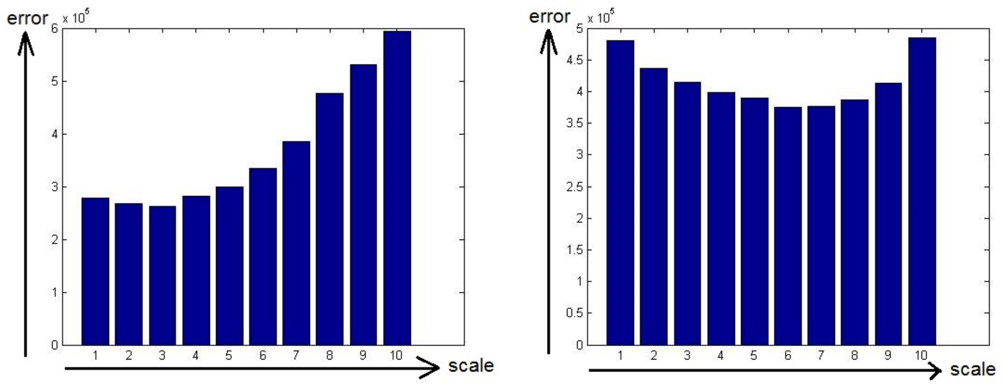


Figure 9: Testing 2 cm sideways (Left) & 4 cm sideways (Right)