A GPU PARALLEL APPROACH TO REAL-TIME DENSE RGB-D VISUAL ODOMETRY

Pedro Daniel de Cerqueira Gava, Cairo Lúcio Nascimento Júnior, Geraldo Adabo*

*Division of Electronic Engineering
Instituto Tecnológico de Aeronáutica - ITA
12228-900-São José dos Campos-SP, Brazil

Email: pdcg@ita.br, cairo@ita.br, adabo@ita.br

Abstract— This article presents a new approach to implement real-time Dense RGB-D Visual Odometry taking advantage of the computational power of a modern GPU and the kind of parallelization it offers, exploring the quantity of data that needs to be handled. Our objective is to provide a fast and accurate method to estimate the camera motion using photometric error minimization. Assuming constant intensity, the error minimized is modeled in a nonlinear fashion requiring robust iterative methods to solve it. The algorithm developed was made having in mind now everyday hardware GPU present in PCs, phones, and embedded computers. Our results show that the algorithm is robust enough to handle displacements of a camera moving with small velocities and performing real-time odometry. Also, a linear algebra library was created alongside this work to near the gap between parallel programming concepts and linear algebra with high level procedures and data structures.

Keywords— Dense Visual Odometry, RGB-D camera, Parallel Programming , GPU

1 Introduction

The usage of RGB-D cameras have become increasingly popular over the last years (Huang et al., 2017; Gupta et al., 2014; Jafari et al., 2014), and with it applications taking advantage of its properties have been constantly developed specially in robotics field for visual odometry and SLAM (Henry et al., 2014; Endres et al., 2014). In general these applications and techniques focus on the usage of the CPU of a computer to process the information although they are capable of achieving real time processing, considering a camera frame rate of 30Hz. Most of them have to give up on some precision to do it (Kerl et al., 2013). So one can say that it is a problem of computation effort, and this type of problem have always being an issue in the entertainment industry because of realistic computer graphics. Nowadays it appears also when dealing with large amounts of data which is known as Big Data (Gutiérrez et al., 2017), and in recently research of Artificial Intelligence and Deep Learning (Schmidhuber, 2015). All those problems have recurred to the usage of massive computation inside GPUs (Graphics Processing Unit) and in this paper we use this processing unit to process a dense visual odometry algorithm developed for RGB-D cameras.

Today GPU is a well established technology in present days, ranging from graphics cards for personal computers to embedded solutions that commute both CPU and GPU (NVIDIA®, 2017a), from self-driving cars to cell phones (NVIDIA®, 2017b) and not using it is a waist of a valuable resource. Not only because one could do a task faster but also because one can separates tasks that need massive and concurrent computation of elements, e.g. subtraction of an image, and tasks that are inherently serial problems which have do done in a CPU.

Applications of the parallel paradigm in RGB-D visual algorithms for movement estimation are scarce if compared to usual approaches, and our objective is to enable the motion of a camera mounted in any platform, being it automated or not, to be estimated in high frequency with accuracy, enabling usage of smaller resolutions of cameras but higher frame rate. In this paper we present

- A parallel algorithm for real-time dense RGB-D visual odometry based on (Kerl et al., 2013) implemented in a GPU;
- A library that has been a spin off of the main algorithm to straight the gap between the GPU parallel programming paradigm and the usual developer;

The code resultant from this work can be accessed in `https://github.com/Akindart/cgmapping`.

Also, in this paper it is used the CUDA™ technology, made by NVIDIA®, therefore all concepts and examples of applications described here has an implicit information that it is done with CUDA™ and compatible hardware. Section 2 describes briefly previous approachs to the problem of visual odometry using RGB-D cameras. Section 3 explain how to perform the estimation of the camera motion and presents the first version of our algorithm, and a description of how we parallelize one of the functions we use. Section 4 presents our results, comparing the frequency our algorithm executes with the algorithm based on.

2 Related Work

2.1 Dense Visual Odometry

This type of approach uses the whole image to estimate the camera motion, first presented to be used with a pair of stereo images (Comport et al., 2010). Recently such methods were proposed to RGB-D cameras (Steinbrücker et al., 2011; Tykkälä et al., 2011) using photometrical error, which can be considered a 3D version of the 2D Lucas-Kanade image alignment algorithm (Lucas et al., 1981). There is also the possibility of minimizing the geometrical error between 3D surfaces instead of photometrical error, and this problem is solved using the iterative closest point (ICP) of algorithms (Rusinkiewicz and Levoy, n.d.). One drawback of ICP is its dependency on using nearest neighbor-search, and several strategies and algorithms have been proposed to overcome this (Blais and Levine, 1995; Henry et al., 2014; Stückler and Behnke, 2012).

The key assumption of this approach is that it requires constant intensity from 3D observed from different positions. Jumps in the estimated trajectory if can occur because of occlusions, dynamic objects, and sensor noise, which problems were dealt with in (Huber, 2011; Gelman et al., 2014). Our work use the robustness presented in (Huber, 2011) and implemented also in (Kerl, 2012) to handle such problems.

2.2 Sparse Visual Odometry

Sparse visual odometry relies on obtaining signatures of point features present in a image, in some sense condensing information before processing some correspondence between them and afterwards deciding how these signatures are related to previous ones. These signatures can be created using small patches around features or using feature descriptors (Bay et al., 2008), where the later requires more computational power. The minimization of error between correspondents signatures are then carried out minimizing the reprojection error between each pair.

Visual odometry approaches have had success in controlling robotic platforms such ground vehicles (Nistér et al., 2004) and unmanned aerial vehicles (Weiss et al., 2012). But to reconstruct a rigid body motion one needs the depth of the camera points. Hence when estimating it with triangulation point depths cannot be determined in metric values, only up to a scale factor and RGB-D cameras make the problem easier by giving absolute depth. To estimate this unknown scale some authors use the measurements from IMU and EKF simultaneously (Engel et al., 2012; Weiss et al., 2012). A detailed revision on the topic can be found in (Scaramuzza and Fraundorfer, 2011; Fraundorfer and Scaramuzza, 2012).

3 The Parallel RGB-D visual odometry

Our proposed solution take advantage of the parallelism brought by modern day GPUs such as those from NVIDIA®, which enables fast and massive computation based on calculation bandwidth output rather than velocity. In this work we address the problem using NVIDIA® hardware which enables the usage of CUDA™ (NVIDIA®, 2018) and its libraries to process information in the GPU. The algorithm implemented is a parallel version of the one presented in (Kerl et al., 2013), the difference resides in how and which device performs the computation. While the former run theirs in a single core of an i5 from Intel, in ours, we use a commoner from NVIDIA®.

Before diving into implementation details, first we will present the concepts of visual odometry based on photogametry consistency with RGB-D cameras and after it is presented how we achieve computation parellelism in the GPU. The first concept is how to describe the reconstruction of movement based on the apparent brightness displacement between two images; the second is how to minimize the displacement error between images using the description previously discussed within a robust probability framework.

3.1 Movement estimation

3.1.1 Warping function

A warp function links the pixel $\mathtt{x}$ in the first image I_1 and the warped pixel in the second image I_2, see Figure 1. The first step is to write the position of a given pixel $\mathtt{x} = (u, v)^T$ in 3D as the point $\vec{p} \in \Re^3$ using the inverse of pinhole projection function π as

$$\vec{p} = \pi^{-1}(\mathtt{x},\ Z_1(\mathtt{x})) \tag{1}$$

$$= Z_1(\mathtt{x}) \left(\frac{u - c_x}{f_x}, \frac{v - c_y}{f_y}, 1 \right) \tag{2}$$

where f_x and f_y are the focal length of the camera, c_x and c_y are the optical center and $Z(x)$ is the depth of the given pixel x.

Next step is put point $\vec{p}$ written with respect to (w.w.r.t.) I_2 using rigid body transformation consisted of a rotation matrix $R \in SO(3)$ and a translation vector $\vec{t} \in \Re^3$ together, it is common to refer to both rotation and translation as the rigid body transformation $g \in SE(3)$.

$$T(g, \vec{p}) = R\vec{p} + \vec{t} \tag{3}$$

In order to find a minimal representation we use the twist coordinates $\vec{\xi} = (\nu_{1:3}, \omega_{1:3})^T \in \Re^6$, which contains linear $[m/s]$ and angular velocities

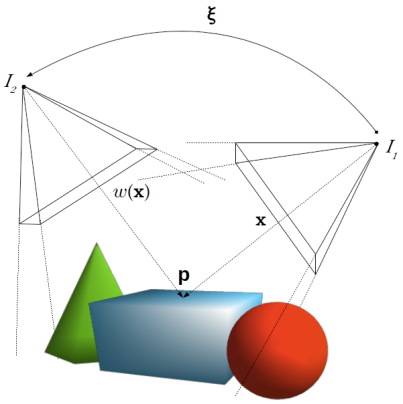


Figure 1: Graphical interpretation of the photo-consitency assumption used in this paper. Two different views of the same scene and the goal is to warp the second image so it matches the first.

$[grad/s]$, and is related to the $SE(3)$ through the Lie Algebra $\mathfrak{se}(3)$ using the exponential map

$$g(\vec{\xi}) \;=\; exp(\hat{\xi}) \tag{4}$$

where $\hat{\xi}$ is a matrix in the tangent space of $SE(3)$. If the reader has any further interest in Lie Algebra and Lie Groups, *q.v.* (Ma et al., 2012).

Last, we shall take the point $T(g,\vec{p}) = (x,y,z)^T$ which is w.w.r.t. I_2 and make use of the projection function π to find the correspondent pixel in I_2.

$$\pi(T(g,\vec{p})) \;=\; \left(\frac{f_x}{z}+c_x, \frac{f_y}{z}+c_y\right) \tag{5}$$

And finally, the warping function τ arises as the following expression

$$\tau(\vec{\xi},\mathbf{x}) \;=\; \pi(T(g(\vec{\xi}),\vec{p})) \tag{6}$$

$$\;=\; \pi(T(g(\vec{\xi}),\pi^{-1}(\mathbf{x},\; Z_1(\mathbf{x})))) \tag{7}$$

3.1.2 Robust optimization

Assuming the camera displacement is small enough we have used the photogametry difference between images as the parameter we want to minimize, i.e., we want to use the brightness residual between images in order to estimate the camera movement. Taking the residual to have the same number of elements of the images used one can define $\vec{r}(\vec{\xi}) \in \Re^{m.n}$, where m and n are the image dimension of height and width. Also $r_i(\vec{\xi}) \in \Re$ it is defined as the residual of the i-th pixel and its expression is

$$r_i(\vec{\xi}_t,\mathbf{x}_i) \;=\; I_2(\tau(\vec{\xi}_t,\mathbf{x}_i)) - I_1(\mathbf{x}_i). \tag{8}$$

To minimize the residual the Gauss-Newton method comes in play to solve it iteratively. Linearizing the residuals using first order of Taylor Expansion Series, one is confronted with

$$r_i(\vec{\xi}_t,\mathbf{x}_i) \;\approx\; r_i(\vec{\xi}_t,\mathbf{x}_i) + \left.\frac{\partial r_i(\vec{\xi},\mathbf{x}_i)}{\partial\vec{\xi}}\right|_{\vec{\xi}=0}\Delta\vec{\xi} \tag{9}$$

$$= r_i(0) + J_i(0)\Delta\vec{\xi} \tag{10}$$

where $J_i \in \Re^{(m.nx6)}$ it is the Jacobian of the residual equation and $\Delta\vec{\xi}$ is the update of the initial $\vec{\xi}$ that was used as pivot for the linearized function. The update is given by the well know formula

$$J^T W J \Delta\vec{\xi}^{(k)} = -J^T W \vec{r}(\vec{\xi}_t^{(k-1)}) \tag{11}$$

where $\vec{\xi}_t^{(k)}$ describes the k-th iteration during the estimation of $\vec{\xi}$ in instant t. $W = diag(w_1,w_2,...,w_{m.n}) \in \Re^{(m.nxm.n)}$ is a squared diagonal matrix that weights the residuals according to some criteria. If Gaussian distribution is assumed for the residuals W is the information matrix Σ^{-1}, but as have been shown by (Kerl et al., 2013), the best fitting distribution for these residuals is the t-distribution. Ergo, to calculate the weight of the residuals two parameters have to be determined, the degree of freedom ν of the t-distribution and the variance σ^2. The former is done empirically while the later is calculated iteratively until convergence and is dependent of the first. The following equations are derived in (Kerl, 2012) and presented here

$$\sigma^2_{k+1} = \frac{1}{m.n}\sum_i r_i^2 \frac{\nu+1}{\nu+\left(\frac{r_i}{\sigma_k}\right)^2} \tag{12}$$

$$w(r_i) = \frac{\nu+1}{\nu+\left(\frac{r_i}{\sigma}\right)^2} \tag{13}$$

The update of the variables describing the movement is done using sequentially the logarithm and exponential map, this operation will be written with a $\boxplus$ binary operation for the sake of simplicity.

$$\vec{\xi}_k = \vec{\xi}_t^{(k-1)} \boxplus \Delta\vec{\xi} = log(exp(\vec{\xi}_t^{(k-1)})exp(\Delta\vec{\xi})). \tag{14}$$

Using a coarse-to-fine scheme. An image pyramid is built where each layer has half of the size of the layer under it, and the estimated movement of upper layers are used as first estimation of those below. The image pyramid concept is illustrated in Figure 2.

In the absence of sufficient texture and structure the approach presented does not estimates accurately the camera motion and as was shown in (Steinbrücker et al., 2011).

3.2 The parallelization

The parallelization procedure is done analyzing the seven principal steps of the algorithm described below. Theses steps are macro procedures

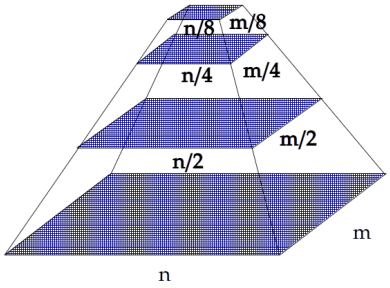


Figure 2: Illustration of the concept of an image pyramid with four layers, where each superior layer has half the dimension of the one under it.

that encapsulates previous discussed equations in 3.1.1 and 3.1.2.

- **Step 1 - Constructing the image pyramids**: In this step each pyramid can be built separately, in a sense that even if to build a coarser image of the RGB one need also depth map, the action of reading the depth map does exclude the possibility of process it separately to build its own pyramid, therefore one can write that $pyrDown_{rgb}(I_{rgb})$ and $pyrDown_{depth}(I_{depth})$ are independent of each other and can be done at the same time interval;
- **Step 2 - Warping function**: Recalling eq. 6, the warping function implicitly depends on the already treated images of **Step 1**, therefore implicating in information dependency;
- **Step 3 - Partial derivatives calculation**: Obviously, partial derivatives of the warped image in x ($I^*_{2,x}$) and y ($I^*_{2,y}$) directions can be computed separately if the source image is only read thus it can be read by two operations at the same time;
- **Step 4 - Residual calculation**: From eq. 8 it is straightforward to observe the dependency on **Step 1** and **Step 2** but not on **Step 3**, and more, **Step 3** is not affected by **Step 4** either;
- **Step 5 - Jacobian Calculation**: The Jacobian can be factored in two parts J_w and J_I. J_w is dependent of the three dimensional point of pixel $\mathbf{x}_i$ in I_1, this inverse projection procedure was carried during the calculation of the warped image I^*_2 in **Step 2**, then it is clever to cache it to be used here;
- **Step 6 - Weight calculation**: Weights are dependent of the residuals. One thing to notice is that from **Step 2**, the warped image I^*_2 may have invalid pixels and those are discarded when computing the weights;
- **Step 7 - Solve normal equations**: This is dependent of all other precedent steps.

The solution of the normal equations (cf. eq. 14) also is parallelized and it is implemented as presented in Figure 3

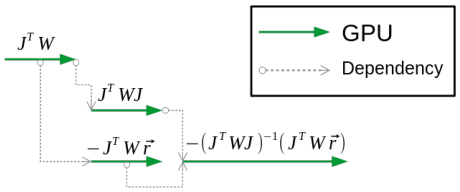


Figure 3: Parallelized normal equations.

Below, in Algorithm 1, we present a straight forward linearized version of the algorithm itself in a high level style, highlighting its main functions described previously. Also this version is a *Foward Compositional Algorithm*(Baker and Matthews, 2004) version, not requiring relinearization every iteration. IT also explains how a solution is accepted based on convergence or the reaching of maximum number of iterations. Observe that in the weighted error have to be reinitialized after every change of level of the pyramid because errors of different sizes are not comparable.

4 Results

Our results have shown that our implementation has achieved a similarly precision in the track of the camera but we have outperformed by little the original version in speed. Experiments have been done with the same datasets used by (Kerl et al., 2013) and we noticed that using simple and yet effective scheduling policies to execute the algorithm made possible to run it at 16Hz with full resolution of the camera 640x480 using double precision floating point. Table 1 shows the frequency of our algorithm.

The accuracy obtained can be seen in Figure 5, where the horizontal axis presents the the lowest level of the image pyramid used in the optimiza-

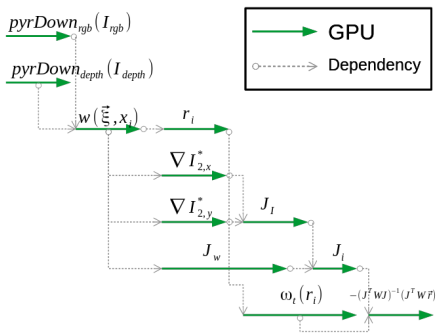


Figure 4: This image depicts the time schedule defined for the direct visual odometry algorithm as seen from a parallel perspective.

Input: Reference image I_1, current image I_2, reference depth Z_1, current depth Z_2, a convergence acceptation value ϵ, degrees of freedom ν for the t-distribution, max number of iterations it_{max}

Result: The estimated motion of the RGB-D camera in twist coordinates $\vec{\xi}^*$

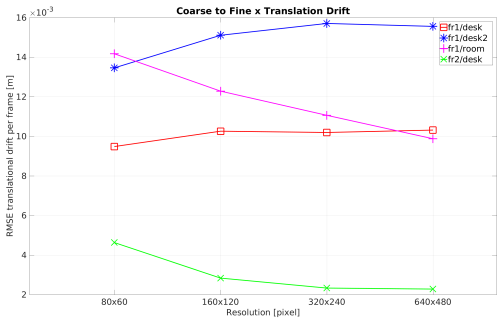
$\vec{\xi}^* \leftarrow 0$
$I_{1,pyr} = pyrDown(I_1)$
$I_{2,pyr} = pyrDown(I_2)$
$Z_{1,pyr} = pyrDown(Z_1)$
$Z_{2,pyr} = pyrDown(Z_2)$
for $(l \leftarrow 0; l > 0; l \leftarrow l-1)$ **do**
  $I_{1,l} \leftarrow I_{1,pyr}(l)$
  $I_{2,l} \leftarrow I_{2,pyr}(l)$
  $Z_{1,l} \leftarrow Z_{1,pyr}(l)$
  $Z_{2,l} \leftarrow Z_{2,pyr}(l)$
  $k \leftarrow 0$
  $error_{k-1} \leftarrow \infty$
  $error_k \leftarrow 0$
  $\Delta\vec{\xi} \leftarrow \infty$
  $\vec{\xi}_t^{(0)} \leftarrow \vec{\xi}^*$
  while $(error_{k-1} - error_k) > \epsilon \wedge k > it_{max}$ **do**
    $I^*_{2,l} \leftarrow I_{2,l}(w(\vec{\xi}_t^{(k)} \boxplus \Delta\vec{\xi}, \mathbf{x}))$
    $\vec{r} \leftarrow I^*_{2,l} - I_{1,l}$
    $\nabla I^*_{2,x,l} \leftarrow$ image_derivative_x$(I_{2,l})$
    $\nabla I^*_{2,y,l} \leftarrow$ image_derivative_y$(I_{2,l})$
    $J \leftarrow$ calculate_jacobian$(\nabla I^*_{2,x,l}, I^*_{2,y,l})$
    $\sigma^2 \leftarrow$ calculate_scale$(\vec{r}, \nu)$;
    $W \leftarrow$ calculate_weight_matrix$(\vec{r},\ \nu,\ \sigma^2)$;
    $error_{k-1} \leftarrow error_k$
    $error_k \leftarrow \frac{1}{n}\vec{r}^T W \vec{r}$
    if $(error_{k-1} - error_k) < \epsilon$ **then**
      $\Delta\vec{\xi} \leftarrow -(J^T W J)^{-1} J^T W \vec{r}$
      $\vec{\xi}_t^{(k+1)} \leftarrow \vec{\xi}_t^{(k)} \boxplus \Delta\vec{\xi}$
    end
    else
      $\Delta\vec{\xi} \leftarrow 0$
    end
    $k \leftarrow k+1$
  end
  $\vec{\xi}^* \leftarrow \vec{\xi}_t^{(k)}$;
end

tion, and the vertical axis is the RMSE of the drift per frame. The sequences *fr1/desk* and *fr1/desk2* have higher linear and angular velocities (close to) while *fr1/room* and *fr1/desk* have lower velocities. Observe that due to blur, the more levels of the image pyramid is used the poor is the estimation for faster sequences of images.

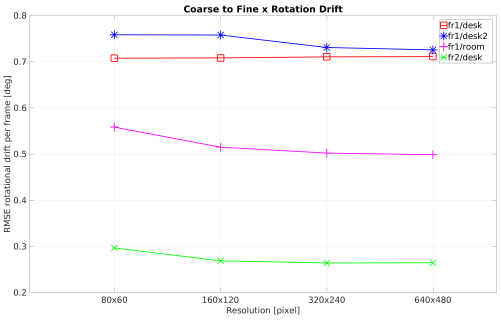
Another interesting result is a sudden gain of precision when lowering three times the magnitude of the convergence value, from 10^{-2} to 10^{-5} in all resolutions. Further diminishing in the convergence value leads to almost non improvement, see Figure 6.

Resolution [pixels]	Avg. Frequency [Hz]
80×60	79.2358
160×120	53.4852
320×240	33.2358
640×480	16.3258

Table 1: This table presents the average frequency the algorithm runs when using more than one resolution to estimate the camera motion.



(a) Coarse-to-fine VS Translational Drift

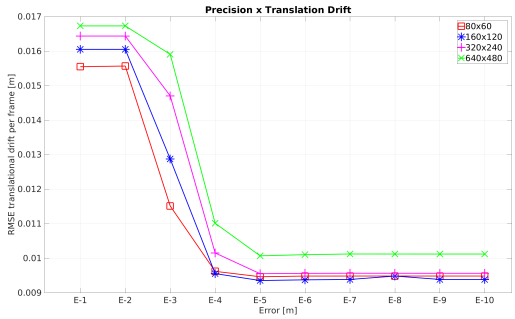


(b) Coarse-to-fine VS Rotational Drift

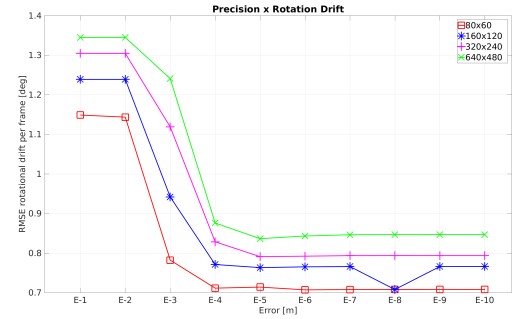
Figure 5: In (a) there is the behavior of the algorithm with respect to the translational drift over different resolutions used in the motion estimation and (b) presents the rotational drift.

Now, to determine the best suitable degree of freedom we have analyzed how it influences the drift per frame. Figure 7 shows the result and from it one can observe that the value of degrees of freedom that is fits best for all of the sequences is $nu = 5$.

From these results it is possible to determine the parameters to be used in the algorithm, being the convergence value $\epsilon = 10^{-5}$, the degree of freedom is set to $\nu = 5$ and the the level used are from 80x60 up to 320x240. For completeness purposes, the number of iterations is set to a maximum of 30, although experiments have shown that the number of iterations rarely passes the number

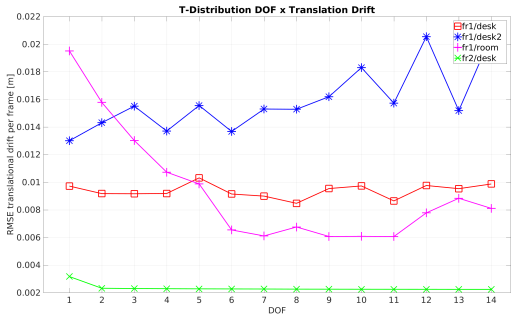


(a) Precision VS Translational Drift

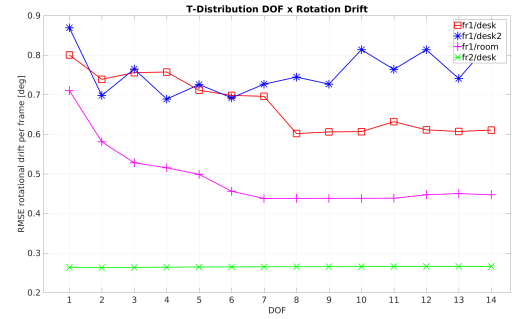


(b) Precision VS Rotational Drift

Figure 6: In (a) there is the behavior of the algorithm with respect to the translational drift over different convergence acceptance ϵ values used in the motion estimation of **frieburg1_desk** and (b) presents the rotational drift.



(a) T-Distribution DOF VS Translational Drift



(b) T-Distribution DOF VS Rotational Drift

Figure 7: In (a) there is the behavior of the algorithm with respect to the translational drift over different sequences and DOF used in the optimization for the motion estimation and (b) presents the rotational drift.

of 30.

5 Conclusion

This work has presented the basis for a parallel version of visual odometry using an RGB-D camera, its implementation details and demonstrates that is possible to configure it to run on real-time with acceptable centimetric or millimetric precision when estimating the camera motion between to image pairs. Real-time operations is possible when analyzing the algorithm for different scenarios with different configuration parameters and choosing those that presents better precision and accuracy but has a good trade-off with cost of computation. Also, an easy to use liner algebra library was created in the process and enables other authors to use GPU power programming in high level thinking, close to conventional calculations narrowing the gap between conventional hand held calculations and programming procedures.

One bottleneck found in this work is the usage of a CUDA Library called Thrust that performs reduction and counting operations in GPU in a synchronous fashion, holding the time of computation. Another problem found is that the algorithm here developed is sensitive to the velocity with which the camera is maneuvered, higher velocities yield lower precision when estimating motion due to blur.

For future works, it is possible to create asynchronous versions of those procedures of Thrust library. Another improvement is to enable the usage of single precision matrices that, in theory, would enable the algorithm to perform at 60 Hz with images at 320×240 resolution and 30Hz for 640×480. Another possibility of improvement is to make the algorithm robust against higher velocities of camera performing, perhaps, some bundle adjustment procedure.

Aknowledgements

Special thanks to Coordination for the Improvement of Higher Level Personnel - CAPES.

References

Baker, S. and Matthews, I. (2004). Lucas-kanade 20 years on: A unifying framework, *International journal of computer vision* **56**(3): 221–255.

Bay, H., Ess, A., Tuytelaars, T. and Van Gool, L. (2008). Speeded-up robust features (surf),

Computer vision and image understanding **110**(3): 346–359.

Blais, G. and Levine, M. D. (1995). Registering multiview range data to create 3d computer objects, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **17**(8): 820–824.

Comport, A. I., Malis, E. and Rives, P. (2010). Real-time quadrifocal visual odometry, *The International Journal of Robotics Research* **29**(2-3): 245–266.

Endres, F., Hess, J., Sturm, J., Cremers, D. and Burgard, W. (2014). 3-d mapping with an rgb-d camera, *IEEE Transactions on Robotics* **30**(1): 177–187.

Engel, J., Sturm, J. and Cremers, D. (2012). Camera-based navigation of a low-cost quadrocopter, *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, IEEE, pp. 2815–2821.

Fraundorfer, F. and Scaramuzza, D. (2012). Visual odometry: Part ii: Matching, robustness, optimization, and applications, *IEEE Robotics & Automation Magazine* **19**(2): 78–90.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A. and Rubin, D. B. (2014). *Bayesian data analysis*, Vol. 2, CRC press Boca Raton, FL.

Gupta, S., Girshick, R., Arbeláez, P. and Malik, J. (2014). Learning rich features from rgb-d images for object detection and segmentation, *European Conference on Computer Vision*, Springer, pp. 345–360.

Gutiérrez, P. D., Lastra, M., Benítez, J. M. and Herrera, F. (2017). Smote-gpu: Big data preprocessing on commodity hardware for imbalanced classification, *Progress in Artificial Intelligence* pp. 1–8.

Henry, P., Krainin, M., Herbst, E., Ren, X. and Fox, D. (2014). Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments, *Experimental robotics*, Springer, pp. 477–491.

Huang, A. S., Bachrach, A., Henry, P., Krainin, M., Maturana, D., Fox, D. and Roy, N. (2017). Visual odometry and mapping for autonomous flight using an rgb-d camera, *Robotics Research*, Springer, pp. 235–252.

Huber, P. J. (2011). Robust statistics, *International Encyclopedia of Statistical Science*, Springer, pp. 1248–1251.

Jafari, O. H., Mitzel, D. and Leibe, B. (2014). Real-time rgb-d based people detection and tracking for mobile robots and head-worn cameras, *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, IEEE, pp. 5636–5643.

Kerl, C. (2012). *Odometry from rgb-d cameras for autonomous quadrocopters*, Master's thesis, Technical University Munich, Germany.

Kerl, C., Sturm, J. and Cremers, D. (2013). Robust odometry estimation for rgb-d cameras, *International Conference on Robotics and Automation (ICRA)*.

Lucas, B. D., Kanade, T. et al. (1981). An iterative image registration technique with an application to stereo vision.

Ma, Y., Soatto, S., Kosecka, J. and Sastry, S. S. (2012). *An invitation to 3-d vision: from images to geometric models*, Vol. 26, Springer Science & Business Media.

Nistér, D., Naroditsky, O. and Bergen, J. (2004). Visual odometry, *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, Vol. 1, Ieee, pp. I–I.

NVIDIA® (2017a). *Jetson TX2 Developer Kit.* **URL:** *https://tinyurl.com/yckouxc5*

NVIDIA® (2017b). Tegra x1. **URL:** *http://www.nvidia.com/object/tegra.html*

NVIDIA® (2018). *CUDA™ Programming Guide.* **URL:** *http://docs.nvidia.com/cuda*

Rusinkiewicz, S. and Levoy, M. (n.d.). Efficient variants of the icp algorithm, *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on.*

Scaramuzza, D. and Fraundorfer, F. (2011). Visual odometry [tutorial], *IEEE robotics & automation magazine* **18**(4): 80–92.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview, *Neural networks* **61**: 85–117.

Steinbrücker, F., Sturm, J. and Cremers, D. (2011). Real-time visual odometry from dense rgb-d images, *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, IEEE, pp. 719–722.

Stückler, J. and Behnke, S. (2012). Model learning and real-time tracking using multi-resolution surfel maps., *AAAI*.

Tykkälä, T., Audras, C. and Comport, A. I. (2011). Direct iterative closest point for real-time visual odometry, *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, IEEE, pp. 2050–2056.

Weiss, S., Achtelik, M. W., Lynen, S., Chli, M. and Siegwart, R. (2012). Real-time onboard visual-inertial state estimation and self-calibration of mavs in unknown environments, *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, IEEE, pp. 957–964.