Resource Efficient Navigation Using Bitstream Computing

Kyle Daruwalla
daruwalla@wisc.edu
University of Wisconsin - Madison
Madison, Wisconsin, USA

Mikko Lipasti
mikko@engr.wisc.edu
University of Wisconsin - Madison
Madison, Wisconsin, USA

Abstract

With the proliferation of self-driving vehicles and autonomous drones, the problem of navigating an environment has become an important research topic. Current solutions, utilizing machine learning, are computationally intensive, and the learned path planning model is poorly understood. In contrast, traditional computer vision techniques provide solutions that are well understood but still computationally intensive.

Given that the brain is efficient at navigation, it is natural to ask the source of this efficiency. Many works have indicated that the data format employed by the brain is critical to its performance. In this work, we apply this insight to a traditional computer vision algorithm for path planning — homography estimation and decomposition. Utilizing prior work in stochastic (bitstream) computing, we design a circuit capable of performing the most computationally intensive component, singular value decomposition, found in the homography algorithm. We show that combining traditional navigation techniques and bitstream computing, we can design a system that is both well understood and efficient.

Keywords homography decomposition, morris water maze, navigation, bitstream computing, stochastic computing

1 Introduction

A typical formulation of the navigation problem in biology is the Morris water maze [1]. As shown in Fig. 1, a rat is dropped in a pool of water, and it must navigate to a landing just below the water's surface to receive rest. Visual cues are placed along the walls of the pool to assists the rat's navigation capabilities.

This is the setting under which we will construct the problem of navigation. Prior work has demonstrated techniques to solve this problem [2], but the solutions rely on performing a singular value decomposition (SVD) [3] [4] [5] which is known to be a computationally intensive task. Previous authors have demonstrated energy efficient stochastic computing implementations of the pseudoinverse [6]. Applying similar techniques to the SVD, we demonstrate that bitstream computing can provide a low resource implementation of traditional homography-based navigation.

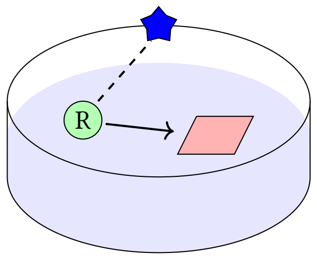


Figure 1. A diagram of the Morris water maze. A rat (green circle) navigates to a landing (red) via visual cues from a target on the pool wall (blue star).

2 Algorithm Design

Homography estimation and decomposition addresses the problem of navigation using a geometric formulation. In this section, we will introduce that formulation, and the algorithm to solve the navigation task.

2.1 Problem Setup

From the visual cue in Fig. 1, we extract feature points (e.g. the points of the star). Let F_1 denote the frame of reference (i.e. position and orientation) of the drone at the current time step, and F_2 denote the frame of reference at the target location (the landing platform in Fig. 1). As shown in Fig. 2, a feature point, p, is projected onto two different perspective planes with respect to each frame. The homography matrix, $H \in \mathbb{R}^{3\times3}$, is a transformation matrix that maps each projected feature point to its corresponding pair:

$$p_2 \sim Hp_1 \quad \text{or} \quad p_2 \times Hp_1 = \vec{0} \tag{1}$$

where "$\sim$" indicates equality up to scale.

This relation allows to create the following system of equations:

$$A^j h = \begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_2x_1 & x_2y_1 & x_2 \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & y_2x_1 & y_2y_1 & y_2 \end{bmatrix} h = \vec{0} \tag{2}$$

where h is a vector of the elements of H, $p_1 = [x_1\ y_1\ 1]^\top$, and $p_2 = [x_2\ y_2\ 1]^\top$. Then for each pair of points (p_1^j, p_2^j), we can construct Eq. 2 and "stack" the equations to get an overall

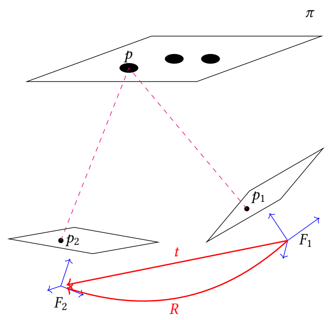


Figure 2. A geometric formulation for navigation. We are trying to determine the rotation, R, and translation, t, that transforms our perspective (p_1 and p_2) of a visual cue, p, from the current position, F_1, to the final position, F_2.

system:

$$Ah = \begin{bmatrix} A^1 \\ A^2 \\ A^3 \\ A^4 \end{bmatrix} h = \vec{0} \tag{3}$$

where $A \in \mathbb{R}^{8\times 9}$ and $h \in \mathbb{R}^{9\times 1}$. This method for solving for h is known as the eight-point algorithm, and it is sufficient to estimate the homography matrix, H [5]. Eq. 3 has infinitely many solutions. Thus, we typically also enforce the condition that $\|h\|_2 = 1$. With this additional constraint, the solution is given by the right most singular vector of A [7] [5] [3].

2.2 Recovering Rotation and Translation

Once H is estimated using pairs of feature points, it must be decomposed into its rotational and translational components in Eq. 4 (n is the vector normal to the image projection plane).

$$H = R + tn^\top \tag{4}$$

Several numerical methods exist to find this decomposition, but we use the common Zhang SVD-based decomposition, which recovers R, t, and n from the SVD of $H^\top H$ [2]. For the sake of brevity, we will not discuss the derivation of this method, and we refer the reader to [2].

The overall algorithm for using the homography for navigation is given in Alg. 1. It performs some normalization as well to stabilize the numerical solutions.

Algorithm 1 Homography Estimation and Decomposition Overview (citations per step)

```
Require: Extracted pairs of feature points
1: Calculate normalization matrix, $T$ [5]
2: Normalize feature points: $x' = Tx$
3: Find homography by solving linear system of equations [7] [8]
4: Reverse normalization: $H = T_B^{-1} H' T_A$
5: Decompose homography by taking SVD of $H^\top H$ [2] [3]
```

2.3 Iterative SVD

Many of the computations in Alg. 1 have been implemented with stochastic computing by prior work [6] [9]. Instead, we focus on an efficient implementation of the singular value decomposition (SVD). The SVD of a $m\times n$ matrix, A, is defined as

$$A = U\Sigma V^\top \tag{5}$$

where $U \in \mathbb{R}^{m\times r}$, $V \in \mathbb{R}^{n\times r}$, and $\Sigma \in \mathbb{R}^{r\times r}$. Here $r = \operatorname{rank}(A)$. U and V's columns are orthogonal unit vectors, and Σ is a diagonal matrix. The columns of U and V are referred to as the left and right singular vectors, respectively, while the elements of the diagonal of Σ are the singular values.

H is a 3×3 matrix, so we can quickly find its SVD by the power iteration method (referred to as the iterative SVD) [10]. Alg. 2 provides an overview of this method. We note that [9] has provided an SC implementation of the eigenvalue decomposition which is closely related to the SVD. Our method does not require the stretching techniques proposed in the previous work. Fig. 3 illustrates a block diagram of our stochastic computing circuit. [6] and [11] provide the details on implementations of the blocks in the diagram, which we omit for the sake of brevity.

Algorithm 2 Iterative SVD

```
Require: Input matrix $A \in \mathbb{R}^{m\times n}$ and initial guess $v_0 \in \mathbb{R}^n$
1: for $k = 1, 2, \ldots$ (until convergence) do
2:     $w_k = Av_{k-1}$
3:     $\alpha_k = \sqrt{w_k^\top w_k}$
4:     $u_k = w_k / \alpha_k$
5:     $z_k = A^\top u_k$
6:     $\sigma_k = \sqrt{z_k^\top z_k}$
7:     $v_k = z_k / \sigma_k$
8: end for
9: return First left/right singular vectors, $u_k$ & $v_k$, and first singular value, $\sigma_k$
```

Alg. 2 only computes the first left and right singular vectors and first singular value. In order to compute the rest of the SVD, we remove the first component from by

$$A' = A - \sigma_1 u_1 v_1^\top \tag{6}$$

then apply Alg. 2 to A'. This process is repeated as many times as the rank(A).

2.4 Navigation Algorithm

Fig. 4 illustrates a full overview of the navigation pipeline. Initially, the drone moves around randomly until the target is found (much like how the rat swims around the pool until it discovers the landing). Upon acquiring the target, the drone stores its current perspective of the visual cues. Subsequently, at each time step, the drone uses Alg. 1 to

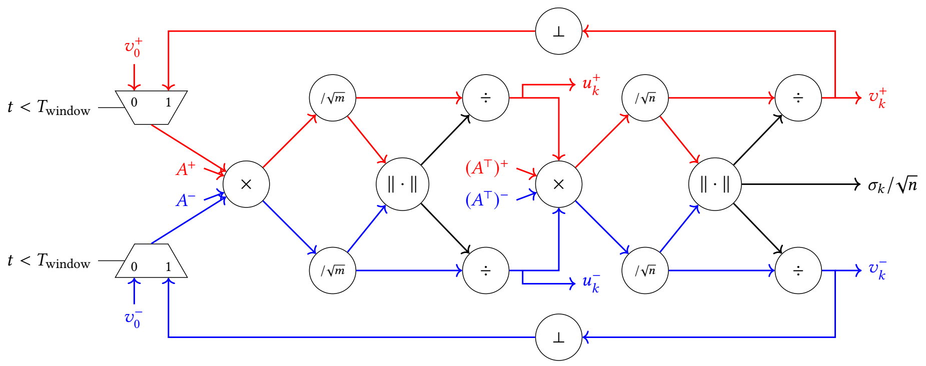


Figure 3. A block diagram of the bitstream computing iterative SVD. T_{window} is the window length over which we estimate a stochastic bitstream. The $/x$ blocks denote a fixed gain division by x, the $\perp$ blocks denote a decorrelator, and the $\|\cdot\|$ blocks denote the L2-norm. [6] and [11] provide details on the implementations of these blocks. Red denotes positive channels and blue denotes negative channels for signed compute as detailed in [6].

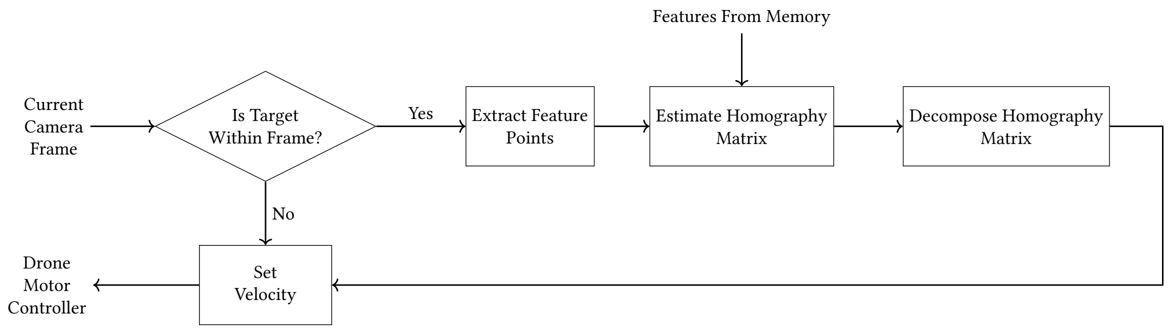


Figure 4. A block diagram of the homography-based navigation system.

determine the direction to travel to efficiently navigate to the target.

3 Simulation Results

We simulated the system described by Fig. 4 in Matlab. Fig. 5 illustrates the result of these simulations. A drone is randomly placed in a circular enclosing space with a visual landmark placed on the wall (an asterisk-like shape). It moves around randomly until it finds a designated landing zone. It then stores its current view of the visual landmark. On the next trial, the drone extracts feature points from the visual landmark using corner detection. It then tries to find the homography matrix between its current set of feature points, and those from when it was last on the landing zone. It decomposes this homography matrix to make a decision about which direction to move. This process is repeatedly iteratively until the drone reaches the landing zone. Fig. 5a shows the path the drone took on the second trial to reach the landing zone. As is evident in the plot, the chosen path is fairly straight. Fig. 5b shows the feature points as seen by the drone at each time step. Notice that they converge quickly to the feature points as seen from the landing zone.

4 Hardware Results

We evaluate the proposed using Verilog implementations mapped to ultra-low power Lattice FPGAs. Floating point and fixed point baselines are created using Vivado HLS. Below, we will discuss the area and dynamic power results.

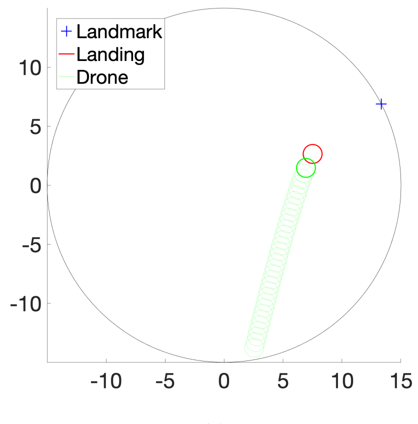
4.1 Area Results

We calculate the area as # LUTs + # FFs and compare the normalized area w.r.t. to the floating point design in Fig. 6. As expected, the stochastic computing design consumes significantly less area than the floating point or fixed point alternatives.

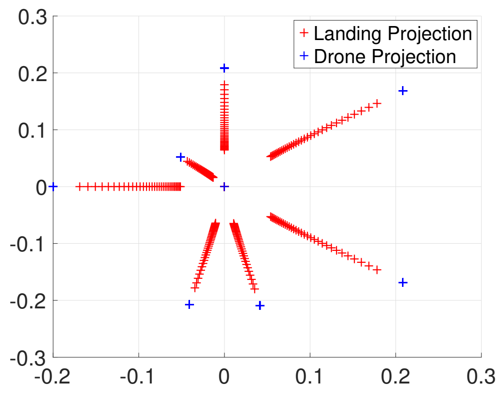
The fixed point design consumes more area that the floating point design since Alg. 2 involves division and square-root operators. Xilinx has optimized floating point IPs for these units, but the FXP design is a multi-cycle pure-Verilog implementation.

4.2 Power Results

Fig. 7 shows the total power consumption for each design variation. Notice that the SC design is an order of magnitude lower than the FP and FXP designs. This is a direct result



(a)



(b)

Figure 5. (a) A drone traveling to a landing using homography-based navigation. The faded green trace illustrates the position of the drone at each time step of the trial. (b) The feature points on the projected image plane as seen by the drone at each time step. Feature points at target location shown in blue. Notice that the projection of the feature points becomes closer to the desired projection by following the homography.

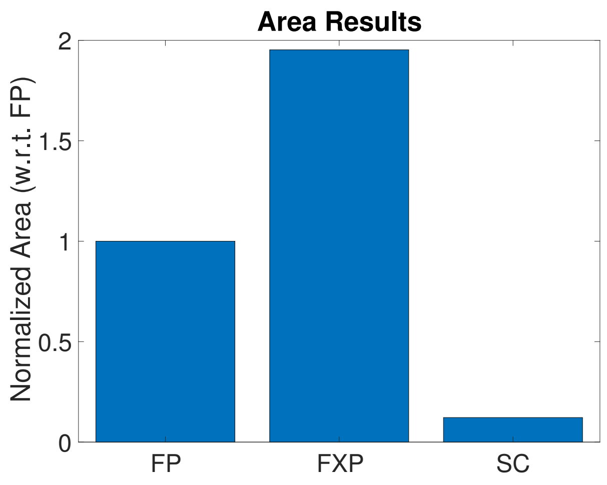


Figure 6. Area results for floating point (FP), fixed point (FXP), and stochastic computing (SC) designs. The SC consumes significantly fewer resources. (Area = # LUTs + # FFs)

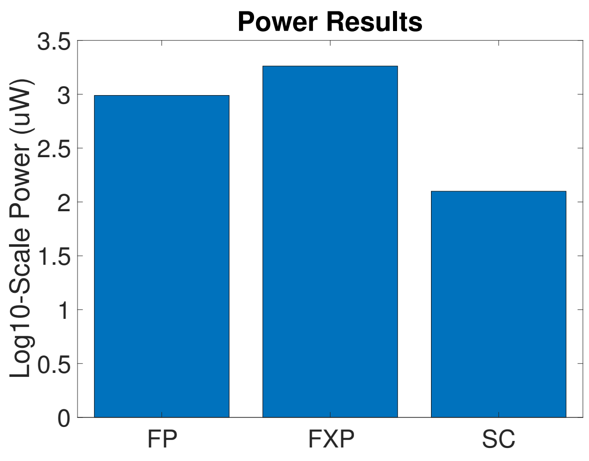


Figure 7. Total power results for floating point (FP), fixed point (FXP), and stochastic computing (SC) designs when mapped to a Lattice LM4K FPGA (note y-axis is log-scale). The SC design is an order of magnitude lower power than the traditional designs.

of the low resource consumption in Fig. 6. In fact, FP/FXP designs need to be partitioned across multiple LM4K FPGAs.

5 Conclusion

Path planning or navigation is becoming an important research area. Current computing and learning techniques fail to address this topic either in term of efficiency or robustness. The applications require both design metrics to be met, and we show that stochastic computing in combination with computer vision can achieve these goals. Our designs are both low-power and well-understood. As a result, the system is feasible for deployment (as we have demonstrated in simulation). We hope to extend this work to other PAV applications beyond navigation.

References

[1] R. G. M. Morris, P. Garrud, J. N. P. Rawlins, and J. O'Keefe. Place navigation impaired in rats with hippocampal lesions. *Nature*, 297(5868):681–683, jun 1982.

[2] Ezio Malis and Manuel Vargas. Deeper understanding of the homography decomposition for vision-based control. *Sophia*, 6303(6303):90, 2007.

[3] Olivier D. Faugeras and Francis Lustman. Motion and Structure From Motion in a Piecewise Planar Environment. *International Journal of Pattern Recognition and Artificial Intelligence*, 02(03):485–508, 1988.

[4] Zhenhua Yang, Lixin Tang, and Lingsong He. A New Analytical Method for Relative Camera Pose Estimation Using Unknown Coplanar Points. *Journal of Mathematical Imaging and Vision*, 60(1):33–49, 2018.

[5] Richard I. Hartley. In defense of the eight-point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):580–593, jun 1997.

[6] Rohit Shukla, Erik Jorgensen, and Mikko Lipasti. Evaluating hopfield-network-based linear solvers for hardware constrained neural substrates. *Proceedings of the International Joint Conference on Neural Networks*, 2017-May:3938–3945, 2017.

[7] Robert Collins. Lecture 16: Planar Homographies.

[8] Elan Dubrofsky. *Homography Estimation*. PhD thesis, The University of British Columbia, 2009.

[9] Pai Shun Ting and John Patrick Hayes. Stochastic logic realization of matrix operations. *Proceedings - 2014 17th Euromicro Conference on Digital System Design, DSD 2014*, pages 356–364, 2014.

[10] A. H. Bentbib and A. Kanber. Block power method for SVD decomposition. *Analele Stiintifice ale Universitatii Ovidius Constanta, Seria Matematica*, 23(2):45–58, 2015.

[11] IBM Neurosynaptic System Neuron Function Library Reference Manual. Technical report, IBM Corporation, 2016.