Image-Based Visual Servoing for Differentially Flat Underactuated Systems using Cross-Entropy Motion Planning

Matthew Sheckells and Marin Kobilarov[1]

***Abstract*— This work considers an Image-Based Visual Servoing (IBVS) technique for differentially flat underactuated systems. We formulate the problem as an optimal control problem that can be solved using cross-entropy motion planning. As a cost function, we use the average squared distance between SURF features matched between the desired image and the image projected from the final state of the trajectory. The optimization is performed over a polynomial parametrization of the flat outputs of the system to decrease the dimensionality of the optimization. Preliminary experimental results show that the technique has a wide radius of convergence for the case considered, which is a simulated quadrotor with a downward-facing camera. This method assumes a model of the environment is known or can be estimated (e.g. using structure from motion algorithms).**

I. INTRODUCTION

Visual servoing concerns bringing computer vision data into the control loop of a robot and has been thoroughly studied over the past two decades [2][9][10]. Image-Based Visual Servoing (IBVS) attempts to directly control image features in order to align them with some objective set of features and thereby bring the system into a desired state. Several works look at applying IBVS to underactuated dynamic models, like quadrotors [7][4][3]. The standard approach is based on a closed form control strategy using the image jacobian, whereas this work formulates the problem as an optimal control problem. By specifying the cost function as the average squared distance between features in 2D image coordinates, we can apply a cross-entropy motion planning algorithm [5] to find an optimal trajectory which brings the system into alignment with the desired image. We use a sampling-based method because local methods like Gauss-Newton do not converge. The optimization is performed over a polynomial parametrization of the flat outputs of the system to increase efficiency and radius of convergence. The cost function relies on the ability to generate images from any state in the workspace, so a 3D model of the environment is constructed beforehand.

II. METHODS

A. Cross-entropy Method

The Cross-Entropy (CE) method is a sampling based approach to multi-extremal optimization and importance sampling. It has been employed in motion planning for nonlinear robotic systems operating in constrained environments [5]. At a high level, it consists of two phases. First, a set of trajectories is generated by sampling a finite dimensional parametrization from a Gaussian Mixture Model (GMM). Next, the parameters of the GMM are fit to the set of samples corresponding to the those with the smallest costs. This process is repeated until convergence.

B. Cost Function

Based on the state of the system, a virtual image is generated from a 3D model of the environment. SURF features [1] are extracted from the virtual image using an OpenCV implementation. The set of features is denoted $F = \{f_1, f_2, ..., f_N\}$. The features are also extracted from the goal image (but only once), and the set is denoted $F_g = \{f_{g_1}, f_{g_2}, ..., f_{g_M}\}$. Features from both images are matched using a k-nearest neighbors matching algorithm. Matches are then filtered using a ratio test as proposed by Lowe [6], resulting in K "good" matches. This gives a mapping $\pi : F \to F_g$ between goal features and current state features. The cost is then calculated as

$$J_{Image} = \frac{\sum_{i=1}^{K} \|f_i - \pi(f_i)\|^2}{K},$$

where the difference between two features is defined as the difference between their 2D pixel coordinates and the magnitude is a Euclidean distance. Hence, the cost function is the average squared 2D distance between matching features.

This function by itself has the drawback that the cost drops to zero when there are no matches. So, when a state yields fewer than K_{min} features, the cost is calculated as

$$J_{Image} = \frac{a}{K_{min} + b}.$$

where K_{min}, a, and b are chosen experimentally. Costs associated with the state or control effort J_{State} can be added directly to this cost function if additional state information is available. Thus, the total cost is $J = J_{Image} + J_{State}$.

C. Differential Flatness and Trajectory Parametrization

This work considers only differentially flat systems. A system with state $x \in \mathbb{R}^n$ and inputs $u \in \mathbb{R}^m$ is differentially flat if one can find outputs $y \in \mathbb{R}^m$ of the form $y = h(x, u, \dot{u}, \ldots, u^{(a)})$ and functions ψ and α such that $x = \psi(y, \dot{y}, \ldots, y^{(b)})$ and $u = \alpha(y, \dot{y}, \ldots, y^{(c)})$.

To reduce the number of optimization parameters, the trajectory is parameterized in time as a Bézier curve over the flat outputs $y(t)$ of the chosen system. The curve should be b-times differentiable to ensure continuity in the state trajectory $x(t)$. The full state of the systems and controls necessary to produce the trajectory can be reconstructed using α and ψ.

[1]Matthew Sheckells and Marin Kobilarov are with the Department of Computer Science and the Department of Mechanical Engineering, Johns Hopkins University, 3400 N Charles Str, Baltimore, MD 21218, USA `msheckells|marin@jhu.edu`

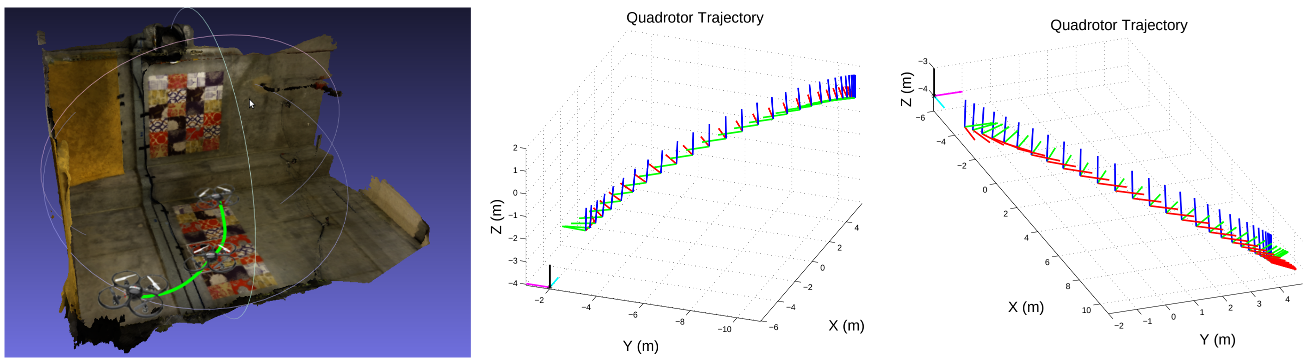

Fig. 1. The image on the left illustrates the environment with an example trajectory. On the right, two computed trajectories are shown that navigate to the same desired image from different initial conditions. Both have a final desired state indicated by an image rendered from the position $(-6, -2, -4)$ with 0 yaw. This position is marked by the cyan-magenta-black axes on the plots. The first trajectory starts at $(5, -11, 1)$ with a yaw of $\pi/3$. The second trajectory starts at $(10, 4, -4)$ with a yaw of $\pi/3$.

D. Feasible Initialization

The trajectories were initialized by first solving the same problem for a fully-actuated 3D rigid body. The relevant flat outputs from that solution were then used to initialize the flat output trajectory for the quadrotor system. This gives an initial guess for the solution that is already close to the final result.

III. Experiment

A. System

The system considered is a simplified quadrotor model with a downward facing camera. It takes four controls as input: three torques that correspond to the roll, pitch, and yaw axes and a thrust force along the z-axis of the quadrotor. It is also subject to gravity. This system is known to be differentially flat [8]. The full state is that of a rigid body with second order dynamics given as $x = (p, R, \dot{p}, \omega) \in \mathrm{SE}(3) \times \mathbb{R}^6$. The flat outputs are given as $y = (p_1, p_2, p_3, \gamma) \in \mathbb{R}^4$, where γ is the yaw of a roll-pitch-yaw parametrization of the rotation matrix R. For this work, we only need to recover the pose and linear velocities of the system since the controls and angular velocities are not used in the image-based cost function. So, we have $p = y_t$, $\dot{p} = \dot{y}_t$ and the three columns of the rotation matrix R_x, R_y, R_z are reconstructed as

$$\begin{aligned} R_z &= m(\ddot{y}_t - g)/\|m(\ddot{y}_t - g)\| \\ R_y &= R_z \times \begin{pmatrix} \cos y_4 \\ \sin y_4 \\ 0 \end{pmatrix} \Bigg/ \left\| R_z \times \begin{pmatrix} \cos y_4 \\ \sin y_4 \\ 0 \end{pmatrix} \right\| \\ R_x &= R_y \times R_z \end{aligned}$$

where m is the mass of the quadrotor, g is the gravity vector, and $y_t = (y_1, y_2, y_3)$.

B. Setup

All experiments were implemented in C++, using OpenCV for feature detection and OGRE for model rendering [12]. An implementation of the CE method was used from the Geometric Control, Optimization, and Planning library [11].

The workspace environment considered was a small lab testing area with textured walls suitable for feature matching. A 3D model of the area was created using the Agisoft Photoscan SFM software [13].

The initial attitude of the system was always specified with zero roll and pitch. In addition to the feature-based cost function, penalties were added to prevent a non-zero final velocity or non-level attitude. For the cost function, we used $K_{min} = 7$, $a = 10^5$, and $b = 0.01$, as these values worked well in practice.

The Bézier curve parametrization used seven control points to ensure smoothness and to provide enough degrees of freedom for shaping the trajectory. The first three points were fixed during optimization so as to keep the first two derivatives of the flat outputs constant. This ensured that the optimized trajectory matched the given initial conditions of the system.

C. Cross-entropy Results

Two trajectories generated by the cross-entropy method are shown in Figure 1. Each trajectory was optimized over 50 iterations while generating 200 sample trajectories per iteration. Both trajectories consider a desired image taken from the position $(-6, -2, -4)$ with 0 yaw. The first trajectory starts at $(5, -11, 1)$ with a yaw of $\pi/3$. The second trajectory starts at $(10, 4, -4)$ with a yaw of $\pi/3$. It can be seen that both trajectories effectively converge to the goal position, with the first ending at $(-5.07, -3.19, -1.70)$ and the second ending at $(-4.24, -1.56, -4.42)$.

IV. Conclusion

This work considers an IBVS technique that is suitably general for any differentially flat system. The method formulates IBVS in an optimal control context and uses a cross-entropy motion planner to minimize an image-based cost function. The trajectories are parametrized in flat output space using Bézier curves to reduce the dimensionality of the optimization problem. It has been shown experimentally that this method shows promising convergence properties when applied to a simulated quadrotor system. Future work will give a more statistical treatment to the convergence of the method and consider performance in the presence of image noise and obstacles.

References

[1] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. 2008. Speeded-Up Robust Features (SURF). Comput. Vis. Image Underst. 110, 3 (June 2008), 346-359.

[2] B. Espiau, F. Chaumette, and P. Rives. A new approach to visual servoing in robotics. IEEE Transactions on Robotics and Automation, Vol.8(3), pages: 313-326, 1992.

[3] N. Guenard, T. Hamel, and R. Mahony, "A Practical Visual Servo Control for an Unmanned Aerial Vehicle," Robotics, IEEE Transactions on, vol.24, no.2, pp.331,340, April 2008

[4] T. Hamel and R. Mahony, Visual servoing of an under-actuated dynamic rigid-body system: An image based approach. IEEE Transactions on Robotics and Automation, 2002, Vol. 18(2), pages: 187-198.

[5] M. Kobilarov, Cross-entropy Randomized Motion Planning, In Robotics: Science and Systems (R:SS), 2011

[6] David G. Lowe, "Distinctive image features from scale-invariant keypoints," International Journal of Computer Vision, 60, 2 (2004), pp. 91-110.

[7] L. Mejias, S. Saripalli, G.S. Sukhatme, and P. Cervera, Visual servoing for tracking features in urban areas using an autonomous helicopter, In Journal of Field Robotics, 2006. Vol 23. Issue 3-4, pages: 185-199.

[8] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," Robotics and Automation (ICRA), 2011 IEEE International Conference on , vol., no., pp.2520,2525, 9-13 May 2011

[9] R. Pissard-Gibollet and P. Rives. Applying visual servoing techniques to control of a mobile hand-eye system. In Proceedings of the IEEE International Conference on Robotics and Automation, ICRA95, pages: 166-171, Nagasaki, JAPAN, 1995.

[10] Lee E. Weiss, Arthur C. Sanderson, and Charles P. Neuman. Dynamic sensor-based control of robots with visual feedback. IEEE Transactions on Robotics and Automation, 3(5):404417, October 1987

[11] Geometric Control, Optimization, and Planning Library. `https://ascol.lcsr.jhu.edu/Software`

[12] OGRE - Open Source 3D Graphics Engine. `http://www.ogre3d.org/`

[13] Agisoft Photoscan. `http://www.agisoft.com/`