NTNU – Trondheim
Norwegian University of
Science and Technology

Detecting Vacuum Cleaner Robots from a Quadrotor

Simen Haugo

Rapport i:	TTK4550 - Teknisk kybernetikk, fordypningsprosjekt
Veileder har vært:	Edmund Førland Brekke, ITK
Rapporten ble levert:	Desember 2016

Abstract

Detecting and tracking moving objects is a key ingredient in building autonomous aerial vehicles that can perform interactive tasks in a dynamic environment. This project report studies this problem in the context of Mission 7 in the International Aerial Robotics Competition, in which a small aerial vehicle must navigate around indoor environments, without external motion cues, and physically interact with moving ground targets. We provide a survey of common and related detection methods, and evaluate the theory in practice on challenging real-life data sets by implementing a cascaded detector based on color segmentation and model-based optimization.

Preface

The work described in this project report has been carried out in association with Ascend NTNU, a student organization with the annual goal of participating in the International Aerial Robotics Competition, the longest running university-based competition for autonomous aerial vehicles. In this competition, students participate to push the boundaries of technology in missions that require robotic behaviour never before demonstrated by a flying vehicle. Ascend NTNU first participated in August 2016, claiming both Best Performance and Best Team T-Shirt Design awards at the American venue, and is currently aiming to defeat the current mission in 2017.

The very first mission was held when fossile-fueled helicopters were the platform of choice, and challenged its participants to pick up a disk and place it at a designated area — a feat regarded as almost impossible for a fully autonomous flying robot at the time. Over three years, the teams improved their entries until one team, at last, was able to move a single disk. Twenty-five years and six completed missions later, the seventh and current mission involves autonomously navigating an indoor arena while avoiding moving obstacles and herding ten robotic vacuum cleaners to a designated area, without any external sensors like motion tracking systems or GPS.

While a flaky WiFi connection prevented us from demonstrating our full capabilities at the actual venue, our full-scale tests at home a week earlier are quite memorable. Our quadrotor had an impressive hardware design and could autonomously fly in any desired path in the arena while avoiding obstacles and detecting the targets. Despite demonstrating many of the required behaviours, there are still problems that must be solved before the remaining behaviour can be integrated. Robust localization at low heights, fully autonomous take-off and landing, obstacle detection at a wider field of view, and more accurate target detection, to name the some.

This project report is my contribution to the target detection. My goal throughout this project has been to make a detector that can run completely on-board and reliably detect targets without false positives. This could be combined with a tracking algorithm to estimate the heading and motion of the targets, that is necessary for strategic planning and controlling the quadrotor. Although there is much work left, I hope that the literature study provides useful reading of the many possibilities that exist and that my work so far can inspire better solutions in the future.

Trondheim, December 2016
Simen Haugo

Thanks to my supervisor Edmund Førland Brekke and to Annette Stahl for reviewing earlier drafts of this report and making valuable suggestions.

Contents

List of Tables

List of Figures

1 Introduction

In which we define the aim of this report.

1.1 Motivation

Micro aerial vehicles (MAV) offer high mobility for a low cost, and provide an ideal robotic platform for a wide range of applications, including aerial photography, autonomous inspection of hazardous areas, and search and rescue. Despite this, there are still many challenges involved in building aerial vehicles that can autonomously perform meaningful tasks in man-made environments. In particular, detecting and tracking moving objects is necessary to react to changes in the environment and to interact with other objects. While this is already a fairly involved problem, and also a hot area of research within computer vision, further difficulties arise when the platform itself is moving in indoor environments without external motion cues.

1.2 Problem specification

This project report studies the problem of detecting moving objects from a MAV in the context of Mission 7 of the International Aerial Robotics Competition (IARC), where MAVs are expected to autonomously interact with moving targets and herd them across a goal line, as shown in Figure 1. The aim of this report is to provide a literature survey of related work and a description and evaluation of our own solution to the detection problem. The work is done in association with Ascend NTNU, a student organization currently comprised of 28 members with the annual goal of competing in IARC.

The hardware platform is a quadrotor built by Ascend NTNU and equipped with a downward facing fisheye camera, whose view is shown in Figure 2, that is capable of providing high framerate video. Additional sensors include a gyroscope, accelerometer and a laser rangefinder, used by an on-board flight controller to estimate the quadrotor's orientation and height above ground.

We evaluate our solution on recorded data from the competition held in August 2016 and from our own tests, where we manually piloted our quadrotor through various maneuvers and interacted with the targets. These data sets present difficult challenges for detection and tracking, including noise in the estimated camera orientation and position, multiple targets, red herrings, targets entering and leaving the field of view, missing video frames, and large viewpoint changes over short periods of time.

Figure 1: Mission 7 consists of modified *iRoomba* robots of two kinds, tall *obstacles* and short *targets*. In total, there are four obstacles and ten targets. The targets have a front bumper and a top plate (colored red or green) connected to a sensor. When either is activated, the robot will turn a programmed angle and continue in a straight line. Participants must exploit this behaviour to guide the targets across a designated edge of the arena.

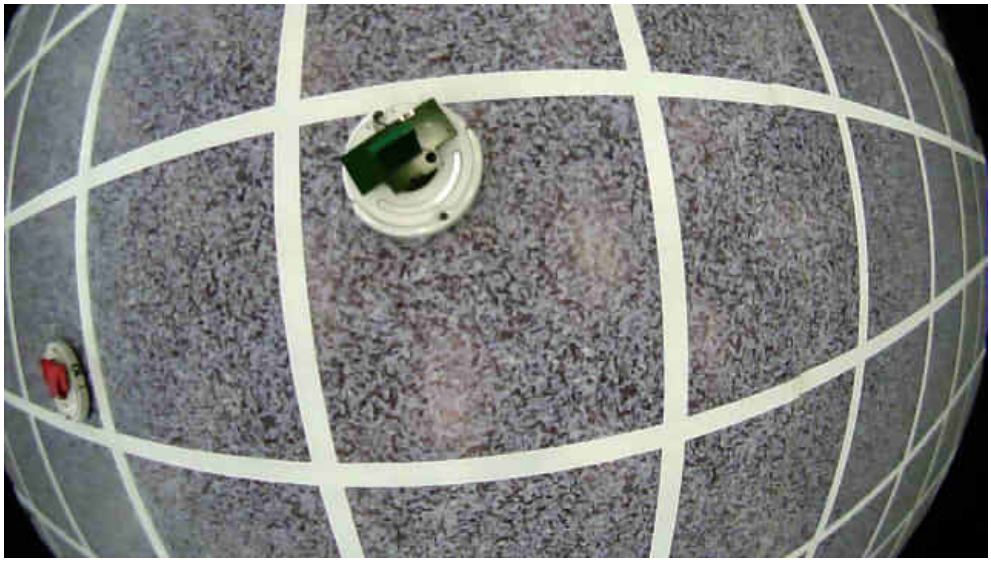


Figure 2: Our quadrotor has a downward facing fisheye camera, with a field of view of more than 180 degrees, capable of capturing video at 60 frames per second at 1280×720, or 120 frames per second at 640×480. We want to use this camera to detect the targets, both red and green, allowing the quadrotor to plan a herding strategy based on their motion, and interact with their touch sensors through visual servoing.

1.3 Report outline

Chapter 2 gives an overview of common camera-based object detection methods, as well as prior attempts at solving the particular IARC problem. Chapter 3 presents our attempt based on research presented in Chapter 2. Chapter 4 evaluates our solution and Chapter 5 summarizes the report and suggests future work.

1.4 Notation

Our main notation is summarized in Table 1.

Vectors written inline with the comma notation expand to column vectors

$$p = (x, y, z) := \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Components of a vector $x \in \mathbb{R}^n$ are accessed by subscript 1 through n

$$x = (x_1, x_2, ..., x_n)$$

If $n \leq 3$ we use subscript x, y, z

$$\boldsymbol{\omega} = (\boldsymbol{\omega}_x, \boldsymbol{\omega}_y, \boldsymbol{\omega}_z)$$

We use the numerator layout when differentiating quantities by vectors, that is, for

$$\begin{aligned} p &\in \mathbb{R}^n \\ x &= x(p) \in \mathbb{R}^m \\ s &= s(p) \in \mathbb{R} \end{aligned}$$

we write

$$\partial s/\partial p = \begin{bmatrix} \partial s/\partial p_1 & \partial s/\partial p_2 & \dots & \partial s/\partial p_n \end{bmatrix}$$

and

$$\partial x/\partial p = \begin{bmatrix} \partial x_1/\partial p \\ \partial x_2/\partial p \\ \vdots \\ \partial x_m/\partial p \end{bmatrix} = \begin{bmatrix} \partial x_1/\partial p_1 & \partial x_1/\partial p_2 & \dots & \partial x_1/\partial p_n \\ \partial x_2/\partial p_1 & \partial x_2/\partial p_2 & \dots & \partial x_2/\partial p_n \\ & & \vdots & \\ \partial x_m/\partial p_1 & \partial x_m/\partial p_2 & \dots & \partial x_m/\partial p_n \end{bmatrix}$$

Symbol	*Meaning*
$\mathcal{M}$	Model image
$\mathcal{I}$	Input image
$\nabla\mathcal{I}$	Input image gradient
$\mathcal{W}$	Geometric warp function
$\phi_{\mathcal{I}}$	Input image descriptor
$\phi_{\mathcal{M}}$	Model image descriptor
$t = (t_x, t_y)$	2D pixel coordinate in model image
$s = (u, v)$	2D pixel coordinate in input image
δs	2D pixel displacement in input image
E	Objective function
R	Regularizer
g	Gradient
r	Residual
θ	Parameters
$\hat{\theta}$	Estimated parameters
$\delta\theta$	Small parameter update
$\delta\hat{\theta}$	Estimated parameter update
p^o	3D point in object coordinates
p^c	3D point in camera coordinates
R_o^c	Rigid-body rotation
T_o^c	Rigid-body translation
H_o^c	Rigid-body transformation
π	Camera projection
$\xi = (\boldsymbol{\omega}, \boldsymbol{v})$	Twist
J	Interaction matrix

Table 1: Main notation

2 Related work

In which we define the target detection problem and conduct a literature survey.

2.1 Problem definition

We define the problem of interest as *detecting* and *tracking* moving objects from a moving platform in partially unknown indoor environments using a single RGB camera, where the objects of interest are the iRoomba *target* robots[1]. By *detecting* we mean determining the 3D rotation and translation, henceforth referred to as the *pose*, of each target present in a single video frame. By *tracking* we mean maintaining the identities of each target while they are visible in the video and determining time-dependent properties like their velocity and mode of motion. Solving this problem will be useful for planning strategic paths and actions, and interacting with the targets through visual servoing. Although our focus in this project report is limited to detection, we have conducted the survey with the mindset that the detection algorithm *is to be* combined with a tracking algorithm, hence it is part of the problem of interest.

There are several constraints that should be noted. The arena consists of a flat floor, where anything with lesser degrees of flatness than that of the floor is likely an object of interest. There is an imperfect grid pattern on the floor, which can — and has been — made of pretty much anything, ranging from loose strips of fabric that flutter about from the wind produced by the quadrotor, to white tape that reflect the twinkling ceiling lights. We have limited information about the pose of our camera, restricted to noisy measurements of its height and orientation, and linear acceleration and angular velocity. Although we are given information about the apperance of the iRoomba targets, their appearance is susceptible to change on a short notice[2].

From the above, we can extract some key aspects of our problem, that may limit the applicability of certain existing solutions:

- A moving camera means that we cannot easily seperate static geometry from moving objects in the image. We must also cope with vastly different viewpoints, such as looking straight down at the target from one meter altitude, or looking at it sideways while near the ground.
- The indoor environment means that approaches assuming the camera is at high altitudes, or equivalently, that the objects are far away, would operate outside their

[1] If we are able to apply our solution to the obstacle robots, then that is an unintended bonus effect.

[2] The top sensor on the robot has already undergone one such modification, which was an unwelcome surprise to one of the participants who had calibrated their algorithms to the early design.

intended domain. Moreover, the lighting conditions are beyond our control, and we should expect to deal with phenomenon like specular highlights.

- The platform is a drone with fast dynamics and the solution must be able to keep up with its motion. The unreliability of the WiFi environment, of which we have direct experience, means that offloading work is infeasible, and that the solution should be capable of running on-board. These aspects put constraints on the computational cost of the solution.
- The unknown pattern of the floor means that we should be prepared for a variety of textures and colors in the image. For example, the pattern may consist of high frequency noise, which can throw off an edge detector looking for sharp transitions — it may contain shapes that resemble circles, perhaps even similiar in scale to the robots — or it could contain colors that are very similiar to the top plates, making it difficult to distinguish objects solely on color.

We can also make some handy simplifications. For example, since the floor is flat, if we know the height and orientation of the camera, we can compute the 3D location of any pixel in the image belonging to the floor itself, or any object at a known height above the floor — information that would otherwise require 3D sensing. Both the constraints we listed above, and the allowable simplifications that follow from them, have guided us in our search for prior work that we describe next.

2.2 Overview of detection methods

This section provides an overview of literature on detection methods. Although some tracking methods are involved, they have not been the focus of the study.

2.2.1 Common terminology

The following is a rough categorization of the different guises the detection problem has gone under in the literature we have found:

- Recognition or Classification: These terms are often used in machine-learning literature, where a database of labelled examples of objects is available, and one wants to determine which of these objects are present in a given image, perhaps localizing them with rough bounding boxes.
- Detection or Localization: These terms often indicate that the author is more concerned with the accurate position the object, for example, its position and rotation in 3D space, the exact region of pixels that make up its silhouette, or a bounding box in the image.
- Segmentation: This term refers to dividing an image into regions of different categories. For example, assigning a label to each pixel that describes what type of object that pixel belongs to. The simplest case is that of binary segmentation, where a region is either background (not the object) or foreground (the object).

To paraphrase Szeliski [47]: If we know what we are looking for, the problem is one of detection. If we have a list of similiar looking objects, we are trying to recognize an instance. If we have no idea what we might be looking at, the problem is one of category recognition. Our problem is most related to *detection* and *localization* literature, because we are interested in the target's position and orientation in 3D space.

Another closely related topic to object detection is *camera localization*. On one hand, we want to estimate the pose of an object relative to the camera, on the other, we want to estimate the pose of the camera relative to a scene. These problems converge to the exact same problem when the scene itself is the object of interest, and are otherwise linked by sharing many underlying techniques. For example, *Simultaneous Localization and Mapping*, or SLAM, estimates the pose of a viewer in an unknown environment (localization) and simultaneously builds a 3D model of the environment (mapping). SLAM could be used to build a 3D model of the object of interest as well as track its pose relative to the camera. For example, Crivellaro et al. [10] compare their object detection algorithm against LSD-SLAM [17], Prisacariu et al. [39] compare theirs against PTAM [24]. Meka et al. [31] track rigid objects with PTAM, and use their (otherwise completely unrelated) algorithm to retexture surfaces.

During our research we developed a rough categorization of detection approaches:

- Model-based — Model-free
- 3D — 2D
- Rigid — Deformable

Model-based methods assume that the appearance of the object is known, usually in the form of textures or 3D CAD meshes. Model-free methods do not require an a priori model, and typically only care about the 2D segmentation of objects, whereas model-based methods are often after the object's position and orientation in 3D space. Models can either be rigid, perhaps linking several rigid parts together in a kinematic chain, or deformable, like a piece of paper being twisted. In light of this categorization we focus our survey on *model-based 3D pose estimation methods for rigid objects*.

2.2.2 Common methods

Several surveys have been done and we eyed out two that are much cited and one that is very recent. Lepetit and Fua (2005) [25] review model-based 3D tracking of rigid objects. They recognize three families of approaches using edges, templates and point features. Yilmaz et al. (2006) [50] review both detection and tracking, leaning more toward the tracking side of things, and consider four categories of detection algorithms using point features, segmentation, background subtraction, and supervised classifiers. Marchand et al. (2016) [28] review detection algorithms, in the context of localizing a camera for artificial reality, but many of the algorithms are applicable, or were even created for, object tracking. We have summarized the main types of detection methods found in these surveys next.

Edge-based methods try to align the projected contour of a 3D model against edge points in the images, by minimizing the sum of squared distances as a nonlinear optimization problem over the pose parameters. These methods are computationally efficient, but they suffer when the model is geometrically simple and texturally complex, and may get stuck in local minima when the image contains clutter. For more details about these methods, we refer the reader to [25] and [28], which cite several popular papers.

Region-based methods try to fix the problems of edge-based methods by considering the internal structure of the object. For example, template matching uses a texture of the object and searches for its best fit in the input image. We will revisit these methods in Section 2.4. A commonly noted problem with region-based methods is their fragility against unmodelled lighting artifacts and occlusions.

Feature-based methods try to combine the best of edge-based methods and region-based methods, by estimating the object pose from a sparse set of local texture descriptors. The sparsity make these methods robust against noise and occlusions, while not completely disregarding texture gives greater disambiguation power. Moreover, unlike iterative region-based methods, feature-based methods can provide an efficient means of initializing a pose estimate from a single frame, which is a prerequisite for tracking. Region-based methods typically need to be given an initial guess that is reasonably close to the true pose, and as such they need an external initialization process. We refer the reader to [28], which covers classical and recent feature descriptors and matching techniques.

Segmentation-based methods try to divide the image into background and foreground. Classical methods, such as region-growing or thresholding, label pixels as background or foreground based solely on individual pixel colors [47], and can therefore run fast on modern hardware, but are fragile against unmodelled lighting (such as specular highlights or global illumination). Nevertheless, they can detect objects from a single frame, and are viable if one can eliminate false positives and improve their accuracy. We will revisit these classical methods in Section 2.3. Another segmentation cue is *motion*. For example, if the camera is static, one can determine which pixels belong to moving objects by comparing images across frames and "subtracting" the background [19, 41, 36].

Voting-based methods estimate the location of the object center by casting votes from local parts in the image. The Hough transform (see [32] for a survey) is a popular kind of voting method that was originally devised to detect lines, but has later been extended to detect circles and ellipses or even arbitrary 2D shapes. Although it is reputable for its robustness, its computational cost grows impractical quickly for objects more complex than lines or circles, and reducing this cost is still a hot research topic.

2.2.3 IARC team papers

To find solutions for our specific problem, we searched for papers that more or less satisfied the constraints we described in the chapter introduction. The work we found can all, to some degree, be classified under the categories of the previous section. We begin with the most obvious source: the team papers from the IARC symposium.

Huang et al. (2014) [20] attempt to track the iRoomba targets from a downward facing camera. Their method detects targets each frame using a discriminative classifier (SVM) on a sparse feature descriptor (HOG), trained on positive examples of the robots, taken at different viewpoints and lighting conditions, and negative examples of backgrounds. They track targets with a particle filter that uses color histograms as the observation model, and estimate bounding boxes around each target. Their method appears to work reliably, but the documented results are limited to downward facing footage near one meter altitudes, so it is unclear how well it would work from low altitude viewpoints.

Nakamura and Johnson (2016) [33] present a complete system for Mission 7 that (1) identifies and tracks targets, (2) controls a UAV to land on moving targets, and (3) localizes the UAV in the indoor environment. However, they augment the robots with fiducial markers, which indicates that their focus is veered to the tracking and landing problem. Their key contribution on the tracking side is that they consider the three motions modes of the targets by running three Kalman filters and estimating the likelihood of each mode. We doubt we can apply their detection algorithm, since we cannot augment the robot appearances, but their tracking approach is relevant for future work.

2.2.4 Other closely related work

At the time of writing, none of the IARC teams have demonstrated a successful detection and tracking system for the competition, according to the official website, although a team at the Chinese venue (2016) supposedly managed to interact with the robots, but to what degree the interaction was autonomous is unclear since their paper is not published. We therefore look outside the IARC domain for other successful systems, focusing on those that share some key aspects of our problem that we described in the chapter introduction.

Helgesen (2015) [19] and Rodrigues et al. (2012) [41] both use a background subtraction scheme to detect and track moving objects from a flying vehicle. Since the camera itself is moving, they cannot subtract background pixels simply by looking at successive frames, and there is a need to seperate changes in the image caused by the motion of the camera from those caused by moving objects. To do this, both methods estimate the camera motion to predict a flow field of pixel velocities for the background motion. This flow field is compared with a flow field measured by tracking features between successive frames. Pixels with flow vectors that differ from the predicted flow field indicate motion against the background, and are subsequently clustered into objects to be tracked. Both methods rely on objects to contain features that can be tracked across frames, and assume that their motion is sufficiently large to appear different from the background. Their methods may therefore not immediately apply to us, but the idea of comparing flow fields is interesting nonetheless, and could be worth pursuing.

Prisacariu and Reid (2012) [39] combine image segmentation and 2D to 3D pose tracking with a known 3D model in PWP3D. By rendering a signed distance field of the silhouette of a 3D model they obtain an approximate segmentation of the image that is used to (a) iteratively update the pose parameters to maximize the discrepency between a statistical background and foreground model, and (b) update the background and foreground mod-

els using the image data. By adapting the model dynamically their method can be robust against unmodelled lighting effects and noise. However, their models are color histograms, and therefore cannot disambiguate poses of geometrically symmetric objects, even if the texture itself is non-symmetric. Prisacariu and Reid solve the ambiguity by using different views (from multiple cameras), while Tjaden et al. [49] suggest, as an improvement of PWP3D, to include a photometric term in the cost function to include the inner structure. The idea of adapting the model during runtime is relevant to us, since our targets contain a few aspects that are difficult to model statically. If we can accept ambiguity in the estimated heading we could perhaps apply this software directly to our case.

Kalal et al. (2012) [22], similar to the above, also use an adaptive strategy to update a foreground and background model, but represent the object as a set of 2D image patches of positive and negative examples, and propose a learning framework to update the models. Again, the adaptation can include intermittent lighting effects and noise in the model. However, the authors note that the algorithm does not cope well with rotation that is not around the optical axis, and that it only handles single targets. We are curious as to how well it deals with the large viewpoint changes in our case, and how it could be extended to track multiple targets without mixing tracks.

Crivellaro et al. (2015) [11] present a 3D pose detection and tracking method with a fixed model, and obtain robustness against lighting effects and occlusion by splitting the object into distinguishable parts that can be detected individually and combined into a refined pose estimate for the collection. Their method can detect objects from single frames, and appears to be robust against large occlusions and orientations, but runs in 150 milliseconds on a high-end desktop computer, which is an order of magnitude too slow for detection and tracking at video rate.

Choi and Christensen (2012) [7] combine an edge-based tracker with feature matching in a multiple pose hypothesis formulation, based on particle filtering, to track complex (like car doors) 3D objects. By considering multiple pose hypotheses they are robust against false edge correspondences during tracking, which has been a notorious problem for edge-based methods. They note that their method works well in cluttered backgrounds and sparsely textured objects, and outperforms state-of-the-art. Although their method is expensive, the authors suggest that it can be sped up with GPU acceleration.

2.2.5 Summary of overview

In summary, there are lots of possibilites, and judging by figures and graphs they all seem inherently promising, but we think that the success or failure of any particular method is still highly dependent on the context in which it is tested. It would be nice if we could test *all* of the methods we found, but we did not make time for that. We therefore chose two particular approaches that seemed promising, and decided to power through all the subtle problems that followed. In the remainder of this chapter, we present our literature study for these two approaches.

2.3 Detection by color segmentation

Color has long been used as a segmentation cue in classical computer vision tasks where the object appearance is well known. For example, in identifying objects on a conveyor belt under controlled lighting. Although these methods are less applicable outside controllable circumstances, they can offer very fast runtimes and initialization-free detection. In this section, we briefly discuss classical color segmentation techniques, and strategies for coping with uncontrolled lighting.

2.3.1 Thresholding and dealing with lighting

Algorithms for grouping pixels of similiar appearance have been around since the beginning of computer vision. We refer the reader to [47, Chapter 5] for an overview of the most prevalent methods, such as mean-shift, graph-cuts, region-splitting or region-merging. We looked at *thresholding*, which can efficiently label individual pixels as either foreground or background. The simplest form of thresholding is *uniform* thresholding, where pixels are compared against a *constant* value. Foreground pixels can then be merged into objects by some clustering method. Uniform thresholding can easily fail if the lighting is unknown, since specular highlights, shadows and global illumination cause the mean intensity of the image to change, preventing the applicability of a constant threshold.

Global lighting effects, such as the time of day or strobing lights, cause the color of all pixels to change by approximately the same amount. A uniform threshold may still be perfectly able to segment the object, but it would need to change along with the illumination. Histogram equalization or normalization could compensate for global effects [36, Chapter 3], but can be expensive to use each frame. Color space transformations can seperate intensity from hue [36, Appendix 4], and allow for thresholding over illumination invariant colors, but their applicability depends on the energy spectrum of the light and the reflectance properties of the object [18].

Local effects cause abrupt color changes among neighbor pixels and can happen if the surface material is not diffuse. Zickler et al. (2008) [53] isolate specular reflection from diffuse lighting using a linear transformation of RGB values, but they assume the lighting color is known and that the surface reflectance fits a particular model. Shen et al. (2013) [42] isolate specular highlights at 20 Hz on a CPU, which looks promising. Meka et al. (2016) seperate shading from reflectance using dense optimization, and achieve impressive and real-time results, but require an Nvidia Titan X, which is too powerful for an on-board computer.

If none of the above strategies are applicable, the foreground segmentation may get *holes* or fail completely. Holes cause problems for *clustering*, since objects can be superfluously split up and (falsely) give multiple detections. Morphological operators [47, Chapter 3] can fill holes in a second pass, or one could use region-growing strategies to allow for dynamic thresholds during the segmentation [47, Chapter 5]. Alternatively, segmentation and clustering could be done jointly to incorporate spatial coherency in the color segmentation. Mean-shift [30] is one such method, which tries to find clusters in a higher dimensional *feature-space* (e.g. 5D for pixel location and its color), but it has a high computational cost.

2.4 Detection by image alignment

Many detection methods rely on extracting a sparse set of features from the image. An alternative is to forgo this indirection and use the complete image as is. This leads to *direct* methods that, as we will see, can be much more accurate than their indirect counterpart and can succeed in cases where feature extraction fails. The tradeoff we make, in using more data, is the potentially large computational cost and sensitivity to occlusion and unmodelled lighting effects. In this section, we study direct methods for estimating 3D poses, and look at ways to lower the computational cost and increase robustness.

2.4.1 History and applications

Recovering poses directly from images, without the need for an intermediate representation, can be traced back almost 40 years ago to the work of Lucas and Kanade [27], where they present a method that can estimate pose parameters directly from pixel intensities. Variants of their method has since been used in a wide range of applications like virtual and augmented reality, video stabilization, object tracking, visual odometry and SLAM.

One application of the Lucas-Kanade method (LK), is *direct image alignment*, in which the goal is to minimize the visual difference between two images by warping one into the other [46]. If both images are photographs taken by a camera from different viewpoints, the LK method can be used to estimate the relative camera pose between the images. This is related to visual odometry, which aims to track the motion of a camera using video. Visual odometry has traditionally been solved using feature extraction and matching across images, but the advent of high framerate cameras and increased computational power has made direct methods an attractive alternative [1]. A similiar problem is visual SLAM, which aims to simultaneously recover the 3D structure of the environment. Engel et al. [17, 16] introduce *dense* [3] visual SLAM, which uses the LK method to track the camera and recover a dense 3D model of the environment. In both of these cases, a direct approach gave higher accuracy and robustness over an indirect, feature-based method.

Direct image alignment can also be used for model-based object tracking (See [28, Chapter 4.2] for a brief survey). In particular, if one of the images is a photograph of an object, and the other is a reference image of the same object, the LK method can be used to estimate the pose of the object in the photograph. The reference image could be static, or dynamically generated. The latter is sometimes referred to as *Analysis by Synthesis* or *Inverse Rendering* [26], in the sense that the object pose is estimated by *synthesizing* or *rendering* an image of the object as it would have been seen by a camera, and comparing the result with the observed photograph. It is also referred to as *generative reconstruction* [40], in the sense that the object pose (or other high-level information) is reconstructed by *generating* an image that, again, is compared with the observation. In any case, the comparison produces an *error*, defined by some appropriate metric, that is used to iteratively update the pose estimate. Aside from the object pose, the estimation variables may include any aspect of the image formation process, such as surface material properties, the location and intensity of light sources, or camera calibration parameters.

[3] See [16] for a description of the terms direct, indirect, sparse and dense.

Object tracking with direct image alignment is becoming an attractive alternative over feature-based methods among some researchers, who argue that extracting and matching features has a tendency to fail outside of ideal conditions [10]. For example, sparsely textured surfaces (such as the clean-looking iRoombas) or motion blur (a common artifact from vibration on UAVs) may prevent the extraction step from finding enough features [10, 29]. On the other hand, feature-based methods can provide global detections from a single frame, whereas direct methods are local and require an initial guess. Szeliski [46, Chapter 4.4] suggests that this property makes direct methods useful for *refining* the result from a global method (such as color thresholding) to compute a more accurate pose.

Since we probably need very accurate 3D pose estimates and our solution must deal with large viewpoint changes, vibrations and objects with mostly flat textures, pose refinement by direct image alignment seemed like a good idea to pursue. We therefore dedicate the remainder of this chapter to a study of this method, presenting first the basic methodology, and proceed to discuss extensions that make the method robust in real-life scenarios.

Our introduction is based on the following sources: Irani and Anandan (1999) [21] give an introduction to the basics of direct parameter estimation on pixel intensities; Szeliski (2006) [46] gives a more thorough exposition and a comparison with feature-based methods; Matthews et al. (2002-2004) [4], in their five-part series, present efficient formulations (1), robust cost functions (2), models for linear appearance variations (3), priors on parameters (4) and extensions to non-planar surfaces (5). As we continue through the chapter, we will refer to more recent work for particular improvements that are not covered in the above.

2.4.2 Pose estimation with image alignment

This section introduces the basic concept of using image alignment, or inverse rendering, for parameter estimation. We assume that we have a photo of an object taken by a camera, that a reference model of the same object is available (for example, a textured 3D mesh or a 2D texture), and that there exist some *true* parameters of an image formation process (that we define) such that the synthesized image looks similiar to the observed photo. Our goal is to estimate these parameters.

We will first consider the basic geometric image alignment problem, similiar to that of Lucas and Kanade [27], where the model is a 2D texture and the unknowns of the image formation process is limited to a deformation of the model into the photo, in other words, intrinsic camera parameters, scene lighting, etc, are assumed to be known. Let $\mathcal{I} : \mathbb{R}^2 \to \mathbb{R}$ be the photo taken by the camera, henceforth referred to as the *input*, and let $\mathcal{M} : \mathbb{R}^2 \to \mathbb{R}$ be the *model*. The input and the model are sampled at subpixel locations to produce grayscale intensities, using some sampling scheme like bilinear or bicubic interpolation. Let $\mathcal{W}(t;\theta) : \mathbb{R}^2 \to \mathbb{R}^2$ be the *warp* function, which takes a pixel $t \in \mathbb{R}^2$ in the model coordinate frame and maps it to a subpixel location $s \in \mathbb{R}^2$ in the input coordinate frame, for a vector of warp parameters θ. For example, if the object is located at an unknown translation $\theta = (\theta_1, \theta_2)$ in the image, but is otherwise correctly scaled and rotated

according to our reference model, the warp may be

$$\mathcal{W}(t;\theta) = \begin{bmatrix} u + \theta_1 \\ v + \theta_2 \end{bmatrix}$$

In general, the warp can have an arbitrary number of parameters. For instance, a planar rotation and translation can be described with three parameters, and the transformation a plane undergoes when projected and viewed from two different viewpoints can be described with six [47, Chapter 2]. Moreover, the warp can include other geometric processes, like intrinsic camera calibration parameters describing lens distortion, or deformation parameters describing non-rigid surfaces [35], but it is unable to describe photometric processes, like lighting or motion blur. In Section 2.4.3 we discuss appropriate warps for 3D pose estimation.

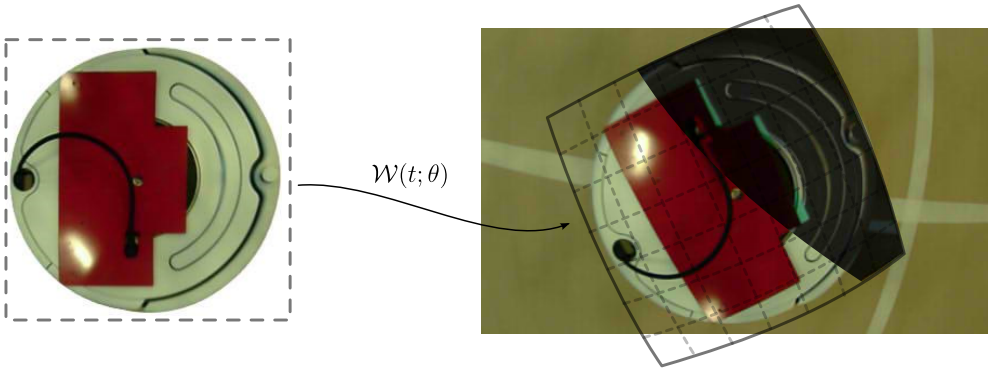


Figure 3: The warped model is compared with the input, and an error is computed at each pixel. The true warp parameters are assumed to coincide with the global minimum of the cost function.

Once an appropriate warp function has been decided, the goal is to find the best parameters, such that the warped model looks most similiar to the photographed object, in terms of a similarity metric. One of the simplest similarity metrics is the sum of squared intensity differences (SSD), which assumes that the pixel intensities are constant under the warp, a property referred to as *brightness constancy* [21]. In other words, it assumes that the model appears *as is* in the photo. As we discuss later, this assumption rarely holds in practice, and there are alternative similarity metrics that are more robust against unmodelled effects.

In terms of the SSD similarity metric, the goal is to minimize the sum of squared intensity differences between the warped model and the input, with respect to the parameters θ

$$\min_{\theta} E(\theta) = \min_{\theta} \sum_{t_i \in \mathcal{D}_{\mathcal{M}}} \left(\mathcal{I}\left(\mathcal{W}(t_i;\theta) - \mathcal{M}(t_i)\right)\right)^2 \tag{1}$$

where $t_i \in \mathcal{D}_{\mathcal{M}}$ is the set of pixels in the model texture (texels). The objective function E is sometimes referred to as *photometric error* [1] or an *energy function* [48]. This is a *nonlinear* least squares problem, because there is no inherent linear relation between image intensities and pixel coordinates; they are, in general, unrelated quantities. Hence, we cannot immediately solve the equation for θ, as one can for *linear* least squares problems.

One way to solve Eq. (1) is to simply search over all possible values of θ and choose the one with the lowest error. While this might be a feasible strategy for small problems, an exhaustive search will quickly become impractical as the number of pixels and parameters increase (see [9] for a complexity analysis). Although graph-based approaches can perform efficient global searches using dynamic programming [6], we do not study them in this report.

If a global search is infeasible, a *local* method is the clear alternative. The Lucas-Kanade method is an example of a local method, and solves Eq. (1) by iteratively approximating the error by a quadratic around the current parameter estimate, and solving a weighted linear least squares problem at each iteration [4]. In particular, let $\hat{\theta}$ be the parameter estimate at the current iteration, and let $\delta\theta$ be a small update. Local methods aim to minimize

$$\sum_{t_i} (\mathcal{I}(\mathcal{W}(t_i; \hat{\theta} + \delta\theta) - \mathcal{M}(t_i))^2$$

with respect to the update $\delta\theta$. The Lucas-Kanade method linearizes the error terms with a first order Taylor expansion around $\hat{\theta}$ to obtain the quadratic error approximation

$$\sum_{t_i} (\mathcal{I}(s_i) + \frac{\partial \mathcal{I}}{\partial s} \frac{\partial \mathcal{W}}{\partial \theta} \delta\theta - \mathcal{M}(t_i))^2 \tag{2}$$

where $s_i = \mathcal{W}(t_i; \hat{\theta})$ are pixels warped using the current parameter estimate, $\frac{\partial \mathcal{I}}{\partial s} = \nabla \mathcal{I}$ is the *image gradient*, and $\frac{\partial \mathcal{W}}{\partial \theta}$ is the *Jacobian* of the warp function[1] . Intuitively, the image gradient describes how the image changes with respect to the pixel coordinates, and the warp Jacobian describes how pixel coordinates change with respect to the warp parameters. The validity and accuracy of the linearization

[1] This terminology is commonly found in the literature [46].

$$\mathcal{I}(\mathcal{W}(t_i; \hat{\theta} + \delta\theta) \approx \mathcal{I}(\mathcal{W}(t_i; \hat{\theta})) + \frac{\partial \mathcal{I}}{\partial s} \frac{\partial \mathcal{W}}{\partial \theta} \delta\theta$$

is a concern that has been brought up in recent literature[2] . Naturally, we cannot expect the linearization to hold for large displacements, but in a naive implementation the range of validity can be impractically small. As we discuss in Section 2.4.5, this issue can be mitigated with hierarchical estimation. Nevertheless Eq. (2) is a *linear* least squares problem, and can readily be minimized by differentiating and equating to zero, giving the solution

[2] See Bristow [6] and Alismai [1].

$$\delta\hat{\theta} = -\left(\sum_i g_i g_i^T \right)^{-1} \sum_i g_i r_i \tag{3}$$

where

$$g_i^T = \frac{\partial \mathcal{I}}{\partial s} \frac{\partial \mathcal{W}}{\partial \theta} \bigg|_{s=s_i, \theta=\hat{\theta}} \tag{4}$$

$$r_i = \mathcal{I}(s_i) - \mathcal{M}(t_i) \tag{5}$$

As the reader might notice from Eq. (3), the Lucas-Kanade method is similiar to the *Gauss-Newton* method for solving nonlinear least squares problems [47, 37]. Following common optimization terminology, g_i and r_i are called the *gradients* and the *residuals*, respectively, and the symmetric matrix

$$\sum_i g_i g_i^T$$

is the (Gauss-Newton approximation of the) *Hessian* of the objective function Eq. (1). The solution Eq. (3) is the step that leads to the extremum of the quadratic approximation, and is called the *Gauss-Newton step direction.* One concern with the Gauss-Newton method is that the Hessian may not be positive definite (it could even be negative definite, in which case the step is in the direction of *increasing* error). The *Levenberg-Marquardt* method is a popular modification of the Gauss-Newton method that adds a damping matrix to the Hessian to ensure positive definiteness [37], and has been discussed in terms of the Lucas-Kanade algorithm in [4, Part 1]. Finally, the Lucas-Kanade method repeatedly solves Eq. (3) for a small parameter update, and adds the update to the current estimate

$$\hat{\theta} \leftarrow \hat{\theta} + \delta\theta$$

until some convergence test applies, for example, that $||\delta\theta||$ is sufficiently small [4].

In summary, the Lucas-Kanade method estimates a set of geometric deformation parameters, such that a model is aligned with a region in the input photo. The alignment error is represented by a photometric error taken over each pixel in the model and the corresponding warped pixel in the input, and is minimized by linearizing the residuals with respect to the warp parameters, and iteratively solving for small parameter updates. There are several concerns with this approach that we briefly summarize below and discuss in the remainder of this chapter.

Robust cost functions: The sum of squared intensity differences (SSD) as a similarity metric relies on the brightness constancy assumption. In particular, it assumes that surfaces are Lambertian and that there are no unknown shading effects. Consequently, SSD is not robust against outliers and therefore tends to perform badly in practice [10]. In Section 2.4.6 we discuss robust estimation techniques for rejecting outliers, and in Section 2.4.7 we discuss alternative similarity metrics that can be made robust to certain lighting effects.

Efficient formulations: The process presented in this section requires recomputing the Hessian at each iteration (except for simple warps), which can be costly. In Section 2.4.4 we discuss alternative formulations of the geometric image alignment problem, which allow for significant speedups by precomputing parameters.

Coarse-to-fine estimation: The accuracy of the first order linearization can be very low for detailed images or large pixel displacements. This means that the initial guess for the parameters must be very close to the correct solution in order to avoid getting stuck in local minima (incorrect solutions). In Section 2.4.5 we discuss hierarchical strategies, commonly called *coarse-to-fine* estimation, that mitigate this issue while dramatically improving speed and widening the region of convergence.

Beyond simple warps: We assumed that all other aspects of the image formation process were known. In some cases, we can benefit from including other aspects of the image formation process as optimization variables. For example, Engel et al. [16] include intrinsic camera calibration parameters in their optimization to allow for online refinement. Park et al. [38] explicitly model motion blur to track 3D objects undergoing fast motion, or similarily, fast camera motion or vibration (a common problem on MAV platforms).

Complex models: The model presented in this section was a 2D texture. In general, the model can be as simple as a textured plane, such as a landing pad [29], or it can be as complex as a deformable triangular mesh, such as a golf ball hitting a metal bat or a flowery cushion being impaled by a pen [35]. In these cases, reference models can easily be made manually, for example, by taking a photograph or modelling a 3D mesh. If the model includes many complex parameters, like shading coefficients, generating the reference model by hand can be impractical. An alternative is to automatically generate reference models from examples using machine learning techniques. For example, Thies et al. [48] use image alignment to track facial expressions in a monocular RGB video, and use thousands of parameters to model the geometric shape and reflectance properties. Reference faces are learned in an off-line process using principal component analysis, and are recalled during run-time for tracking.

Sampling: In practice, the image gradients are computed *stochastically* by finite differences (like central differences or Sobel filtering). What method is most accurate? Moreover, sampling the images with bicubic sampling can produce more accurate results, but is more costly. Is it worth it? Alismai [1] studies various choices that are not commonly adressed, for instance, whether bicubic sampling is worth the additional cost over bilinear sampling, and how different gradient estimation methods fare against each other. Although their conclusion appears to be that bicubic sampling does not hold any advantage over bilinear sampling (and is therefore not worth the additional cost), and that Sobel or Scharr filters had no appreciable advantage of central differences, it is nonetheless important to be aware of the choices that exist.

Photometric calibration: Pixels in the image are not purely functions of incoming light hitting the camera sensor. The imaging process of digital cameras contains numerous noise sources, such as value quantization, additive noise due to thermal properties of the elctronics (camera read noise), the darkening of edges (vignetting), or the distortion caused by (for example) a rolling shutter. As noted by Newcombe [34], geometric calibration is well understood and almost always performed, while photometric calibration remains widely ignored. Although feature-based methods are robust against this omission, it immediately breaks the brightness constancy assumption for direct methods [34]. If the reader is interested, Szeliski [47, Chapter 10] reviews many aspects of photometric calibration. A practical application to dense visual SLAM can be found in [34], where they use a simple photometric camera model that incorporates exposure time, vignetting, read noise and a gamma function.

2.4.3 Warp functions for pose estimation

In order to estimate the 3D pose of a rigid-body object, the warp must describe how points on the object surface are projected into the image to form pixels. A *homography* describes this process end-to-end, by assuming a pinhole projection [47, Chapter 2], but in doing so, it is not easily extended to arbitrary camera projections or constrained 3D poses, nor is it directly applicable to non-planar objects. Alternatively, the warp can seperate the two transformations, describing first a 3D-to-3D rigid-body transformation, and a 3D-to-2D image projection. Together, they define the geometric transformation from surface

points in the model to pixels in the photo. As alluded to in the previous section, both of these transformations can be included in the parameter estimation, but for now we assume that the projection is fixed, while the rigid-body transformation contains the unknown parameters. We refer the reader to either of [44, 8, 47] for an introduction to rigid-body transforms, and to [12] for a review of warp functions in the Lucas-Kanade framework.

Estimating rigid-body transforms is no simple ordeal, as the 4×4 matrix does not lend itself well to optimization problems [5], due to the constraints needed to maintain a valid transformation matrix. We therefore prefer a *minimal parametrization*, as these can be estimated without imposing constraints. Although parametrizing translation is simple, parametrizing rotations gets quite hairy. Discussions around this can be found in any book concerning rigid-body motion, such as [8, 47], suffice to say, the common approach is to use either Euler angles, axis-angle or unit quaternions.

Several authors [14, 49, 17, 16, 7, 43, 34] use an axis-angle parametrization — or in more common terms, the Lie algebra of the Lie group of rigid-body transformation matrices — to represent the small pose update that is iteratively composed with a 4×4 absolute transformation matrix. The axis-angle vector is a minimal parametrization, and is therefore easily applicable for unconstrained optimization. Some authors [39] use quaternions, and noticed that, although they have the disadvantage of being overparametrized and requiring renormalization, they were a better alternative than axis-angle in their case. However, there appears to be disagreement on this, since [49] later claim that their implementation of the software in [39] performed better with axis-angle.

2.4.4 Efficient formulations

There are many ways of approximately solving Eq. (1), the iterative Lucas-Kanade method being one of them. The LK method is sometimes referred to as a *gradient-based* approach, in contrast to *graph-based* approaches [6]. Although gradient-based approaches have many desirable properties, such as implementation simplicity and computational speed, they are only *local* methods and require a decent initial estimate. On the other hand, while an exhaustive brute-force search is usually out of the question, efficient graph optimization methods, like dynamic programming, can be applied to solve for the globally optimal parameters in polynomial time, if certain constraints can be imposed on the parameters [6].

Aside from the original formulation of the Lucas-Kanade method [27], it was later discovered that the cost function could be reformulated in four different ways, depending on the direction of the warp (forward or inverse), and how the parameters are updated in each iteration (additive or compositional) [4]. This resulted in four new formulations: Forward Additive (FA), Inverse Additive (IA), Forward Compositional (FC) and Inverse Compositional (IC). These methods are reviewed by Baker et al. (2004) in [4], where it was shown that IC allows the Hessian and the gradients to be precomputed once, as opposed to recomputed in each iteration, allowing for significant performance increase. The Efficient Second Order Minimization (ESM) method, introduced a couple of years later, aims to combine the best of inverse and forward composition, and is discussed in [9] along with a review on all five methods.

It was discussed in [4, Part 5] that the IC algorithm can easily be extended to the matching of 3D volumetric textures, but doing so for 3D planar surfaces violates one of the underlying assumptions. This is unfortunate, since it is exactly the case where the warp combines a 3D pose transformation followed by a 2D projection. On the other hand, although their implementation details are not available, Crivellaro et al. [10] use both IC and ESM in, what appears to be, the 3D pose estimation scenario. We therefore suspect that it is possible, perhaps with some approximation, to apply IC to the 3D estimation problem, but we have not looked into this any further.

2.4.5 Hierarchical parameter estimation

At the heart of the Lucas-Kanade method is the linearity assumption, whose validity affects the size of the convergence region. Intuitively, it requires that the image intensity at nearby pixel is linearly related to the pixel displacement

$$\mathcal{I}(s + \delta s) \approx \mathcal{I}(s) + \nabla \mathcal{I}\, \delta s$$

For this approximation to hold, it is necessary that the image is sufficiently smooth, or likewise, that the displacement does not exceed the region in which the linearity holds. For images with high frequency content, this region can be quite small, and as such the initial parameter estimate must be very close to the correct solution. This can be limiting in practice, since if such a good initial estimate were available, we might not have a need for improving it in the first place!

The allowable pixel displacement can be improved by filtering away high-frequency components in the image data. This is used in most image alignment methods, and goes under the name of hierarchical or *coarse-to-fine* estimation [46, 4]. A common strategy is to generate power-of-two image pyramids, each level in the pyramid being a filtered and downsampled version of the previous level [29]. The highest detail level is the original image, the second-highest is downsampled by a half in each dimension after filtering, and so it repeats for a specified number of levels. The number of levels appears to depend on the use case. Szeliski claims that important details are blurred away after two or three levels [46], although Engel et al. [17] noted that starting at a very low resolution of only 20×15 pixels helps to increase the convergence radius.

The coarse-to-fine strategy is not restricted to the image data; it can also be applied to the complexity of the parameter variables. As suggested in [45, 46], Martinez et al. [29] apply the coarse-to-fine strategy to both the input images *and* the motion model. Their motivation is that the lowest detail of the image pyramid does not contain enough information to estimate a large number of parameters, thus leading to unstable behaviour. The authors use one motion model per level in the image pyramid, starting with translation (2 parameters) for the lowest detail level, followed by translation+rotation (3 parameters), a similarity transform (4 parameters) and a homography (8 parameters) for the most detailed level.

One method of effectively decreasing the degrees of freedom, without the additional complexity of managing multiple motion models, is to augment the cost function Eq. (1) such that certain parameters of the update are encouraged to stay close to zero, as proposed

in [4]. For example, we can penalize updates away from zero, or equivalently, penalize solutions far away from the previous iterate, by adding a quadratic term to the total cost:

$$E(\theta) = \sum_{t_i} \underbrace{\left(\mathcal{I}(\mathcal{W}(t_i;\theta)) - \mathcal{M}(t_i)\right)^2}_{\text{Data}} + \underbrace{(\theta-\hat{\theta})^T R(\theta-\hat{\theta})}_{\text{Regularizer}} \tag{6}$$

This procedure is often called *regularization* [48], and the latter term is called a *regularizer*, whose role is to encourage certain properties of the solution, distinguishing it from the *data* term, whose role is to maximize fitness against the data. The term is also called a *prior*, when the cost function is built from statistical principles, in which case it represents the a priori probability of certain solutions.

2.4.6 Robust parameter estimation

It is often stated that the quadratic cost function Eq. (1) is not robust against outliers [45, 47, 51]. This is troublesome, especially in computer vision, because measurements are often corrupted with few, but large, outliers, caused by the presence of multiple *structures* [45]. For example, outliers of one structure, say an odd piece of white near an iRoomba, can be the inliers of another, say a piece of tape or a specular highlight.

Robust estimation techniques were designed to cope with outliers. We refer the reader to [51] and [45] for a review of the main techniques, *RANSAC*, *Kalman filtering* and *M-estimators* being some of them. RANSAC is a random sampling technique and is very popular in feature matching [25], but has, to our knowledge, not been used in the Lucas-Kanade framework. M-estimators, on the other hand, are prevalent in many papers we found, and replaces the quadratic cost function with a subquadratic curve ρ

$$\sum_i r_i^2 \quad \rightarrow \quad \sum_i \rho(|r_i|)$$

The new cost function can be written as the *iteratively reweighted least squares* problem [4]

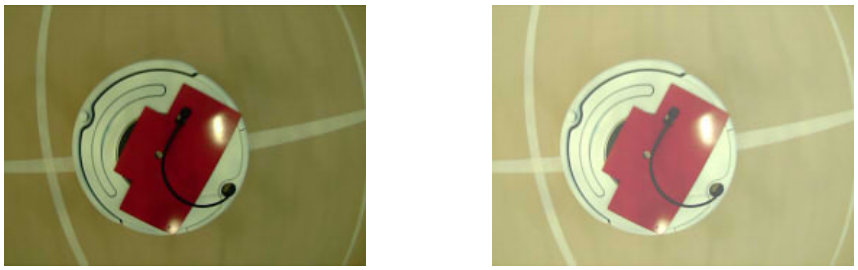
$$\sum_i w_i r_i^2 \tag{7}$$

where the weights $w_i = \rho'(|r_i|)/|r_i|$ are treated as constants during each iteration and alternatingly recomputed as θ is updated. Some common choices for ρ are discussed in [51]. In our survey we find Cremers et al. use the Huber weighting in their direct visual SLAM system (LSD-SLAM); Thies et al. [48] use the L2,1 norm on photometric residuals in their face tracker; Ngo [35] consider both Huber and Tukey weighting; Alismai [1] suggest using the Huber weighting for coarse registration, and shifting to Tukey upon convergence.

M-estimators like Huber and Tukey have a tuning constant which is related to the breaking point at which residuals are viewed as likely outliers. According to Zhang [51, Chapter 9.4], this constant should be chosen based on a robust estimate of the residuals' standard deviation to obtain optimal statistical efficience, that is, to achieve the same efficiency as a least-squares in the presence of only Gaussian noise.

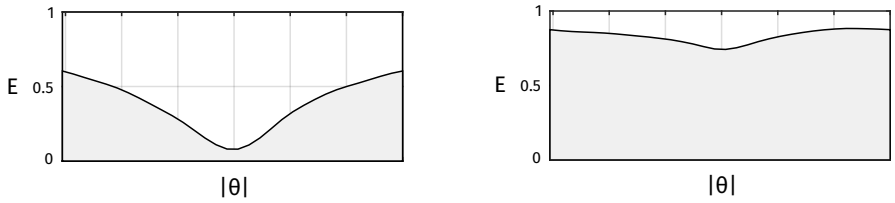
2.4.7 Dealing with lighting and occlusion

The robust estimation techniques in the previous section might not help if most of the data appear as outliers. To illustrate, consider the image shown in Figure 4a, which has been corrupted by constant gain and bias to produce the image in Figure 4b. This corruption could, for example, be caused by dynamic exposure in the camera. Figure 5 shows the cost function surface with the Huber-weighted intensity difference metric, where the model is cropped from Figure 4a and edited to remove the shiny bits, and the warp is a 2D translation θ. Notice how, although the correct global minimum is still present, the signal to noise ratio is much lower for the corrupted case, which makes it difficult to distinguish true alignments from false alignments.



(a) Original **(b)** Brightened

Figure 4: Original input image (left) corrupted with a constant gain and bias brightness effect (middle).



(a) On original input **(b)** On brightened input

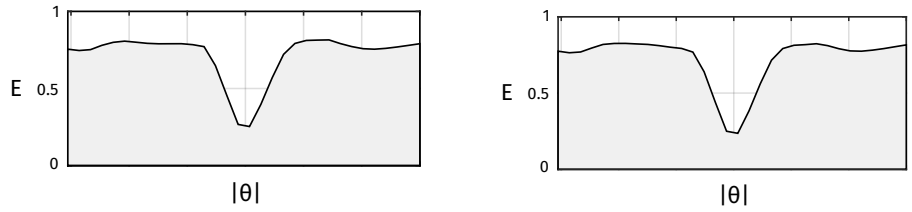
Figure 5: Huber-weighted intensity difference cost surfaces for Figure 4a (left) and Figure 4b (right).

One way to cope with lighting effects is to explicitly include them in the image formation model. The gain and bias model, introduced by Lucas and Kanade [27] and discussed in part 3 of [4], assumes input pixels are corrupted by a common, but unknown, multiplicative and additive term. Estimating these additional parameters requires linear regression, which can be costly [46]. Moreover, it is unable to describe local effects, like specular highlights, although this could be partially solved by combining multiple local gain and bias models. Alternatively, if an accurate map of the incoming lighting (an irradiance map) were available, one could more accurately predict the object appearance using some surface illumination model like Blinn-Phong or Cook-Torrance. Thies et al. [48] and Ngo [35] represent scene irradiance with Spherical Harmonics coefficients, and estimate the coefficients during runtime to realistically render a tracked surface. Silviera and Malis [43] use radial basis functions and demonstrate tracking even under strong specular highlights.

Another option is to preprocess the input to obtain a lighting invariant representation, that can then be aligned with the model under the brightness constancy assumption. For example, global gain and bias can be removed by preprocessing the input to have zero mean and unit variance, while local effects can be removed by bandpass filtering [34]. However, if the problem is that the brightness constancy assumption does not hold, maybe we shouldn't rely on it in the first place, and use a different similarity metric instead. Several alternatives have been suggested in the literature, for example, the *Normalized Cross Correlation* (NCC) and *Mutual Information* (MI) metrics. Both are robust against extreme local effects, but are complex to implement and costly [34]. *Sum of Conditional Variances* (SCV) tolerates less extreme effects, but is simpler to implement [12].

Aside from the similarity metric, we could replace the image intensities themselves. This is the idea behind *dense feature descriptors*, which replace intensities by a neighborhood operator. The simplest example of which is the image gradient, which is invariant to constant biases (since the bias disappears in the finite difference). Several descriptors are described and compared by Ngo [35] and Alismai [3]. Crivellaro et al. (2014) [9] presented *Gradient-Based Descriptor Fields* (GBDF) and demonstrated robustness against strong specularities and global lighting. Alismai (2016) [2] noticed that GBDF does not handle rotations and created *Bit-Planes* (BP), which performs similiarily in the no-rotation case, but outperforms it otherwise. Meanwhile, Ngo (2016) [35], presumably unaware of BP, suggests a fix that makes GBDF rotation invariant.

To illustrate the potential advantage of dense descriptors, consider Figure 6, which shows the cost surfaces for the previous example for the Huber-weighted GBDF metric. Unlike the intensity metric, GBDF is more or less invariant to global gain and bias effects, and the signal to noise ratio is clearly preserved.



(a) On original input **(b)** On brightened input

These are not just the same figure included twice (we checked)

Figure 6: Huber-weighted GBDF cost surfaces for Figure 4a (left) and Figure 4b (right).

Dense descriptors can be combined with any similarity metric, like SSD, MI, NCC or M-estimators. We may therefore write the cost function Eq. (1) in a more general form

$$E(\theta) = \sum_i d\Big(\phi_{\mathcal{I}}(\mathcal{W}(t_i;\theta)), \phi_{\mathcal{M}}(t_i)\Big) \tag{8}$$

where d is the similarity metric, and ϕ is the feature descriptor. For example, $\phi_{\mathcal{I}}(s) = \mathcal{I}(s)$ and $d(\phi_a, \phi_b) = \frac{1}{2}(\phi_a - \phi_b)^2$ gives the usual sum over squared intensity differences. For a dense desciptor, ϕ may be a vector valued function, and d will need some distance metric on the vector space, like the Hamming distance [2] or the Euclidean distance [9].

The validity of linearizing dense descriptors is a concern that has been brought up in literature. Bristow and Lucey (2016) [6] arrive at the surprising result that (a) dense features are in-fact well suited for gradient-based optimization and (b) performs better than pixel intensity matching, despite the Taylor approximation being less accurate. Alismai et al. (2016) [3] compare various dense descriptors and claim that they are well suited for linearization.

Ngo [35] compares several combinations of descriptors and similarity metrics, and finds that MI and NCC on intensities, although robust against errors, suffer from many local minima and narrow convergence regions, indicating that they are not robust against poor initial guesses. Meanwhile, GBDF coupled with an M-estimator is both robust and has a wide convergence region. SSD on intensities was the worst performer, and failed to converge in many cases.

Finally, although dense descriptors are applicable as lighting invariant descriptors, they fail to cope with occlusion. Ngo (2016) [35] reviews several approaches, and presents a real-valued relevancy score that predicts which pixels of the model are occluded, using previously good alignments. However, such occlusion masks (in particular, binary masks) can cause problems for linearization at occlusion boundaries. Rhodin et al. (2015) [40] represent opaque objects as translucent media with a smooth Gaussian density distribution, turning visbility into a smooth and differentiable phenomenom, but they require the occluder to be part of the estimation model (for example, if the object is a kinematic chain of body limbs, or if the occluder is a seperate, but known, object).

2.5 Summary

Currently, many pose estimation methods rely on detecting and matching point features. These methods are popular thanks to their robustness against occlusions, lighting effects and camera calibration errors (photometric ones in particular). Unfortunately, these methods tend to break down when the surface of interest is sparsely or repetitively textured or blurred or out of focus.

Direct pixel-based methods overcome these problems by dense matching on multiple levels of detail, and are becoming an attractive alternative to feature matching, with promises of higher accuracy and robustness in cases where feature extraction fails. However, these methods can be computationally expensive and require a good model of the image formation process. Moreover, dealing with occlusion and lighting is regarded to be more difficult.

In this section we introduced the Lucas-Kanade method as a direct pixel-based approach to pose estimation, and discussed many extensions to improve computational speed and robustness. The method requires an initial pose estimate, so we briefly discussed color segmentation for the purpose of isolating the colored top plates of the targets, although any of the other related work could also be used to generate initial guesses.

3 Our solution

3.1 Overview of our solution

Our solution to the target detection problem uses the theory on direct image alignment and color segmentation presented in the previous chapter. After trying several classical approaches, like feature matching or circle detection, but struggling to achieve the accuracy and robustness necessary for the IARC mission, we found that image alignment gave promising results immediately, and was therefore an approach that we pursued. To initialize the image alignment, we segment the top plates based on their distinct color and compute coarse initial pose estimates by inverse projecting their pixel centers back into the world, using the camera height and orientation measured by on-board sensors.

The general flow of our solution is given below.

Step 1 — *Image retrieval and preprocessing*: Video frames are processed into image pyramids of power-of-two size reductions, with a fixed number of levels of detail. The color segmentation and the image alignment use different data formats and therefore have different preprocessing steps. These are described in more detail in the respective method's chapter.

Step 2 — *Detection by color segmentation*: Each frame is scanned for pixels that are sufficiently red or green, corresponding to the colored top plates of the targets, and the resulting pixels are grouped into clusters by connected components. This method provides global detection, but suffers from false positives and lacks the accuracy needed to perform precise interactions.

Step 3 — *Refinement by direct image alignment*: A textured model of the target is fitted to the input image by directly minimizing the weighted sum of per-pixel photometric error between the input image and the reprojected model, using iterative weighted Gauss-Newton optimization. This method eliminates false positives from the above detector and also estimates more accurate 3D poses.

Step 4 — *Tracking*: Although we did not implement this yet, tracking naturally fits at the end of this pipeline, as it can improve robustness, performance and is also necessary to estimate target velocity. For example, the image alignment lends itself particularly well to tracking, as estimates can be initialized from previous frames and thereby require fewer iterations, since the targets don't move much between successive frames.

This chapter presents details regarding our color segmentation and direct image alignment implementation. We model the image formation process for our camera in Section 3.2, as it is required for direct image alignment, and describe our camera calibration procedure in Section 3.3. We then describe our implementation of the Lucas-Kanade method for local pose refinement in Section 3.4, and our algorithm for global detection based on color segmentation in Section 3.5.

3.2 Modelling the image formation process

Direct image alignment requires a model of our image formation process, which describes how the target robots appear in the image when viewed through our fisheye camera. We omit lighting and shading, and model the targets as static textured planes that are geometrically warped into the input image by a 3D-to-2D rigid-body transformation, described in Section 3.2.2, and a 3D-to-2D camera projection, described in Section 3.2.1.

3.2.1 Camera models

One of the simplest camera devices is called the *pinhole camera*, and consists of a light-tight box with a tiny hole at one end. When uncovered, the hole will allow light to pass through and hit a piece of photographic paper. Despite its simplicity, the pinhole camera is still useful as a model of many cameras of today, even though the insides of a modern digital camera are substantially different from the traditional pinhole box. In the pinhole model, points in the scene are mapped to points in the image through *perspective projection* by intersecting rays with the *viewing plane*. This mapping is often derived from the triangle similarity shown in Figure 7a. In particular, let (x, y, z) be the 3D point in the camera frame and (u, v) be the projected image point, then the pinhole perspective model gives:

$$\frac{(u - u_0)}{f} = \frac{x}{-z}$$
$$\frac{(v - v_0)}{f} = \frac{y}{-z}$$

where f, the distance between the camera and the viewing plane, is called the *focal length*, and (u_0, v_0) is the principal point. This mapping can also be expressed in polar coordinates:

$$u = r \cos \phi$$
$$v = r \sin \phi$$

where ϕ is the angle of the projected point in the uv plane. For the pinhole projection, ϕ is identical to the incident ray's angle in the xy plane, while the distance from the image center is proportional to the tangent of the incident angle:

$$r = f \tan \theta$$

Written in this form, the drawback of the pinhole model becomes apparent: The projected point blows up to infinity as the incident angle tends to 90 degrees. We would need an infinitely short focal length, or an infinitely large lens, to describe cameras of 180 degree FOV using the pinhole model. To allow for high field of view — without infinitely large lenses, which would hardly be suitable for a MAV — we need to abandon the notion of intersecting rays with a viewing plane, and instead consider a *viewing sphere*. Therefore, a more suitable model of projection, illustrated in Figure 7b, is one that maps intersection points on the viewing sphere, described by spherical coordinates (ϕ, θ), to points in the image, described by polar coordinates (ϕ_{uv}, r). Figure 8 illustrates the relation between spherical coordinates and the intersection point from the incident ray.

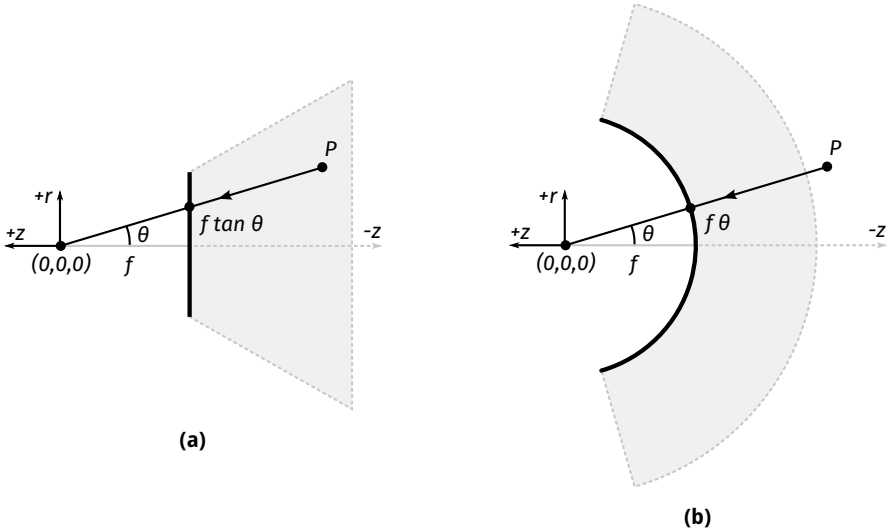


Figure 7: The traditional *pinhole* projection model (a) is unable to describe cameras with very high field of view, as the image coordinate blows up to infinity when the incident angle, θ, tends to 90 degrees. On the other hand, the *equidistant* projection model (b) is able to describe cameras with a field of view of 180 degrees and beyond. This is done by considering the intersection point with a viewing sphere, rather than a viewing plane.

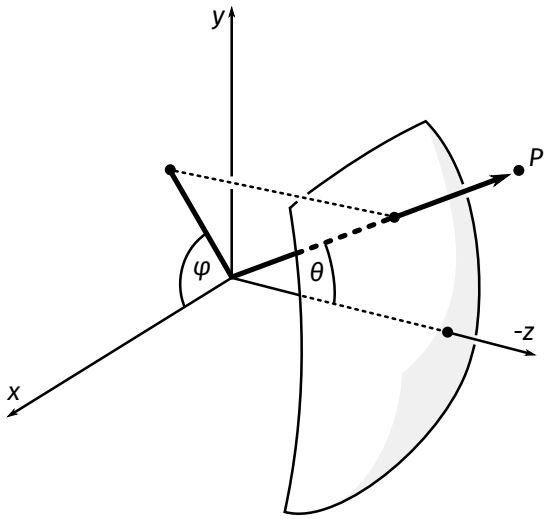


Figure 8: In a generalized camera projection, a point P is projected onto the lens by casting a ray toward the camera origin, intersecting the unit sphere along the path. The intersection point can be described by the *incident angle* between the ray and the optical axis, θ, and the angle in the xy plane, ϕ. A projection model maps the unit sphere coordinates θ, ϕ to a set of polar coordinates on the lens.

When expressed as a mapping from spherical coordinates on the viewing sphere to polar coordinates on the image plane, it is easy to extend the projection to accept arbitrary incident angles. The simplest model of which is the linear relation:

$$r = f\theta \tag{9}$$

for some constant f, which maps the incident angle linearly to the lens. This model, called the *equidistant* or *equiangular* mapping, avoids singularities at ±90 degrees, and can even map the entire sphere of incoming light onto a finite lens. In practice though, the large compression near the edges makes it difficult to extract any meaningful information from large angles, for this particular model. Some alternative fisheye models are [8, p. 271]:

$r = 2k \tan(\theta/2)$ — Stereographic projection

$r = 2k \sin(\theta/2)$ — Equisolid projection

$r = k \sin(\theta)$ — Orthogonal projection

where the main differences lie in how the compression grows by the incident angle. Motivated by the fact that real lenses do not follow the projection model exactly, Kannala and Brandt [23] proposed a polynomial model extending Eq. (9) with higher order terms:

$$r(\theta) = k_1\theta + k_2\theta^3 + k_3\theta^5 + k_4\theta^7 + \ldots \tag{10}$$

The authors found the first five terms to give enough degrees of freedom to approximate different projection curves. Since real lenses may also deviate from precise radial symmetry, they supplement their model with asymmetric terms, not shown in Eq. (10).

Although they obtain sub-pixel accuracy with this generic model, forward and inverse projection can be computationally demanding, depending on how many terms are used in the mapping and the asymmetric distortion. It is, however, possible to precompute distortion terms and store them in a look-up table, reducing computation at the cost of memory. In particular, one could use a look-up table that stores the ray direction (θ, ϕ) corresponding to a pixel (u, v), for inverse projection, and vice versa for forward projection.

We found the equidistant camera model, using only a single term as in Eq. (9), to be satisfactory for our purpose. By ignoring asymmetric distortion effects we simplify both calibration and runtime computation. Nevertheless, our solution is applicable for any camera model. Hence, we denote the general projection of a point $p^c = (x, y, z)^T$, decomposed in the camera frame $\{C\}$, to a point $s = (u, v)^T$ in the image by the projection function $\pi : \mathbb{R}^3 \to \mathbb{R}^2$. In the case of the equidistant model:

$$s = \pi(p^c) := \begin{bmatrix} u_0 + rx/l \\ v_0 - ry/l \end{bmatrix} \tag{11}$$

where f, u_0, v_0 are the equidistant lens model parameters, and

$$r = f \operatorname{atan}\left(\frac{l}{-z}\right) \tag{12}$$

$$l = \sqrt{x^2 + y^2} \tag{13}$$

The principal point (u_0, v_0) and the focal length f are found through camera calibration, whose procedure is described in Section 3.3.

3.2.2 Pose representation

Our choice of pose representation is motivated by its applicability for parameter estimation in the Lucas-Kanade framework. Like many others [14, 49, 17, 16, 7, 43, 34], we use the Lie algebra strategy where absolute poses are stored as homogeneous transformation matrices, but parameter estimation is done in the unconstrained tangent space around the absolute pose [5, Chapter 10.2].

We follow the notation in [15] and denote the rigid-body transformation from the object coordinate frame $\{o\}$ to the camera coordinate frame $\{c\}$ by the 4×4 matrix $H_o^c \in \mathrm{SE}(3)$

$$H_o^c = \begin{bmatrix} R_o^c & T_o^c \\ 0 & 1 \end{bmatrix} \quad \text{with } R_o^c \in \mathrm{SO}(3) \text{ and } T_o^c \in R^3$$

For convenience, we denote the transformation of points in the object frame, p^o, to the camera frame by the composition operator

$$H_o^c \circ p^o = p^c = R_o^c p^o + T_o^c$$

During parameter estimation, we will solve for small updates to the pose, represented as elements of the Lie algebra, that we write as vectors

$$\xi = (\boldsymbol{\omega}, \boldsymbol{v}) \in \mathbb{R}^6$$

It can be shown that a rotating and translating coordinate frame is related to these vectors by the differential equation

$$\dot{H}_o^c(t) = \xi^\times H_o^c(t) \tag{14}$$

where

$$\xi^\times := \begin{bmatrix} \boldsymbol{\omega}^\times & \boldsymbol{v} \\ 0_{1\times 3} & 0 \end{bmatrix} \quad \text{and}$$

$$\boldsymbol{\omega}^\times := \begin{bmatrix} 0 & -\boldsymbol{\omega}_z & \boldsymbol{\omega}_y \\ \boldsymbol{\omega}_z & 0 & -\boldsymbol{\omega}_x \\ -\boldsymbol{\omega}_y & \boldsymbol{\omega}_x & 0 \end{bmatrix}$$

This implies that

$$\boldsymbol{\omega}^\times = \dot{R}_o^c (R_o^c)^T \tag{15}$$

$$\boldsymbol{v} = \dot{T}_o^c - \boldsymbol{\omega}^\times T_o^c \tag{16}$$

where $\boldsymbol{\omega} = \boldsymbol{\omega}_{o/c}^c$ and $\dot{T}_o^c$ are the angular and linear velocity of the object, respectively. The usefulness of this representation is that, if ξ is constant during the integration interval, Eq. (14) can be exactly discretized with the matrix exponential [5, Chapter 10.2]

$$H_o^c(t) = \exp(\xi^\times t) H_o^c(0) \tag{17}$$

whose closed-form expression can be found in [5, Chapter 9.4]. In other words, pose estimation can be done by iteratively solving for vectors ξ, without the need to impose constraints in the optimization, and updating the absolute pose by the matrix exponential, without the need to avoid singularities.

3.3 Calibrating the camera model

Camera manufacturers seldom list a number of important details about their products that are required if one wants to use them for computer vision. Of interest to us are the camera model parameters Eq. (11), known as *intrinsic* parameters [8]. The process of obtaining values for these is known as *calibration*.

As described in Section 3.2.1, the equidistant model has three parameters: The optical center coordinates (u_0, v_0) and the focal length f. In this section, our goal is to determine these parameters such that, when substituted into Eq. (11), the synthesized image of an exact 3D model looks (geometrically) identical to the image of its real-life counterpart. In addition to a way of roughly estimating these parameters by hand, we will also describe an automatic calibration procedure that determines the optimal parameters by minimizing the sum of distance between point correspondences.

It is important to note that our goal in calibrating the camera is not to create high-quality rectified images. Instead, our goal is to strike a balance between the accuracy of synthesized images and the computational cost of forward- and inverse projection, which limits the complexity of the camera model. As noted by Kannala and Brandt [23], a method that satisfies these goal might be very different from one that aims to produce high-quality rectified images.

Before we proceed, we note that there are freely available calibration tools. `OpenCV` is a general library for computer vision, and has functions for calibrating fisheye lenses from point correspondences between an image and a calibration object (like a checkerboard). Their model appears to be a polynomial model, similiar to that of Eq. (10), however — going by their source code on Github and online documentation — it appears that their forward projection first computes the pinhole projection, and then distorts points by the polynomial. This is problematic for wide-angle lenses, because the pinhole projection is numerically unstable for incident angles at or close to 90 degrees (division by zero). Moreover, the installation of `OpenCV` is cumbersome, and we would rather not force anyone to go through such an ordeal for the sake of calibrating a single camera. Alternatively, there are some `MATLAB` toolboxes for calibrating lenses, but `MATLAB` is not free, and it is unreasonable to expect others to have it installed. For these reasons we decided to roll our own tool.

3.3.1 Rough calibration by hand

The principal point often lies around the center of the image, i.e. if the captured image has a resolution of $W \times H$ pixels, then the projection center can be approximated by $(\frac{W}{2}, \frac{H}{2})$. On some cameras, the lens perimeter is clearly visible in the captured images. If the distance from the principal point to a point on the perimeter is known, and the field of view of the camera — commonly found in most datasheets — is available, we can solve $r = f\theta$ for the focal length f, since the other terms are known. That is:

$$f = \frac{\text{Distance to perimeter from center}}{\text{Field of view}/2}$$

For example, if the field of view is 180 degrees, and the distance to the perimeter was measured to be 600 pixels, the focal length in pixel units is $\frac{600}{\pi/2}$.

3.3.2 Automatic calibration

To automatically calibrate the camera parameters we take photographs from different viewpoints of a calibration object with easily discernable points, like the checkerboard in Figure 9. The checkerboard contains squares of known dimensions, whose intersection points in the images are easily located by hand, or automatically using a corner detector [47]. Since checkerboard squares have known dimensions, we can compare these points with predicted points projected from the object frame under the estimated camera pose (extrinsic parameters) and the camera projection (intrinsic parameters). We will jointly solve for the optimal extrinsic and intrinsic parameters, by minimizing the distance between the predicted points and the observed points using unconstrained Gauss-Newton optimization.

Let H_k be the pose of the calibration object relative to the camera for the k'th image and let $p_{i,k}$ be the point on the object corresponding to pixel $s_{i,k}$, in the k'th image. Let $\boldsymbol{x} = (f,\ u_0,\ v_0,\ \xi_1,\ \xi_2 \ldots \xi_M)$ be the concatenation of optimization variables, containing intrinsic parameters, that are fixed for all the images, and extrinsic parameters, that vary across the M images. The extrinsic variables are represented as twists $\xi_k \in \mathbb{R}^6$, as described in Section 3.2.2. Our goal is to iteratively minimize the sum of squared distances between point correspondences taken across all images, described by the cost function

$$E(\boldsymbol{x}) = \frac{1}{2}\sum_{k=1}^{M}\sum_{i=1}^{N_k}\left(\pi(H_k \circ p_{i,k}; f, u_0, v_0) - s_{i,k}\right)^2 \tag{18}$$

Minimization is done using standard Gauss-Newton optimization [37]. We linearize the residuals, which lets us treat the problem as a *linear* least squares problem that can be solved iteratively. Let us first consider the single-image case $M = 1$, and look at a single error term (indices have been omitted for clarity):

$$e = \pi(H \circ p; f, u_0, v_0) - s$$

We want to know how the error changes if we modify any of the parameters. This is done by taking the partial derivative with respect to each optimization variable. Letting $H \circ p = (x, y, z)$, and using the equidistant projection Eq. (11), we get:

$$e = \begin{bmatrix} u_0 + f\theta x/l \\ v_0 - f\theta y/l \end{bmatrix} - s$$

Since s is constant, the derivatives wrt. to the intrinsic parameters are:

$$\frac{\partial e}{\partial f} = \begin{bmatrix} \theta x/l \\ -\theta y/l \end{bmatrix} = \begin{bmatrix} \hat{u} - u_0 \\ \hat{v} - v_0 \end{bmatrix}$$

$$\frac{\partial e}{\partial u_0} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\frac{\partial e}{\partial v_0} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The relationship between a small camera motion and the resulting motion of a projected point in the image can be shown to be linear wrt. the twist vector (see Appendix A.1). The

derivative of the error wrt. to the extrinsic parameters is therefore:

$$\frac{\partial e}{\partial \xi} = J(H \circ p, f, u_0, v_0)$$

where J is the *interaction matrix* (derived in Appendix A.1). Putting these together we get the combined 2×9 Jacobian:

$$\frac{\partial e}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial e}{\partial f} & \frac{\partial e}{\partial u_0} & \frac{\partial e}{\partial v_0} & \frac{\partial e}{\partial \xi} \end{bmatrix}$$

In general, for $M > 1$, we will have M different poses, and the combined Jacobian for the i'th error in the k'th image will be a $2 \times (3 + 6M)$ matrix of the form:

$$J_{i,k} := \frac{\partial e_{i,k}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial e_{i,k}}{\partial f} & \frac{\partial e_{i,k}}{\partial u_0} & \frac{\partial e_{i,k}}{\partial v_0} & 0 & \dots & 0 & \frac{\partial e_{i,k}}{\partial \xi_k} & 0 & \dots 0 \end{bmatrix}$$

In the Gauss-Newton method, the cost function Eq. (18) is iteratively minimized by solving a quadratic approximation at each iteration, based on the current parameter estimates

$$\begin{aligned} E(\boldsymbol{x} + \boldsymbol{\delta}) &= \frac{1}{2} \sum (e_{i,k} + J_{i,k}\boldsymbol{\delta})^T (e_{i,k} + J_{i,k}\boldsymbol{\delta}) \\ &= \frac{1}{2} \sum e_{i,k}^T e_{i,k} + \sum \left(e_{i,k}^T J_{i,k} \right) \boldsymbol{\delta} + \frac{1}{2} \boldsymbol{\delta}^T \left(\sum J_{i,k}^T J_{i,k} \right) \boldsymbol{\delta} \end{aligned}$$

The optimal parameter update $\boldsymbol{\delta}$ is found by differentiating and equating the above to zero

$$\boldsymbol{\delta}^* = - \left(\sum J_{i,k}^T J_{i,k} \right)^{-1} \left(\sum e_{i,k}^T J_{i,k} \right) \tag{19}$$

and the optimization variables are updated by addition, for the intrinsic parameters, and the exponential matrix Eq. (17), for the extrinsic paramters.

In our implementation, we find that we need a good initialization to ensure that the Gauss-Newton algorithm converges to the correct solution. The rough calibration described in the previous section is sufficient to initialize the intrinsic parameters. Among the extrinsic parameters, the distance along the optical axis is most critical, and should at the very least be initialized to a negative number, since the checkerboard is in front of the camera. The remaining pose variables are initialized manually through measurements or via a graphical user interface.

To lessen the need for good initialization, we use a hierarchical optimization method so that early iterations have effectively fewer degrees of freedom, focusing on the least certain parameters, and gradually increasing the degrees of freedom as the error goes down. This is done by adding a regularizer to the Hessian in Eq. (19):

$$\boldsymbol{\delta}^* = - \left(\sum J_{i,k}^T J_{i,k} + R \right)^{-1} \left(\sum e_{i,k}^T J_{i,k} \right) \tag{20}$$

where, for example, $R = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_{3+6M})$. Initially, the weights for the intrinsic parameters are set large, while the weights for the uncertain extrinsic parameters are set to zero. As the error decreases, or going by the iteration count, the weights are decreased for all parameters, gradually allowing all the degrees of freedom to be taken into account.

Once the calibration has converged, the quality of the result can be verified either by the total resulting error Eq. (18), and visualized by projecting the checkerboard points back

into each image, and manually inspecting the errors. The extrinsic parameters can be useful to determine how the camera is mounted on a platform. For example, if the camera is mounted on a drone, it is useful to know how the camera is rotated or translated with respect to the drone's body frame. In particular, if the drone estimates its orientation using an IMU, the orientation of the camera will be the composition of H_c^b, the camera-in-body transformation, and H_b^w, the body-in-world transformation. H_c^b can be found by manually aligning the body frame with the checkerboard (the object frame), and estimating the extrinsic parameters for the camera:

$$H_c^b = (H_o^c)^{-1}$$

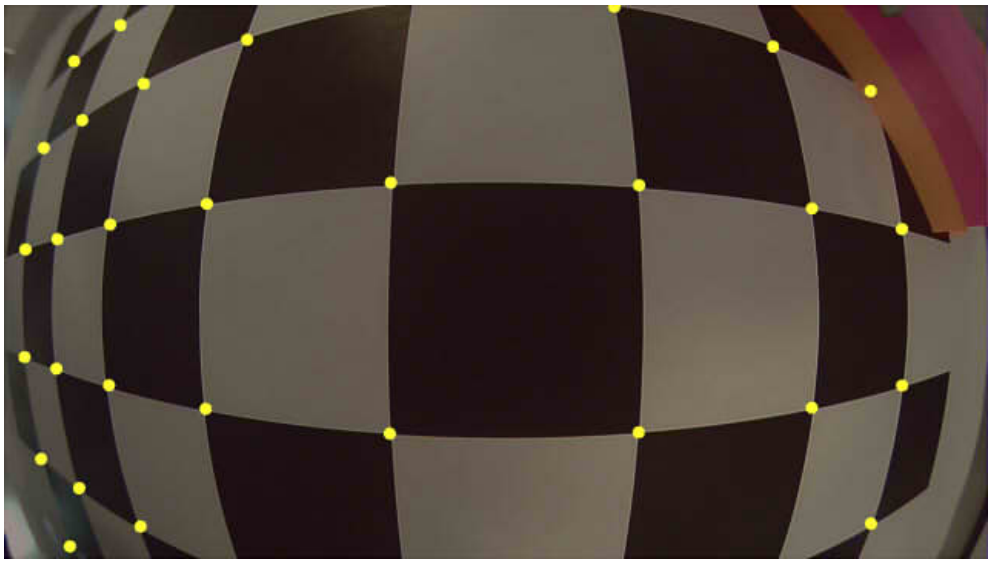


Figure 9: Automatic calibration of an equidistant camera model. A 6×8 grid of points on a printed checkerboard are compared with the projected points from a world-scale model of the checkerboard. The optimal calibration parameters are computed by minimizing the sum of squared distances between observed points (tile intersections) and the points predicted by the camera model (round dots).

3.4 Detection by direct image alignment

This section describes our implementation of the Lucas-Kanade algorithm to estimate the 3D orientation and position of planar objects with a known texture. Our implementation performs the following steps for each input frame:

1. (Offline) Model image is processed into an image pyramid.
2. Input image is read and processed into an image pyramid.
3. Poses are initialized from previous frames or from the color detector.
4. Poses are iteratively updated by hierarchical Gauss-Newton minimization, with regularization in each pyramid level to improve stability and convergence.

We implemented several cost functions based on M-estimators and dense descriptors. These are described in Section 3.4.1 and use the warp function that is described in Section 3.4.2. In Section 3.4.3 we describe how we iteratively update pose estimates by Gauss-Newton optimization on the forward compositional formulation. Finally, Section 3.4.4 details our implementation of rotation invariant Descriptor Fields, which we felt was valuable to include since we could not find any description in the literature.

3.4.1 Objective function

We implemented four objective functions that combine a robust weighting function Eq. (7) with a dense descriptor Eq. (8) under the Euclidean distance, each described by the general iterative reweighted least squares objective

$$E(H) = \sum_i w_i ||r_i||_2^2 \tag{21}$$

where

$$r_i = \phi_{\mathcal{I}}(\mathcal{W}(t_i;\theta))) - \phi_{\mathcal{M}}(t_i) \tag{22}$$

We compare two choices for the weight function w_i

$$w_i = 1 \qquad \text{Quadratic } (L_2)$$

$$w_i = \begin{cases} 1 & \text{if } ||r_i|| \leq k \\ \frac{k}{||r_i||} & \text{otherwise} \end{cases} \qquad \text{Huber (Hu)}$$

and two choices for the feature descriptor

$$\phi_{\mathcal{I}} = \mathcal{I} \qquad \text{Intensity (In)}$$

$$\phi_{\mathcal{I}} = (\mathcal{I}_u^+, \mathcal{I}_u^-, \mathcal{I}_v^+, \mathcal{I}_v^-) \qquad \text{Descriptor field (Df)}$$

where the descriptor field is made rotation invariant, using the procedure described in Section 3.4.4.

3.4.2 Warp function

The warp function transforms surface points on the model to camera coordinates and projects the point into the image using the equidistant camera projection. In particular, we let $H_o^c \in \mathrm{SE}(3)$ be the model coordinate frame decomposed in the camera coordinate frame, and $t_i \in \mathbb{R}^2$ be the texture coordinate in the model with an associated intensity $\mathcal{M}(t_i) \in \mathbb{R}$. The texture coordinates are mapped to 3D points $p_i^o = p_i^o(t_i) \in \mathbb{R}^3$ by a linear surface parametrization, rotated and translated into the camera frame, and projected into the image by. Therefore, the warp function is

$$\mathcal{W}(t_i; H_o^c) = \pi(H_o^c \circ p_i^o) \tag{23}$$

and the cost function Eq. (21) becomes

$$E(H_o^c) = \sum_{t_i \in \mathcal{D}_\mathcal{M}} w_i \|\phi_\mathcal{I}(\pi(H_o^c \circ p_i^o)) - \phi_\mathcal{M}(t_i)\|_2^2 \tag{24}$$

3.4.3 Gauss-Newton minimization

We minimize Eq. (24) by iteratively minimizing

$$E(\xi) = \sum_{t_i \in \mathcal{D}_\mathcal{M}} w_i \|\phi_\mathcal{I}(\pi(\exp(\xi)\hat{H}_o^c \circ p_i^o)) - \phi_\mathcal{M}(t_i)\|_2^2$$

for small updates $\xi \in \mathbb{R}^6$ that are composed with the absolute pose estimate

$$\hat{H}_o^c \leftarrow \exp(\xi)\hat{H}_o^c$$

This is done by linearizing the residuals with respect to ξ and solving the resulting linear least squares problem. Let $s_i = \pi(H_o^c \circ p_i^o)$ be the i'th surface point projected into the image, and let $s_i' = \pi(\exp(\xi)H_o^c \circ p_i^o)$ be the same point after a small pose change. We show in the Appendix these are related by

$$s_i' = s_i + J_i\xi$$

where J_i is the warp Jacobian evaluated at $H_o^c \circ p_i^o$, whose derivation is given in Appendix A.1. The first order Taylor expansion of the image descriptor is therefore

$$\phi_\mathcal{I}(s_i') \approx \phi_\mathcal{I}(s_i) + \nabla\phi_\mathcal{I} J_i \xi \tag{25}$$

leading to the quadratic cost approximation

$$E(\xi) = \sum_i \|\phi_\mathcal{I}(s_i) + \nabla\phi_\mathcal{I}(s_i) J_i \xi - \phi_\mathcal{M}(t_i)\|_2^2 \tag{26}$$

and the optimal parameter update

$$\hat{\xi} = -\left(\sum_i J_i^T \nabla\phi_\mathcal{I}^T \nabla\phi_\mathcal{I} J_i\right)^{-1} \sum_i \nabla\phi_\mathcal{I} J_i \tag{27}$$

For the intensity descriptor $\phi_{\mathcal{I}} = \mathcal{I}$ the descriptor gradient is the image gradient estimated by central differences

$$\nabla\phi_{\mathcal{I}} = \nabla\mathcal{I} = \begin{bmatrix} \frac{\mathcal{I}(u+1,v)-\mathcal{I}(u-1,v)}{2} & \frac{\mathcal{I}(u,v+1)-\mathcal{I}(u,v-1)}{2} \end{bmatrix} \tag{28}$$

whereas the descriptor field gradient is slightly more involved, since it is a vector descriptor of four components, or equivalently a four channel image, which means that the gradient becomes a 4×2 matrix, unlike the image gradient which is a 1×2 matrix. Moreover, the steps needed to make the descriptor rotation invariant means that the gradient cannot be directly computed by finite differences. We deal with this in the next section.

3.4.4 Rotation invariant descriptor fields

Aside from sum of squared intensities, we also implement a robust dense feature descriptor. We base our implementation on the Gradient Based Descriptor Field (GBDF) of Crivellaro et al. [10]. Motivated by the criticism of Ngo [35] and Alismai [1], who noticed that GBDF is not rotation invariant, we extend the descriptor to be so. Since Ngo [35] did not provide any details of their implementation, and Alismai did not attempt to fix it, we will need to derive this on our own.

The descriptor field $\phi_{\mathcal{I}}(s)$ of an image $\mathcal{I}$ at pixel $s = (u, v)$ is defined in [10] as the vector

$$\phi_{\mathcal{I}}(s) = (\mathcal{I}_u^+, \mathcal{I}_u^-, \mathcal{I}_v^+, \mathcal{I}_v^-)$$

where subscript denotes the derivative with respect the image coordinate and superscript $+, -$ denotes the operator

$$x^+ = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad x^- = \begin{cases} -x & \text{if } x < 0 \\ 0 & \text{otherwise} \end{cases}$$

which preserves the positive and negative components. We call this the *unsigned* gradient field. The *signed* gradient field can be reconstructed from the unsigned components by

$$\mathcal{I}_u = \mathcal{I}_u^+ - \mathcal{I}_u^- \quad \text{and} \quad \mathcal{I}_v = \mathcal{I}_v^+ - \mathcal{I}_v^- \tag{29}$$

The residual Eq. (22) is the L_2 norm of the vector difference

$$||r_i||_2^2 = ||\phi_{\mathcal{I}}(s_i) - \phi_{\mathcal{M}}(t_i)||_2^2 \tag{30}$$

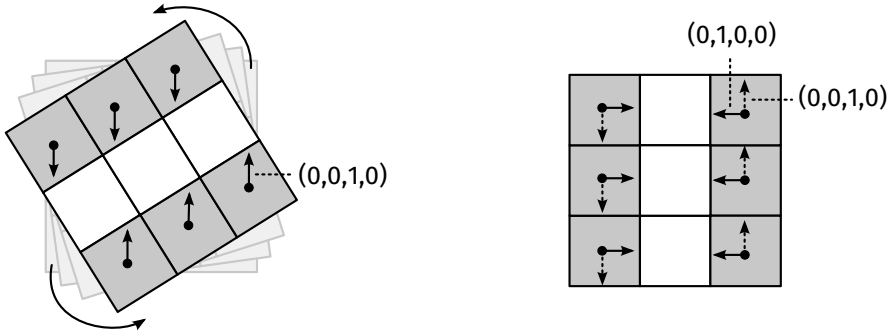
Following the suggestion in [10], we smooth the descriptors by a Gaussian kernel. We found that smoothing the signed gradient field, and then extracting the unsigned components, removed valuable information due to cancellation. Therefore, we extract the unsigned components from the signed field, and then smooth these individually.

The problem occurs when the model is rotated relative to the image, and the texture containing the descriptor values is sampled without taking this into account. To illustrate, consider the situation shown in Figure 10, where the model and the input are both identical 3×3 images, but rotated ninety degrees relative to each other. The descriptor field for both images is computed from the gradients, whose directions are indicated by the black arrows.

(a) Input **(b)** Model

Figure 10: The model and input images are identical 3×3 images, but one is rotated ninety degrees. The gradients are indicated by the black arrows.

Consider now what happens when the model is rotated into the vertical orientation of the input, as shown in Figure 11a. For the intensity difference metric, one would simply compare each intensity value in the model texture with the intensity sampled from the input image at the warped pixel. Descriptor fields can be thought of as an extension of grayscale images where the model and input are now four-channel images. A naive implementation would therefore do as before, and compare the vector in the model texture with the bilinearly sampled vector in the input. However, since the texture values are constructed from image gradients they are *directional*, and they too must be rotated along with the warp!



(a) Model being rotated **(b)** Comparison

Figure 11: The model descriptors are unmodified during rotation and erroneously compared with those of the input.

The consequence of *omitting* this step is shown in Figure 11a, where the descriptor vectors in the model texture are unmodified during the rotation. Consider the gradient pointing up in the model, emphasized in Figure 11a, where $\mathcal{M}_u = 0$ and $\mathcal{M}_v = 1$, giving the descriptor vector $\phi_\mathcal{M} = (\mathcal{M}_u^+, \mathcal{M}_u^-, \mathcal{M}_v^+, \mathcal{M}_v^-) = (0, 0, 1, 0)$. Once rotated, this vector is compared with the upper-right pixel in the input emphasized in Figure 11b. The input pixel has a gradient pointing left, that is $\mathcal{I}_u = -1$ and $\mathcal{I}_v = 0$, hence $\phi_\mathcal{I} = (\mathcal{I}_u^+, \mathcal{I}_u^-, \mathcal{I}_v^+, \mathcal{I}_v^-) = (0, 1, 0, 0)$. Since the two are not component-wise identical, the residual Eq. (30) is erroneously non-zero:

$$||r_i||_2^2 = ||\phi_\mathcal{I} - \phi_\mathcal{M}||_2^2 = ||0, 1, -1, 0||_2^2 = 2$$

even though the textures are correctly aligned.

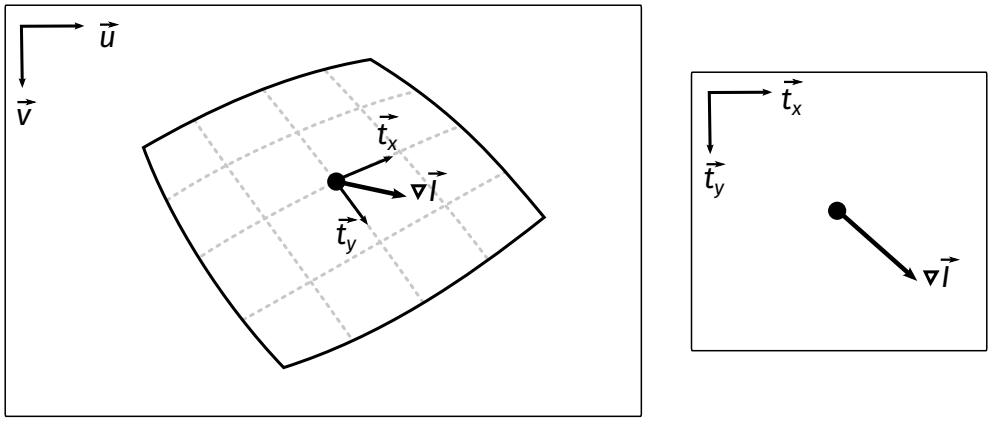


Figure 12: The input gradient $\nabla \mathcal{I}$ is computed by finite differences in the input coordinate frame, therefore, its coordinates are decomposed by the u, v basis vectors. Since these basis vectors are different than those of the warped model texture, t_x and t_y, gradients in the input and gradients in the warped model are computed in different frames and can therefore not be directly compared. Shown on the left is an input gradient, obtained from finite differences, plotted as a vector. This gradient must be transformed to the model texture space, shown on the right, before comparing it with model gradients.

The problem gets slightly more hairy when the model undergoes a nonlinear warp, as opposed to a simple rotation. Our solution is to notice that we can transform gradients from either coordinate frame into the other, and applying the chain rule to express the input image gradients with respect to the model texture coordinates. That is,

$$\frac{\partial \mathcal{I}}{\partial t_x} = \frac{\partial \mathcal{I}}{\partial u}\frac{\partial u}{\partial t_x} + \frac{\partial \mathcal{I}}{\partial v}\frac{\partial v}{\partial t_x} \tag{31}$$

$$\frac{\partial \mathcal{I}}{\partial t_y} = \frac{\partial \mathcal{I}}{\partial u}\frac{\partial u}{\partial t_y} + \frac{\partial \mathcal{I}}{\partial v}\frac{\partial v}{\partial t_y} \tag{32}$$

written compactly as

$$\begin{bmatrix} \frac{\partial \mathcal{I}}{\partial t_x} & \frac{\partial \mathcal{I}}{\partial t_y} \end{bmatrix} = \nabla \mathcal{I} \frac{\partial s}{\partial t}$$

where $\nabla \mathcal{I}$ is computed as in Eq. (28). The right-hand term describes how model texture coordinates are mapped to input texture coordinates. For the warp in Eq. (23) we have

$$\frac{\partial s}{\partial t} = \frac{\partial \pi}{\partial p^c}\frac{\partial p^c}{\partial p^o}\frac{\partial p^o}{\partial t}$$

where

$$\frac{\partial \pi}{\partial p^c} = J(p^c)$$

$$\frac{\partial p^c}{\partial p^o} = \frac{\partial}{\partial p^o}(R_o^c p^o + T_o^c) = R_o^c$$

$$\frac{\partial p^o}{\partial t} : \text{ is obtained from the surface parametrization.}$$

Note that Figure 12 can be a little misleading, since the columns of $\frac{\partial s}{\partial t}$ do not, in general, form an orthogonal basis. Now, the unsigned descriptor can be recomputed in the correct

frame from $\partial \mathcal{I}/\partial t_x$ and $\partial \mathcal{I}/\partial t_y$, the result we denote by (f_1, f_2, f_3, f_4), where

$$f_1 = \left[\frac{\partial \mathcal{I}}{\partial t_x}\right]^+ \quad f_2 = \left[\frac{\partial \mathcal{I}}{\partial t_x}\right]^- \quad f_3 = \left[\frac{\partial \mathcal{I}}{\partial t_y}\right]^+ \quad f_4 = \left[\frac{\partial \mathcal{I}}{\partial t_y}\right]^-$$

We now need to compute the gradient of the descriptor vector that forms the linearized residual Eq. (25). That is, we need to compute the input-space gradient of the unsigned gradient components transformed to model-space:

$$\nabla f_1 = \frac{\partial f_1}{\partial s} \qquad \nabla f_2 = \frac{\partial f_2}{\partial s} \qquad \nabla f_3 = \frac{\partial f_3}{\partial s} \qquad \nabla f_4 = \frac{\partial f_4}{\partial s}$$

One option, that we see, is to rasterize $f_{1..4}$ to seperate textures, and then compute their input-space gradients by finite differences. However, these textures must be rasterized in each iteration *and* for each target being refined, since the transformation depends on the pose parameters. That seemed expensive, so we chose to do the following. Consider the first component of the descriptor, f_1. Its gradient is

$$\frac{\partial f_1}{\partial s} = \frac{\partial}{\partial s}\left[\frac{\partial \mathcal{I}}{\partial t_x}\right]^+ = \begin{cases} \frac{\partial}{\partial s}\frac{\partial \mathcal{I}}{\partial t_x} & \text{if } \frac{\partial \mathcal{I}}{\partial t_x} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

where

$$\frac{\partial}{\partial s}\frac{\partial \mathcal{I}}{\partial t_x} = \frac{\partial}{\partial s}\left(\frac{\partial \mathcal{I}}{\partial u}\frac{\partial u}{\partial t_x} + \frac{\partial \mathcal{I}}{\partial v}\frac{\partial v}{\partial t_x}\right)$$

Intuitively, this derivative is affected by two factors. One, as the projected input-space point s moves, the input-to-model transformation will change, since the transformation is nonlinear (in our equidistant projection). The second factor is the input-space gradient of the input-space descriptor, which is computed in the preprocessing stage. In our case, the transformation basis is relatively constant, so we approximate its derivative by zero. This implies that

$$\frac{\partial}{\partial s}\frac{\partial \mathcal{I}}{\partial t_x} \approx \left(\frac{\partial}{\partial s}\frac{\partial \mathcal{I}}{\partial u}\right)\frac{\partial u}{\partial t_x} + \left(\frac{\partial}{\partial s}\frac{\partial \mathcal{I}}{\partial v}\right)\frac{\partial v}{\partial t_x} \tag{33}$$

and

$$\frac{\partial}{\partial s}\frac{\partial \mathcal{I}}{\partial t_y} \approx \left(\frac{\partial}{\partial s}\frac{\partial \mathcal{I}}{\partial u}\right)\frac{\partial u}{\partial t_y} + \left(\frac{\partial}{\partial s}\frac{\partial \mathcal{I}}{\partial v}\right)\frac{\partial v}{\partial t_y} \tag{34}$$

The desired gradients can now be readily computed as

$$\frac{\partial f_1}{\partial s} = \begin{cases} \frac{\partial}{\partial s}\frac{\partial \mathcal{I}}{\partial t_x} & \text{if } \frac{\partial \mathcal{I}}{\partial t_x} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{\partial f_2}{\partial s} = \begin{cases} -\frac{\partial}{\partial s}\frac{\partial \mathcal{I}}{\partial t_x} & \text{if } \frac{\partial \mathcal{I}}{\partial t_x} \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{\partial f_3}{\partial s} = \begin{cases} \frac{\partial}{\partial s}\frac{\partial \mathcal{I}}{\partial t_x} & \text{if } \frac{\partial \mathcal{I}}{\partial t_y} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{\partial f_4}{\partial s} = \begin{cases} -\frac{\partial}{\partial s}\frac{\partial \mathcal{I}}{\partial t_x} & \text{if } \frac{\partial \mathcal{I}}{\partial t_y} \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

We saw the need to smooth the unsigned gradient field, instead of smoothing the signed gradient field and extracting the unsigned components. This introduces some additional hoops to jump through. When transforming the input descriptor to the model frame, we first need to reconstruct the signed gradient using Eq. (29), rotate that by Eq. (31), and then extract the unsigned components.

We also do this when computing the input-space gradient of the input-space gradient (the terms inside the parentheses in Eq. (33) and Eq. (34)). That is

$$\frac{\partial}{\partial s}\frac{\partial \mathcal{I}}{\partial u} = \frac{\partial}{\partial s}\left(\mathcal{I}_u^+ - \mathcal{I}_u^-\right)$$
$$\frac{\partial}{\partial s}\frac{\partial \mathcal{I}}{\partial v} = \frac{\partial}{\partial s}\left(\mathcal{I}_v^+ - \mathcal{I}_v^-\right)$$

The terms on the right-hand side are available in textures, so the input-space derivative can be obtained by finite differences, and we get

$$\frac{\partial}{\partial s}\frac{\partial \mathcal{I}}{\partial u} = \nabla \mathcal{I}_u^+ - \nabla \mathcal{I}_u^-$$
$$\frac{\partial}{\partial s}\frac{\partial \mathcal{I}}{\partial v} = \nabla \mathcal{I}_v^+ - \nabla \mathcal{I}_v^-$$

3.4.5 Regularization in the camera frame

We use a hierarchical motion model similiar to [29], but we implement it by imposing a prior [4] on the allowed motions, thus avoiding the complexity in switching between models during runtime, and also allowing us to dynamically weight different motions based on their uncertainty.

Suppose we have information that tells us where we can expect model to be located. For example, maybe we are fairly certain that the model is not rotated very much about the x- or y-axis, nor translated very much along the z-axis. In that case, we would want to disallow parameter updates that tries to move in those directions. This can be done by adding a regularizer that penalizes the associated components of the twist vector. From Eq. (15) the components of $\xi = (\boldsymbol{\omega}, \boldsymbol{v})$ are related to the object motion by

$$\boldsymbol{\omega} = \boldsymbol{\omega}_{o/c}^c$$
$$\boldsymbol{v} = \dot{T}_o^c - \boldsymbol{\omega}^\times T_o^c$$

Note that a regularization matrix with diagonal weights, corresponding to weighting $\boldsymbol{\omega}_x, \boldsymbol{\omega}_y$ and $\boldsymbol{v}_z$, would not penalize the correct motion, since $\boldsymbol{v}$ includes both linear velocity and angular velocity. If we want to penalize translation in the camera frame we need to extract the $\dot{T}_o^c$ components from $\boldsymbol{v}$

$$\dot{T}_o^c = \boldsymbol{v} + \boldsymbol{\omega}^\times T_o^c$$

where T_o^c is obtained from the previous solution and assumed constant during one iteration. Denoting $\eta = (\boldsymbol{\omega}_{o/c}^c, \dot{T}_o^c)$ and using the identity $a^\times b = -b^\times a$ we have

$$\eta = \begin{bmatrix} I_{3\times3} & 0_{3\times3} \\ -(T_o^c)^\times & I_{3\times3} \end{bmatrix} \xi = R_1 \xi$$

Using this relation we can construct a regularization matrix R that weights the individual components of η

$$\eta^T R_2 \eta = \xi^T \underbrace{R_1^T R_2 R_1}_{R} \xi$$

so that the regularized cost function Eq. (26) becomes

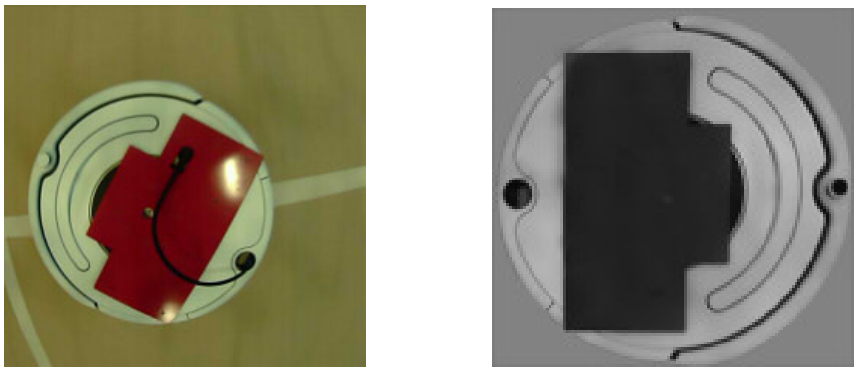
$$E(\xi) = \sum_i \|\phi_I(s_i) + \nabla\phi_I(s_i) J_i \xi - \phi_M(t_i)\|_2^2 + \xi^T R \xi$$

and the optimal parameter update Eq. (27) becomes

$$\hat{\xi} = -\left(\sum_i \left(J_i^T \nabla\phi_I^T \nabla\phi_I J_i \right) + R \right)^{-1} \sum_i \nabla\phi_I J_i$$

3.4.6 Model acquisition

We acquired the model texture from a frame of recorded video where the target was mostly planar and not corrupted by glare. Figure 13 shows the frame and the acquired model texture. This was done by manually aligning the geometric shape of the target to the image, and sampling the image at the resulting surface points using the forward projection equation, thereby also rectifying lens distortion. The resulting image was edited in a painting program to remove the specular highlights, the cable, and the background pattern, since these are not permanent elements.



(a) Source **(b)** Model

Figure 13: The model texture (b) is captured and rectified from recorded video (a)

3.5 Detection by color segmentation

Although the image alignment method can be initialized from previous frames, we still need a way to generate poses for when targets first appear. One option we see is a sliding window approach, where we fix the orientation and scale of the target, and translate it around looking for good matches, at the coarsest level of detail (for efficiency). This could be useful for general objects, and it might not even be that slow (taking into account that the method we are about to describe *does* also has a cost), but we did not investigate this option in detail. Instead, we generate initial pose estimates by segmenting the colored top plates of the targets, and compute their 3D position and orientation by inverse projection, using the height and orientation of the camera, measured by on-board sensors.

Our approach first creates a binary segmentation, by comparing each pixel against a constant threshold (one for red and one for green). Then, we cluster the binary pixels together by computing the set of connected components. The main steps involved in this is illustrated in Figure 14. Finally, we recover the 3D position and orientation of the plates by intersecting the ray toward their pixel centers against the world plane.

3.5.1 Connected components

First, in an attempt to be robust against global and local lighting effects, we transform colors to the normalized rgb color space:

$$(r, g, b) = \frac{(R, G, B)}{R + G + B}$$

We create the binary segmentation by iterating over each pixel, computing its normalized rgb color, and comparing the relative magnitude of red against green and blue (to test for red), and the relative magnitude of green against red and blue (to test for green). In pseudocode the per-pixel comparison looks like this:

```
IsRed = r > RedGreen * g && r > RedBlue * b;
IsGreen = g > GreenRed * r && g > GreenBlue * b;
```

where `r,g,b` are the normalized rgb components of the given pixel, and `RedGreen`, `RedBlue`, `GreenRed` and `GreenBlue` are constant thresholds indicating the required relative magnitude. If a pixel satisfies either comparison its binary pixel is set to 1, indicating *active* pixels that are to be connected, otherwise it is left as 0.

We then compute the set of connected components in the binary image. For each active pixel, that is not already part of a component, we grow a new connected component by a breadth-first expansion on its active neighbors, with 8-neighborhood connectivity. Once there are no more active neighbors, we continue with the next pixel not part of any component.

3.5.2 Recovering the object pose

Mapping pixels back to their original world coordinates involves inverting the camera projection. In general, we can only determine these coordinates up to an unknown scale,

but if we assume that the pixels belong to an object located at a fixed height relative to the ground, and we know the height and orientation of the camera, it is possible to recover the original coordinates. This is done by intersecting the ray, originating at the camera aperture and passing through the pixel on the viewing sphere, with the ground plane. Let d^c be the ray passing through a pixel (u, v), decomposed in the camera frame. Then, as shown in Figure 8, the ray can be written in terms of the viewing sphere coordinates (ϕ, θ):

$$d^c = \begin{bmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ -\cos\theta \end{bmatrix} \tag{35}$$

For the equidistant model in particular, the spherical coordinates are:

$$\cos\phi = \frac{u - u_0}{r} \tag{36}$$

$$\sin\phi = \frac{v_0 - v}{r} \tag{37}$$

$$\theta = \frac{r}{f} \tag{38}$$

where $r = \sqrt{(u-u_0)^2 + (v_0 - v)^2}$.

We compute the center of each component by taking the mean of the pixel coordinates of its members. Then, using the height and orientation of the camera, we compute the ray passing through the center and find its intersection with the target's top plate. In particular, let h and h_t be the height of the camera and the plate above the ground, respectively, and let $R_c^g = R_y(e_y^c)R_x(e_x^c)$ be the orientation of the camera $\{c\}$ relative to the ground plane $\{g\}$, expressed in Euler angles. Then $d^g = R_c^g d^c$ is the ray expressed in ground plane coordinates and its intersection point with the target's top plate is

$$p^g = p_0^g + td^g = (x, y, h_t)$$

where $p_0^g = (0, 0, h)$ is the position of the camera. After solving for t we find that

$$p^g = (0, 0, h) + \frac{h_t - h}{d_z^g} d^g$$

Since we need the object position relative to the camera, we rotate its relative vector to the camera frame and get the simplified expression

$$T_o^c = R_g^c(p^g - p_0^g) = R_g^c(td^g) = td^c = \frac{h_t - h}{d_z^g} d^c \tag{39}$$

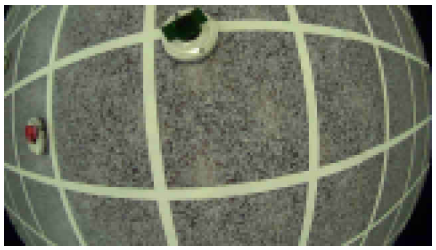
Since the object is (presumably) flat with the floor, its orientation, aside from its heading, relative to the camera is the inverse of that of the camera relative to the floor. We can therefore express its orientation as

$$R_o^c = R_x(e_x^o)R_y(e_y^o)R_z(e_z^o) = (R_c^g)^T R_z(e_z^o) \tag{40}$$

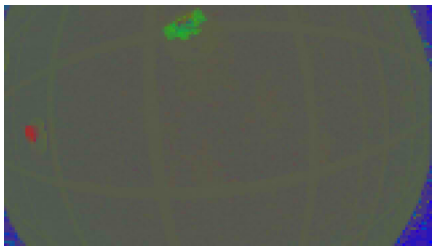
where e_z^o is its heading angle (relative to the camera). The camera orientation is obtained from a state estimation filter in the on-board flight controller, and the height is obtained

by combining this orientation with the raw LIDAR range. We only measure the pitch and roll angles of the camera, e_x^c and e_y^c, and not the yaw angle, but that does not affect d_z^g and therefore does not affect T_o^c either.

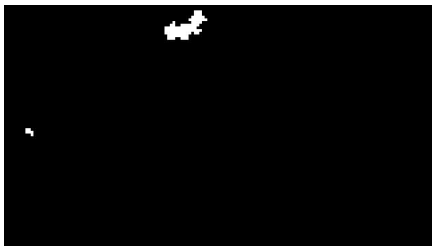
We merge neighboring components if the distance between their projected ground plane coordinates is less than the diameter of the target robot. This reduces superfluous detections. We also compute the aspect ratio of each connected component and require that it be sufficiently square, thereby ignoring components that are thin and long in one direction (in particular, the red and green arena edges). We also compute the filled area of each component, by dividing the number of pixels in it with the area of its bounding box, and require it to be above a threshold. This avoids thin structures that still have a square aspect ratio, such as line crossings. We also ignore components that have very few pixels.



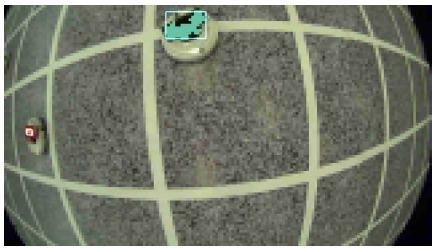
(a) Input image



(b) Normalized colors



(c) Binary segmentation



(d) Connected components

Figure 14: The input image is downscaled four times (a), and segmented into a binary image (c) by a uniform threshold on normalized rgb colors (b). Finally, the active pixels in the binary image are grouped into connected components (d).

4 Evaluation

This chapter evaluates how our solution fares on real life data. For the image alignment detector we look at its ability to report whether a detection is true or false, as this is an important attribute in practice, and study how much error there can be in the initial pose estimate. For the color segmentation detector we look at its ability to segment the top plates across different lighting conditions with the same runtime parameters.

4.1 Datasets

Figure 15a shows selected frames from videos recorded at the competition venue, our full scale tests in a gym and our small scale tests in our lab. We use the videos and selected frames to evaluate the image alignment and the color detector.

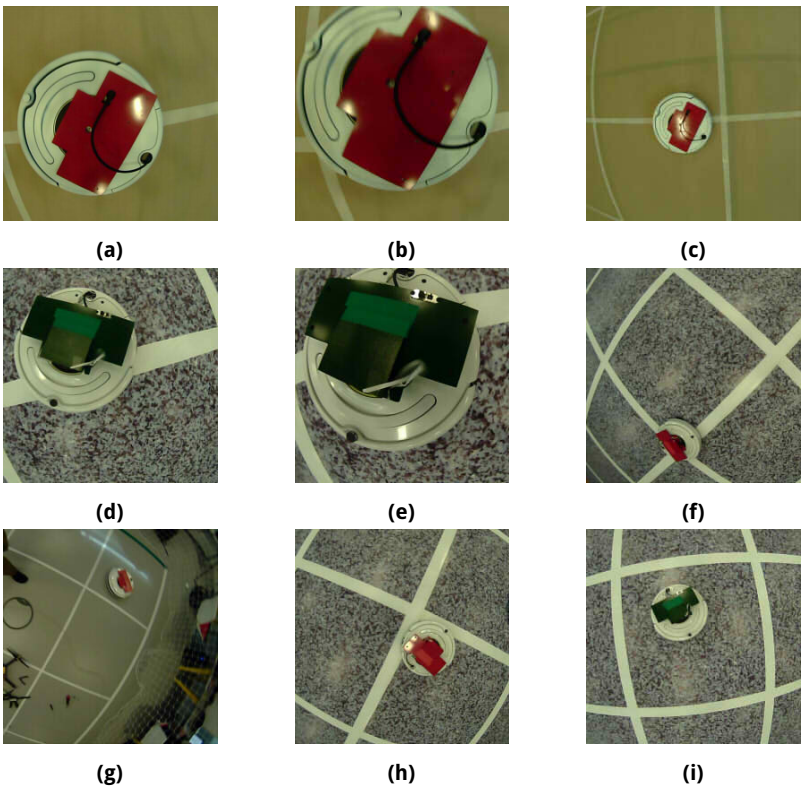


Figure 15: Selected frames from recorded video containing scale variations, multiple structures (when the target is on top of the white grid lines), specular highlights, shadows (when the quadrotor is close), model variation (some of them are missing the switch on their plate), non-rigid parts (the string flapping in the breeze) and motion blur.

4.2 Detection by image alignment

We evaluate the following aspects of our image alignment implementation:

- The necessary sampling interval for yaw initialization
- The ability to report incorrect alignments and false positives
- The maximum error tolerated in the initial estimate

and compare the four cost functions described in Section 3.4.1 on these aspects.

4.2.1 Basin of convergence

The color detector gives a coarse estimate of the position T_o^c and orientation R_o^c of the target, excluding its heading, by combining the estimated pixel center with the orientation and height of the camera, obtained from IMU and LIDAR measurements. The noise in these sensors, and the inaccuracy of the color segmentation, contribute to noise in the coarse position and orientation. How much noise can there be before the image alignment fails to converge to the correct pose, or, how wide is the *basin of convergence*? We study this by sampling a distribution of poses centered around the true pose and, at increasing levels of noise, counting the number of times the algorithm converges to the correct solution.

First, we ballpark the noise in the camera orientation and height based on data logs from recorded flights. The target pitch and roll, e_x and e_y, are directly obtained from those of the camera by Eq. (40), and therefore have the same noise. The target translation T_o^c is obtained from the LIDAR range, the camera orientation, and the pixel center, by Eq. (39). Since the camera is always mostly pointing straight down, the noise in T_z is approximately the same magnitude as the LIDAR range noise, whereas T_x and T_y mostly depend on the estimated pixel center, which we reckon will be anywhere within the top plate.

Based on the above, we obtained our noise baseline. We run the experiment on eight scales of the baseline, given in Table 2. For each scale, σ_i, delta parameters are uniformly sampled from their corresponding range, and added to the true parameters to generate the initial pose estimate. The target heading is not measured by any sensor, and we set its range to a constant ±40 degrees, that we justify experimentally in Section 4.2.2.

σ	Scale	e_x (degrees)	e_y (degrees)	T_x (meters)	T_y (meters)	T_z (meters)
σ_1	0.25	1.25	1.25	0.04	0.04	0.05
σ_2	0.50	2.50	2.50	0.08	0.08	0.10
σ_3	0.75	3.75	3.75	0.11	0.11	0.15
$\boldsymbol{\sigma_4}$	**1.00**	**5.00**	**5.00**	**0.15**	**0.15**	**0.20**
σ_5	1.25	6.25	6.25	0.19	0.19	0.25
σ_6	1.50	7.50	7.50	0.22	0.22	0.30
σ_7	1.75	8.75	8.75	0.26	0.26	0.35
σ_8	2.00	10.0	10.0	0.30	0.30	0.40

Table 2: Scales of the coarse pose estimate noise baseline

We manually calibrate each image for the correct pose and count the number of times the alignment converges to the correct pose. We run 10 iterations at the coarsest level, 10 at the medium level and 10 at the most detailed level, and sample 20 poses per scale.

We say that the alignment converged to the correct pose if the Euler angles e_x, e_y, e_z and translation components T_x, T_y, T_z of the estimated pose H are sufficiently close to those of the true pose H^*. We define *sufficiently close* by the distance metric

$$d(H, H^*) = \frac{1}{6}(|\delta e_x| + |\delta e_y| + |\delta e_z| + |\delta T_x| + |\delta T_y| + |\delta T_z|) \tag{41}$$

where the deltas are weighted such that they are ±1 at an appropriately far away point

$$\delta e_x = \frac{e_x \ominus e_x^*}{0.7} \quad \delta e_y = \frac{e_y \ominus e_y^*}{0.7} \quad \delta e_z = \frac{e_z \ominus e_z^*}{1.14}$$
$$\delta T_x = \frac{T_x - T_x^*}{0.7} \quad \delta T_y = \frac{T_y - T_y^*}{0.7} \quad \delta T_z = \frac{T_z - T_z^*}{0.4}$$

and $a \ominus b$ represents the smallest angular difference between a and b. The criterion for correct convergence was determined by dragging slider bars around and seeing how bad the pose looked for a given distance. We set the criterion to $d(H, H^*) \leq 0.1$.

Figure 16 shows our results for the images shown in Figure 15 in row order (a) through (i). The descriptor field objective appears to have a smaller basin of convergence and overall lower likelihood of converging correctly in all images. In either case, there appears to be no clear advantage of using Huber weighting over L_2.

In Figure 15e and Figure 15f, all objectives erroneously pitch forward in order to align the plate, but cause a large pose error in doing so, since the target is far from planar at that viewpoint. In (e) this is because the target is very close, while in (f) the target is far away but very off-center. Since the top plate extends some centimeters above the body, the target only appears sufficiently planar when sufficiently centered and far away. We think a 3D model would mitigate this issue.

In Figure 15g, all objectives were unstable and blew up at the lowest level of the image pyramid. This was because the target appeared too small, and lost too much detail at the coarsest level. The descriptor field objectives demonstrated greater signs of instability, such as oscillation, than the intensity objectives, due to the descriptor being more sparse. We found that instabilities could be mitigated by reducing the amount of smoothing of the input image and skipping the coarsest pyramid levels, but doing so causes the alignment to converge very slowly, or not at all, in close-ups such as Figure 15e. This hints at the necessity of a dynamic level of detail scheme, where the level of detail is selected based on the size of image region being aligned against.

In Figure 15h all objectives failed to align correctly due to the misleading background pattern and the specular highlight, which caused the objectives to erroneously tilt the pose to align the top plate against the non-specular portion. The descriptor field objective failed to reject this error because it is mostly zero in smooth areas, and non-zero in contrasting areas. In this case, the glare and the background removed a sufficiently large area containing contrast (between the body and the background, and the plate and the body) to cause it to align incorrectly.

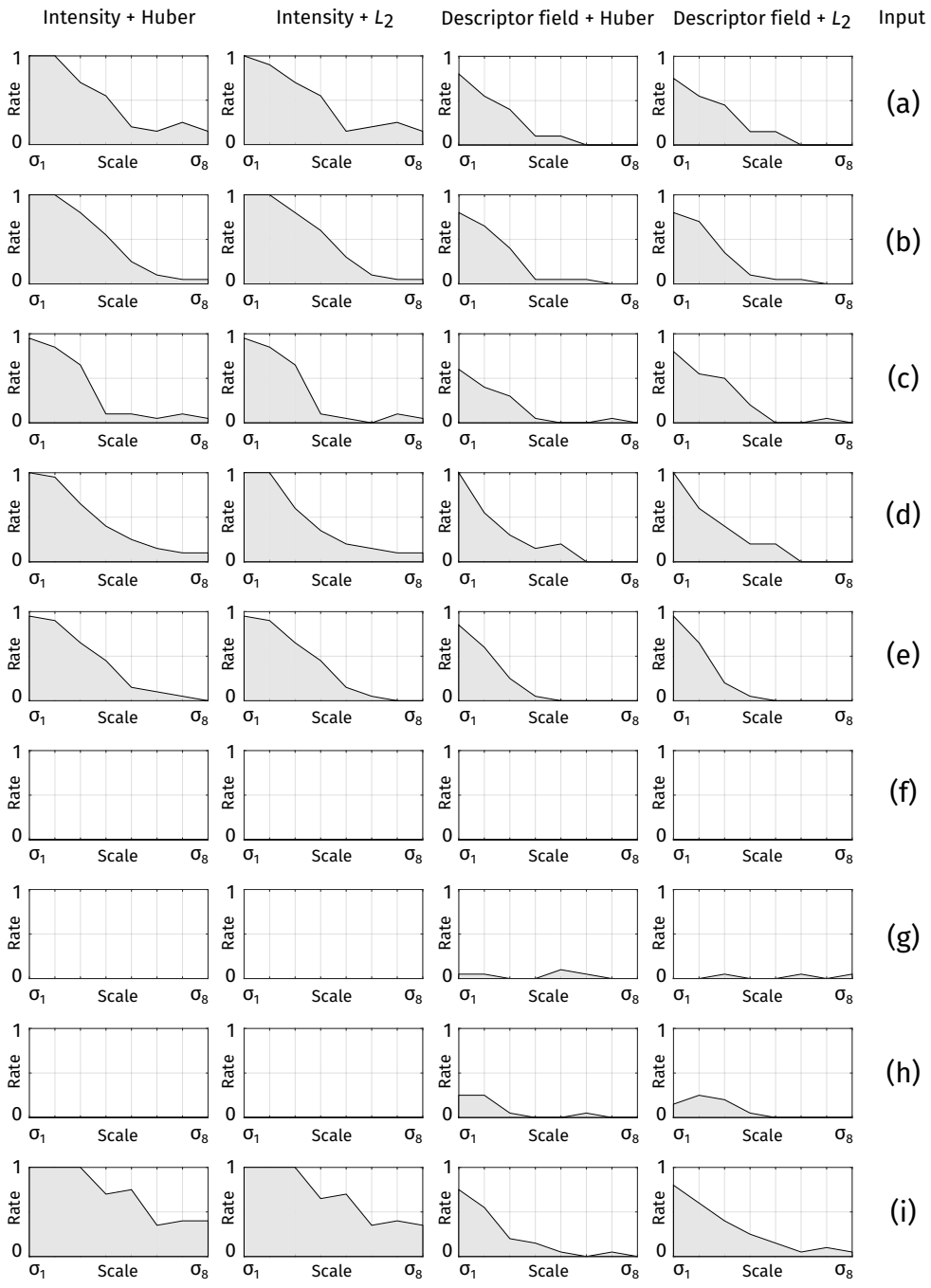


Figure 16: Convergence rate (y-axis) for the four cost functions on each of the 9 images, for increasing scalar multiples of the noise baseline (x-axis). The rate is computed as the number of correct estimates divided by the total number of samples for that scale. Blank graphs indicate that the algorithm failed to converge to the correct pose regardless of initialization.

4.2.2 Minimum sampling for yaw initialization

The color segmentation does not measure the yaw angle of the target, and the convergence basin does not cover all 360 degrees of possible angles. Therefore, we try multiple initial yaw angles and choose the one that converged with the lowest error, but how many angles do we need to try? We study this question by discretizing the range of yaw angles and, for each angle, estimating the likelihood that the alignment converges to the correct pose.

To illustrate, suppose we want to be atleast 60 percent certain that we obtain the true pose from a detection in a single frame. Then we hope to see that (a) the convergence likelihood at the correct yaw angle is above 60 percent and (b) that there exists an angle further away that is also above 60 percent. We could then choose the yaw sampling resolution based on the difference between these angles.

Figure 17 shows our results for a single test image, where the target is slightly corrupted by glare, at a height of 1 meter where the planarity assumption holds well. We restricted our sampling to 180 degrees, since the convergence rate was depressingly low beyond 90 degrees offset from the true yaw. From the figure, we see that the Huber weighting decreases the basin of convergence for both descriptors, and also that the intensity metric has a wider basin than descriptor fields. It appears that sampling 40 degrees at a time is sufficient to sample atleast one angle within ±40 degrees, for which the convergence rate is above 60 percent for all cost functions.

Note that we have 60 frames per second of video, and that once the correct yaw angle has been found, the target can be tracked without the need for reinitialization by reusing the previous frame's pose. This means that we do not need 100 percent certainty of obtaining the correct yaw angle from a single frame, but can instead amortize the cost over several frames, if the delay is acceptable. For example, if the sampling guarantees a 60 percent likelihood per frame of correctly locating the target, and each frame is independently sampled, then five frames is sufficient to be 98 percent certain.

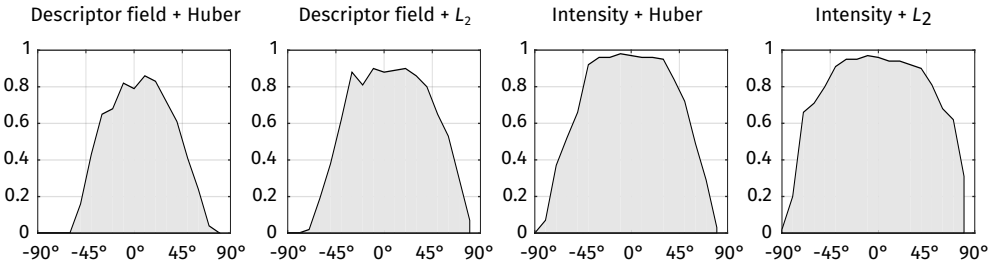


Figure 17: Convergence rate for fixed yaw angles and the remaining parameters sampled from the baseline range in Table 2. The x-axis is the initial yaw in degrees offset from the true yaw, and the y-axis is the number of samples that converged to the correct solution divided by the total number of samples. We used 100 samples per angle.

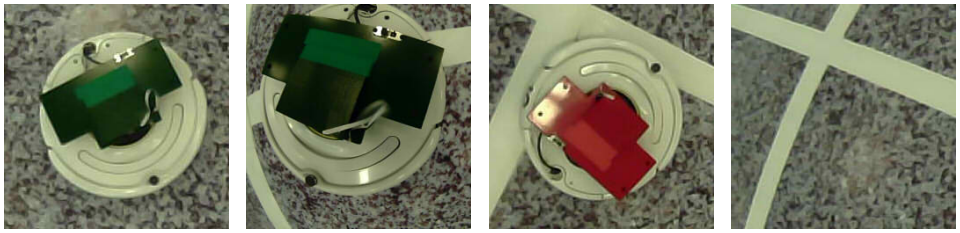
4.2.3 Detecting alignment failure

A critical aspect of the algorithm is its ability to indicate whether a detection is true or false. We investigate whether the value of the cost function, the alignment error, is an appropriate discriminator for distinguishing false detections from true detections. In particular, we would like to answer the following: Let H^* be the true pose, that may not exist if there is no robot present, and let H be a pose estimate with the corresponding alignment error $E(H)$ after some iterations. Does there exist a threshold E_0, such that

$$E(H) < E_0 \Rightarrow H^* \text{ exists and } H \text{ is sufficiently close to } H^*$$
$$E(H) \geq E_0 \Rightarrow H^* \text{ does not exist or } H \text{ is far away from } H^*$$

In other words, can we compare the alignment error with a constant threshold to determine whether or not there is a robot present and, if there is, that the estimated pose is sufficiently close to the true pose? The existance of such a threshold requires that the global minimum of the cost function actually corresponds to the true pose, and that its value at this minimum is sufficiently distinct from incorrect, local minima.

We study this by inspecting the landscape of the cost function in terms of the above properties. Since we have six variables, we flatten the cost function to one dimension, where the y-axis is the alignment error, and the x-axis is the distance from the true parameters, computed by Eq. (41). We randomly sample initial pose parameters from ranges centered at the true pose parameters, and iterate until the estimates converge or reach the maximum number of iterations[4]. The sampling ranges are similiar to those of the noise baselines in Table 2, except the heading is sampled from 360 degrees. We study four images, shown in Figure 18. Figure 18a is almost ideal, with very little glare, Figure 18b breaks the planarity assumption, Figure 18c has strong glare and misleading background patterns, and Figure 18d contains nothing of interest. We obtain ground truth poses for each image by manual calibration.



(a) Almost ideal **(b)** Close-up **(c)** Lines in background **(d)** False detection

Figure 18: Test cases for evaluating alignment failure detection.

Figure 19 show our results for Figure 18a. The scatter plot shows the distribution of pose estimates after optimization, plotted at their alignment error against their distance to the true parameters, while the histogram shows the distribution of estimates for particular

[4]The effect of this is similiar to shaking a cereal box vigurously to force the good bits to bubble up, where, in this case, the good bits are the local minima of a dense photometric similarity function.

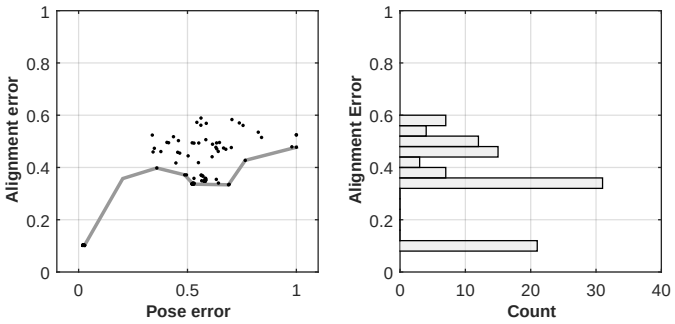
alignment errors. A gray line is drawn between minimum alignment errors across the pose distance axis to indicate the worst-case cost function.

What we want to see is a large gap in alignment error between poses that converged to the correct pose and those that didn't. An example of this is seen in Figure 19a. We see that many (read off the histogram) estimates converged to a pose that is very close to the true pose (since the pose error is small), and they all have an alignment error close to 0.1, while the estimates that failed to converge correctly all have an error above 0.3. This indicates that a threshold around 0.2 could distinguish true and false alignments for this particular image. Similiar behaviour is seen for the L_2 weighted intensity objective Figure 19b, that has a slightly wider gap, and the descriptor field objectives Figure 19c and Figure 19d, although they both have a smaller gap than the intensity objectives.

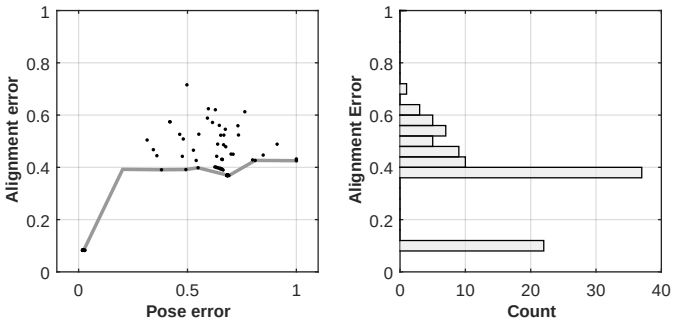
Examples of the contrary is seen in Figure 20, where the target was very close to the camera so that the planar texture model is no longer a good approximation. Compared with Figure 19, all objective functions have larger alignment error for all pose estimates. Moreover, some almost identical pose estimates appear to have vastly different alignment errors. The intensity objective estimates converged to a slightly incorrect pose, while the descriptor field objective estimates suffered from very slow convergence and tended to stay where they were initialized. We also see that the descriptor field objectives have the global minimum far away from the true pose.

Figure 21 shows how the effect of noise on the global minimum location. All objectives converged to a pose that was much farther away than the correct pose compared to that observed in Figure 19 and Figure 20. This is caused by the glare and the background pattern that removes useful contrast against the body. The intensity objectives have an error at 0.2, which is an increase from 0.1 in Figure 19a and Figure 19b, whereas the descriptor field objectives have an error of 0.4, which is an increase from 0.3 in Figure 19c and Figure 19d. The overall increase indicates that all objective functions are sensitive to the fact that the alignment is slightly off, in the sense that the error has increased compared to that in the almost ideal case.

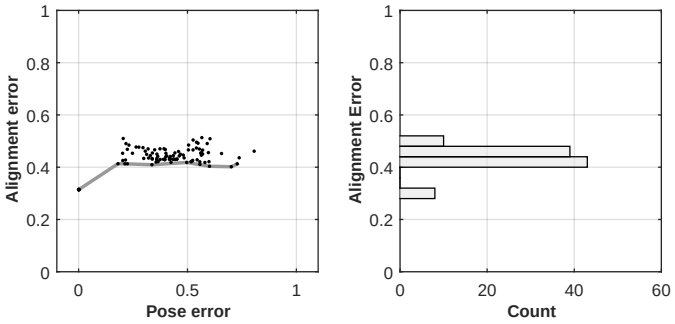
Figure 22 is meant to inspect the correlation (or lack thereof) between the model and an image region not containing a target. Ideally, we do not want empty background regions to appear as targets. We see that Huber weighted intensity has an alignment error around 0.4 across all poses, and the L_2 weighting has a higher error at 0.6. The Huber weighted descriptor field lies around 0.4, and the L_2 weighted lies around 0.6. Since these values are higher than that of the global minimum in Figure 19, this suggests that there does exist a threshold that discriminates true and false detections. However, this threshold would likely label the close-up case of Figure 20 and the contaminated case of Figure 21 as being false detections.



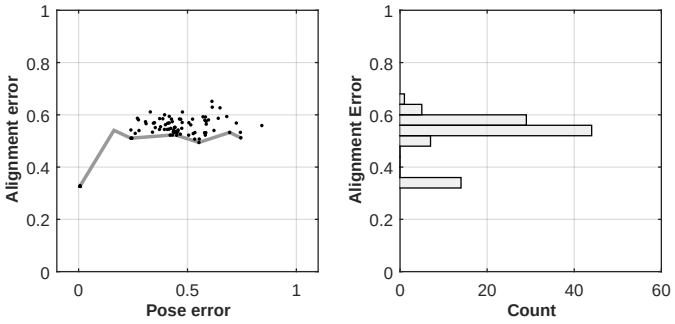
(a) Intensity + Huber



(b) Intensity + L2

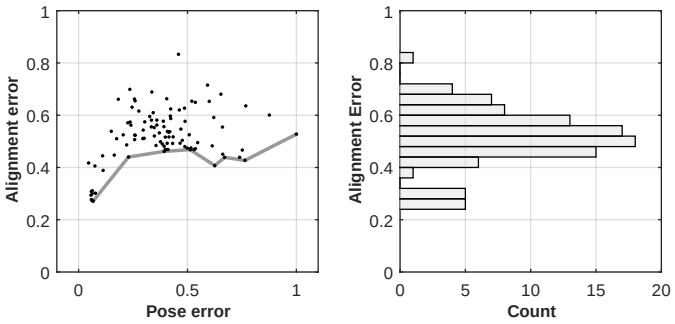


(c) Descriptor Field + Huber

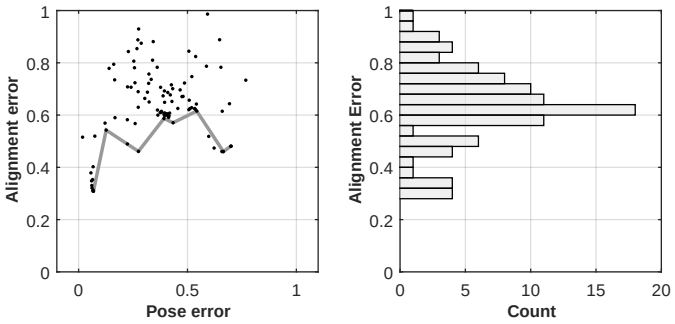


(d) Descriptor Field + L2

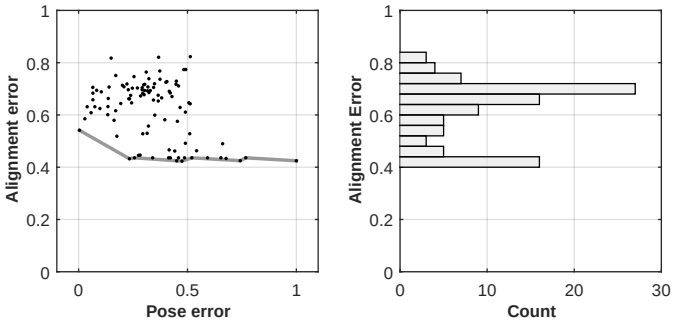
Figure 19: Results for alignment error discrimination for Figure 18a.



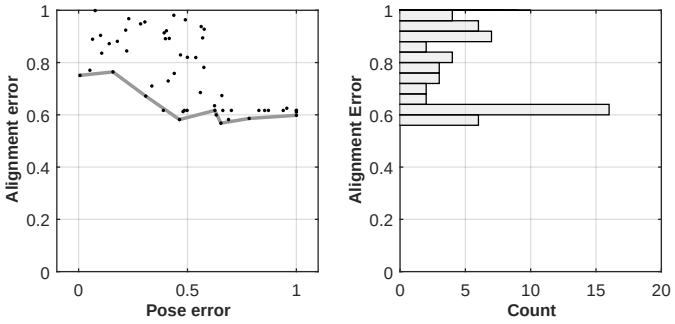
(a) Intensity + Huber



(b) Intensity + L2

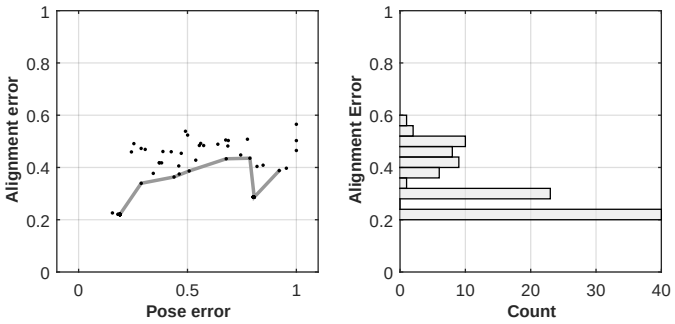


(c) Descriptor Field + Huber

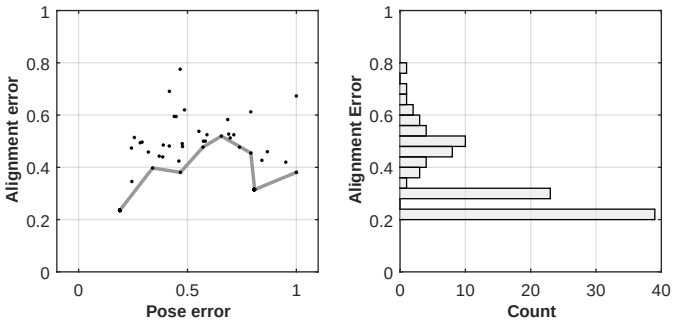


(d) Descriptor Field + L2

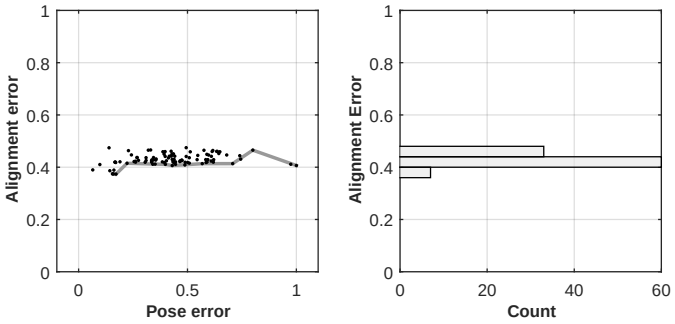
Figure 20: Results for alignment error discrimination for Figure 18b.



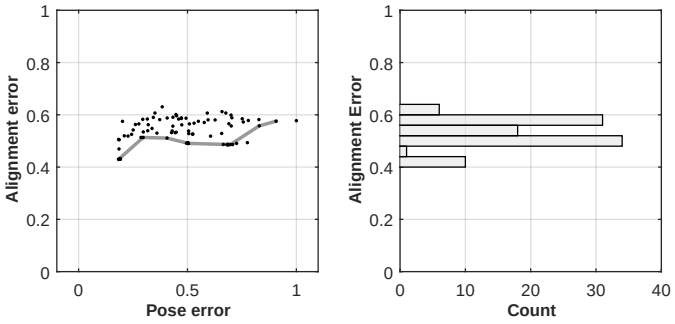
(a) Intensity + Huber



(b) Intensity + L2

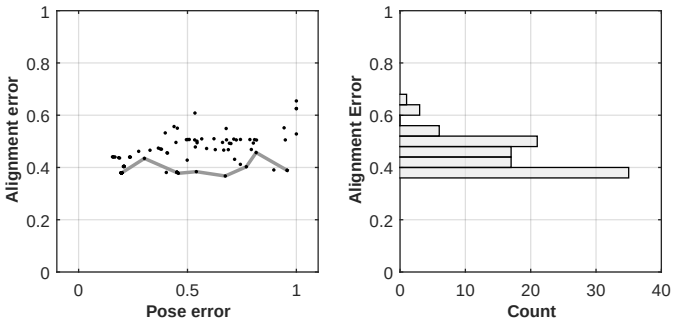


(c) Descriptor Field + Huber

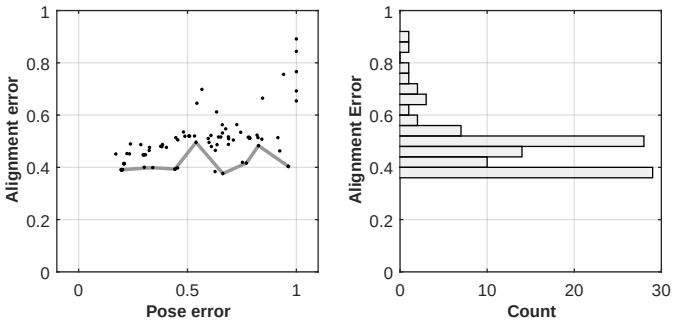


(d) Descriptor Field + L2

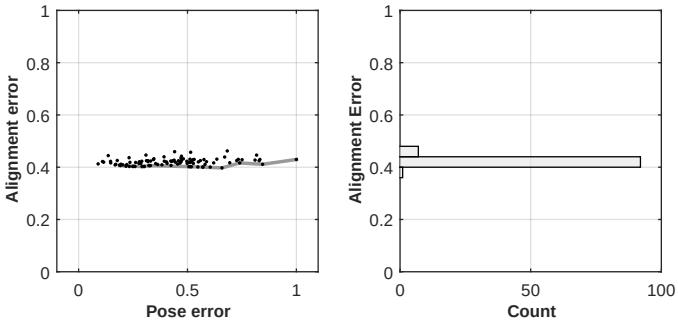
Figure 21: Results for alignment error discrimination for Figure 18c.



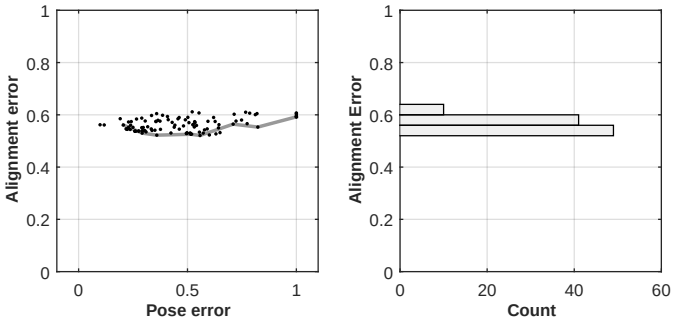
(a) Intensity + Huber



(b) Intensity + L2



(c) Descriptor Field + Huber

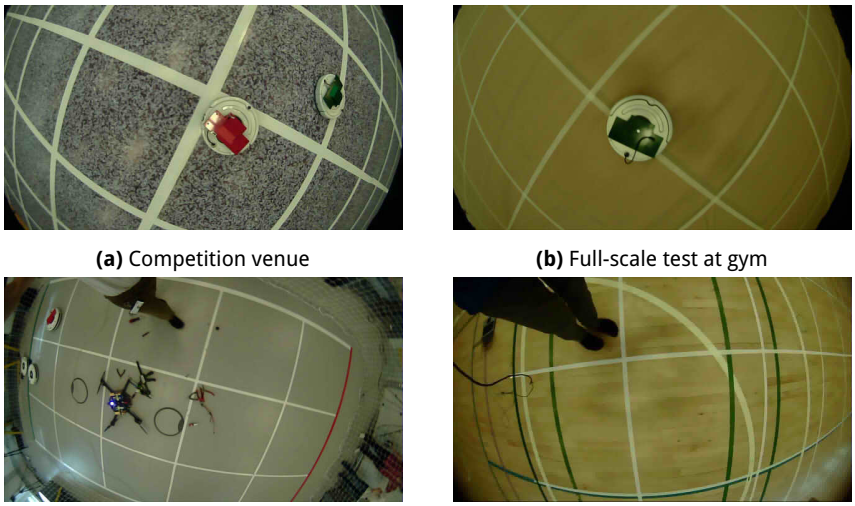


(d) Descriptor Field + L2

Figure 22: Results for alignment error discrimination for Figure 18d.

4.3 Detection by color segmentation

We evaluate the detection rate of our color segmentation algorithm by manually counting the number of true and false positive detections in the four videos shown in Figure 23. We use the same parameters (color and area thresholds) for all videos, instead of tuning them for each particular video, to see if the algorithm is robust against scene changes when the parameters are fixed. The latter is similiar *parameter sensitivity*, that is, how much the color and area thresholds can be adjusted without affecting the performance.



(a) Competition venue

(b) Full-scale test at gym

(c) Small-scale test at lab

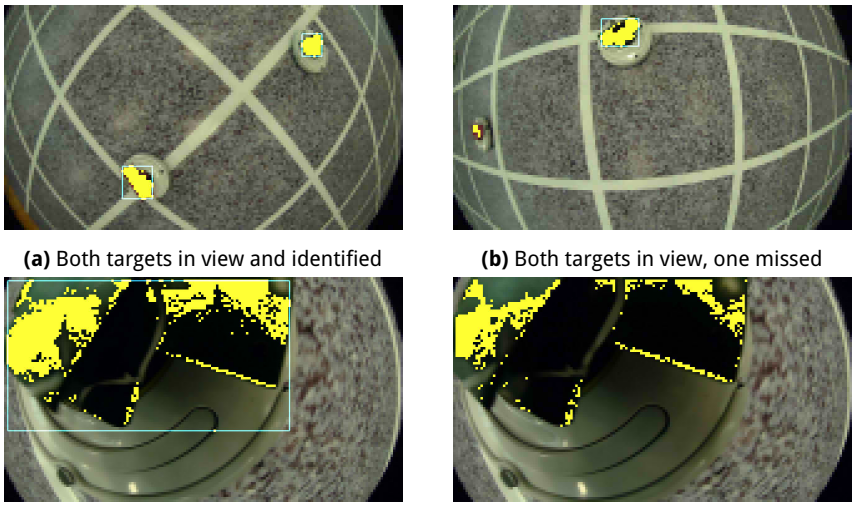
(d) Small-scale test at gym

Figure 23: Videos used to evaluate the color detector.

Figure 23 shows an extract of frames from four videos that we use to evaluate the detection rate. Video (a) contains one red and one green target, and has the quadrotor landing on the targets twice. Video (b) contains two targets, but the lighting is of a higher temperature and the floor texture is brown paper and tape. Video (b) also contains the quadrotor landing on both targets. Video (c) contains a single red target, the red and green arena edges, a red pillow, a red shirt, the author's socks and pants, someone else's socks and pants, a red Tenga screwdriver, a pair of red pliers and, in general, just a lot of stuff. Video (d) contains no targets whatsoever, but has a lot of green marker lines.

4.3.1 Detection rate: Competition venue

Figure 25 shows the detection rate for the video in Figure 23a. The detector had zero false positives, and was able to recognize the top plates as long as a sufficiently large portion was visible and well lit. Figure 24 shows a selection of frames. It correctly identified both targets whenever they were sufficiently centered, and at about 1 meter altitude (a). Targets were missed when the segmented pixels did not occupy enough image area (b). The green plate was correctly identified during a landing (c), until the camera was close enough to occlude most of the plate (d), at which point the filled area was too small compared to the bounding box.



(a) Both targets in view and identified

(b) Both targets in view, one missed

(c) Close-up of green target, identified

(d) Close-up of green target, missed

Figure 24: A selection of frames from Figure 23a. Components that passed the aspect ratio and area tests are drawn with neon bounding boxes. Segmented pixels are otherwise drawn in yellow.

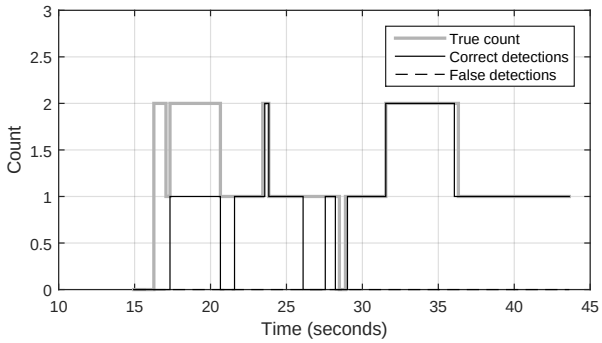
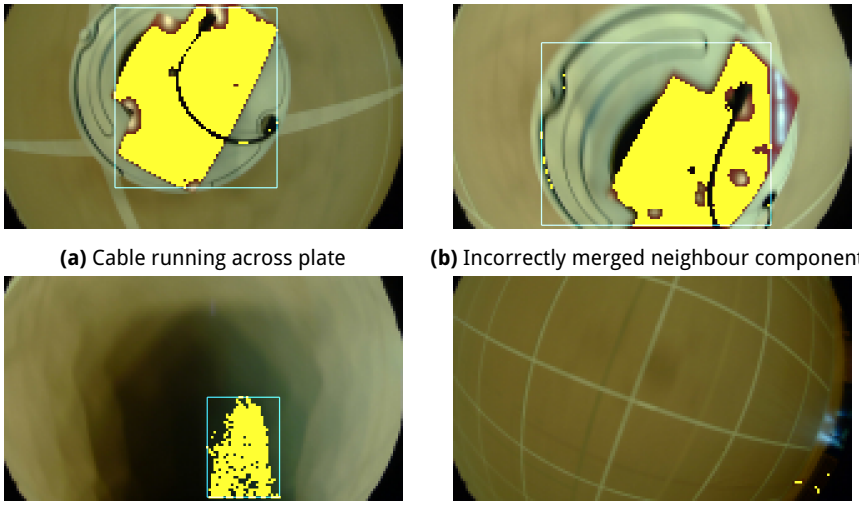


Figure 25: Detection rate for the video in Figure 23a. We manually counted the number of targets actually in the frame (gray), the number of targets that were correctly identified (black), and incorrectly identified (stipled).

4.3.2 Detection rate: Full-scale test at gym

Figure 27 shows the detection rate for the video in Figure 23b and Figure 26 shows a selection of frames. The detector had a single false detection in this video caused by a shadow with a green tint when flying low during a landing (c). It consistently found targets whenever they were visible, well lit and sufficiently centered, and correctly merged plate regions seperated by a black cable (a). The merging sometimes included unrelated pixels that were close enough (b). Some regions outside the arena looked sufficiently red, but were correctly ignored because they were too small (d).



(a) Cable running across plate

(b) Incorrectly merged neighbour components

(c) Misidentified shadow

(d) Correctly unidentified speckles

Figure 26: A selection of frames from Figure 23b. Components that passed the aspect ratio and area tests are drawn with neon bounding boxes. Segmented pixels are otherwise drawn in yellow.

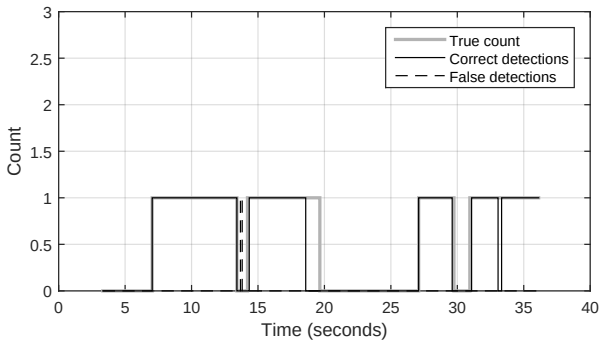
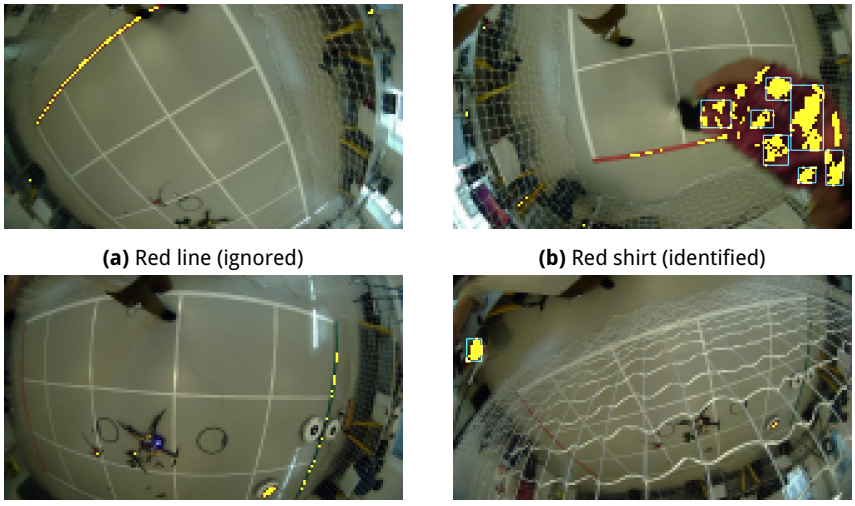


Figure 27: Detection rate for the video in Figure 23b. We manually counted the number of targets actually in the frame (gray), the number of targets that were correctly identified (black), and incorrectly identified (stipled).

4.3.3 Detection rate: Small-scale test at lab

Figure 29 shows the detection rate for the video in Figure 23c and Figure 28 shows a selection of frames. The detector had several false detections due to shirts (b) and pillows (d). It correctly ignored the red line (a) and the green line (c). It identified the red target whenever it was sufficiently centered, which it wasn't in (c). Regardless, some portion of the red plate was always segmented, even when it was far away (d), although the required pixel count threshold caused it to be ignored.



(a) Red line (ignored)

(b) Red shirt (identified)

(c) Green line (ignored) and red target (ignored)

(d) Red target (ignored) and chair pillow (identified)

Figure 28: A selection of frames from Figure 23c. Components that passed the aspect ratio and area tests are drawn with neon bounding boxes. Segmented pixels are otherwise drawn in yellow.

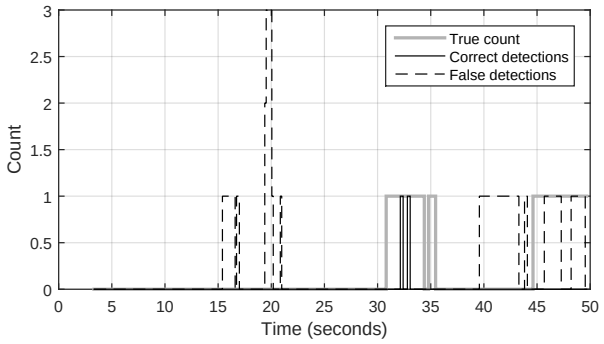
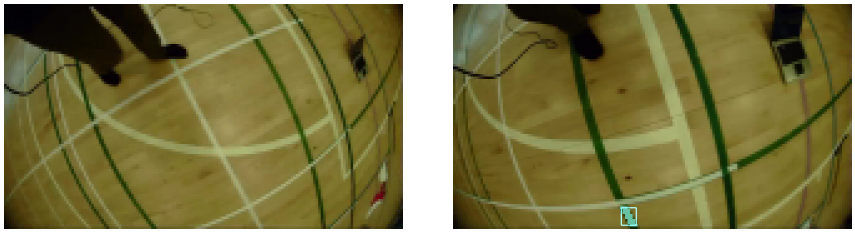


Figure 29: Detection rate for the video in Figure 23c. We manually counted the number of targets actually in the frame (gray), the number of targets that were correctly identified (black), and incorrectly identified (stipled).

4.3.4 Detection rate: Small-scale test at gym

Figure 31 shows the detection rate for the video in Figure 23d and Figure 30 shows a selection of frames. The detector incorrectly identified a partially visible piece of green tape (b), but correctly identified nothing else throughout the rest of the video, even though there were lots of green tape and even something red in the background (a).



(a) Green lines and something red (all ignored)

(b) Part of green line (identified)

Figure 30: A selection of frames from Figure 23d. Components that passed the aspect ratio and area tests are drawn with neon bounding boxes. Segmented pixels are otherwise drawn in blue.

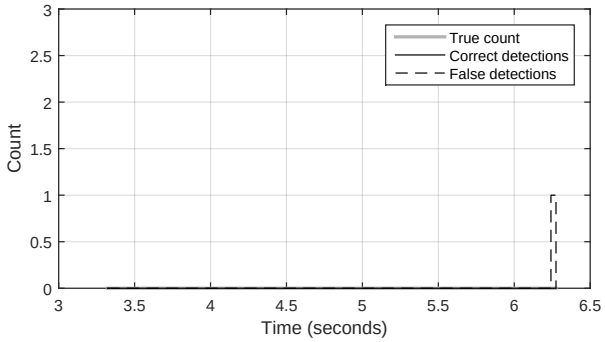


Figure 31: Detection rate for the video in Figure 23d. We manually counted the number of targets actually in the frame (gray), the number of targets that were correctly identified (black), and incorrectly identified (stipled).

4.4 Discussion

The color segmentation algorithm appears to work rather well, although it is clearly fragile to incorrect detections, as there are many things that can have the same geometric and photometric properties as those we search for. First, there are obvious objects, perhaps placed with malicious intent, such as chair pillows or shirts of bystanders. Any object that is not a target or a drone is banned from the inner arena by the competition rules, but may accidentally come into view when the camera is near the arena edges. This risk can be mitigated if the drone position is known (which it hopefully is when the arena edge is visible), by ignoring detections that end up outside the arena.

A far more problematic case are the non-obvious objects, such as a green piece of tape that is partially occluded, perhaps just for a moment by a play of light and shadow, such that it appears to have the correct geometric properties. Since there are no restrictions on the type of pattern of the arena floor, such detections could occur both spuriously or permanently, depending on the particular pattern and the variability of the lighting or viewpoint.

Glare on the top plates cause the segmentation to miss pixels, and predict the wrong center point. This could be mitigated by incorporating a "soft" threshold when growing connected components. That is, a pixel that is added to a component that is mostly comprised of red pixels, can have a lower threshold to entry. Incorrectly labelled pixels like in Figure 26b cause the center point to shift off the plate. Since these outliers appear to be outnumbered by the number of pixels on the actual plate, this could be mitigated by using the median instead of the mean as the center point.

Many of the issues can probably be mitigated by sophisticated tracking. For example, a detection that only lasts a single frame is unlikely to be correct. One could therefore eliminate spurious detections by requiring a detection to occur over several frames, as a measure of its saliency, before accepting it. Tracking the motion of the components over time could be used to reject persistent, yet false, detections. For example, erratic motion is likely a sign of a false detection since the targets move in smooth paths (if not then landing on robots is perhaps not of primary concern at the moment). Zhao et al. [52] use this principle in their algorithm for tracking humans and their limbs.

We had hoped that image alignment would help the color detector remove false detections, and also more provide accurate pose estimates, since the position estimated from on-board sensors and potentially perturbed by glare and outliers might not be sufficiently accurate for control. That being said, we have not quantified the required accuracy, since we have yet to perform our motion-tracked landing experiments.

To remove false detections the objective function must be sufficiently discriminative. That is, correct alignments must have an error that is sufficiently distinct from any other pose estimate. As long as the target is sufficiently far away and centered, the alignment error appears to be an appropriate discriminator. When the target is close or off-center, the alignment error at the correct pose is quite high, due to model errors, and there is less margin for a threshold to discriminate correct and incorrect alignments. This could potentially be mitigated with a 3D model.

The accuracy of the estimated pose is limited by the accuracy of the model. The top plate dominated in all objective functions, and tended to cause erroneous pitching and rolling when the target was close to the camera or off-center. Again, this could be mitigated with a 3D model. Alternatively, a weight mask that decreases the importance of top plate pixels, and increases the importance of pixels on the planar portion of the target, could potentially mitigate this. In particular, the target has two holes on the front and back of the circular plane. These could be assigned a high weight to be prioritized in the alignment, so that a planar model would still be applicable at dramatic viewpoints.

We observed oscillation and signs of instability when the target had a small area in the image. Reducing the smoothing and the number of image pyramids mitigated the issue, but caused very slow convergence for the opposite case. This suggests that we should select the level of detail based on the area of region considered for alignment.

All objective functions were unable to reject prominent glare, and tilted erroneously to compensate. Increasing the regularization of the pitch and roll components mitigated this issue slightly, but in return the position would sometimes be off. Despite being twice as computationally costly as the intensity metric, the descriptor field metric was not more robust against model errors, specular highlights or misleading backgrounds. Moreover, it had a narrower basin of convergence. We think a color segmentation includes specular regions, the intensity metric, and some image preprocessing to eliminate bias and gain, would be more robust and efficient than descriptor fields, but did not investigate this further.

5 Conclusion

In this project report, we have studied the problem of detecting moving objects from a MAV for Mission 7 of the International Aerial Robotics Competition. We did a literature survey and covered both general object detection methods and prior work for the particular IARC problem. We also gave a focused review of color segmentation in uncontrolled lighting and pose estimation by direct image alignment.

Since we believe surveys alone do not fully capture the subtle issues that arise in practice, atleast in the domain of computer vision, we tried to solve the IARC target detection problem ourselves and find out where the challenges lie. We applied the color segmentation and image alignment theory from our survey, and implemented a solution that detects targets in two-steps. First, the top plates are segmented by their distinct color, then, the resulting pose estimates are refined by Gauss-Newton optimization on a dense photometric similarity function. Our motivation for a two-step solution was that color segmentation could globally detect targets from single frames, but with a high risk of false positives and insufficient accuracy for visual servoing. We therefore follow with a local method that can verify detections and generate accurate pose estimates.

We evaluated our solution on videos from the official competition and our lab. The color detector was able to segment the top plates as long as they were sufficiently centered at an appropriate distance from the camera. False detections were primarily caused by intentionally similar objects, such as people or walls outside the arena, and intermittent happenstances, such as marker lines that are momentarily lit and shaded in just the right way. The image alignment algorithm also worked well when the target was sufficiently centered and not too close. In these cases, it provided pose estimates that were as accurate as the model allowed, and the alignment error could discriminate true detections from false detections. However, its performance degraded at viewpoints where the planar model is a poor approximation.

5.1 Future work

In summary, we would not use our solution on the drone. Even though the color segmentation works surprisingly well, and can run at video rate already, the image alignment is not quite as robust as we would like it to be. In particular, it is unable to refine poses or discriminate detections when the target is viewed from certain viewpoints. Moreover, it is too slow as it is, and would require clever optimization to run in real-time on-board, although we don't think this is impossible. We conclude our report with some thoughts on improvements and alternative strategies.

5.1.1 Color segmentation

The pixel center produced by the color segmentation can be anywhere on the actual plate when it is corrupted by specular highlights. Furthermore, if the plate is strongly shaded,

it is possible for the plate to be missed entirely. This could be improved by using a soft threshold in the connected components computation, to allow bright pixels to become part of adjacent green or red components. A more costly alternative is mean-shift segmentation, which might handle large color variation better, and could be initialized with a uniform threshold segmentation. Other color space transformations could also be interesting to study. Tracking could also be used to reduce the effect of noise and lighting, as well as reject false positives. This could be done in image-space or in world-space, and could include various measures of saliency, such as number of frames a detection has been active, or on a motion model.

5.1.2 Image alignment

The target appearance is subject to sudden change, and we may only get a good look once we arrive at the competition venue. We should therefore have a way to easily generate the model. We can already do this if the model is a planar texture, but a 3D model requires more work. One could make a graphical user interface for combining geometric primitives, such as plates and cylinders, and texture mapping the geometry based on camera images by backprojection. Another option is to obtain the geometry directly from camera images using 3D reconstruction software. In its current form, the top plate actually consists of two plates connected by a hinge. This means that the target is not actually a single rigid object, but is instead a kinematic chain. In practice, we see that the moving plate can have a large impact on the silhouette of the plate, especially when viewed from a low vantage point. For further accuracy, the model may need to include the angle of the moving plate as another optimization variable.

Neither the intensity metric nor the descriptor field metric were able to reject specular highlights. Since the descriptor field metric was otherwise costly, complex and had a smaller basin of convergence, it does not appear to have any strong benefits other than being invariant to global lighting. The intensity metric could cope with global gain and bias by filtering the input, as in [34]. An explicit lighting model could include specular highlights in the synthesized image, and could be estimated during runtime or be manually calibrated offline. The latter might be sufficient in our case, since the ceiling lights are likely fixed and constant throughout the day.

We noticed instability, particularly for the descriptor field metric, when the target was far away. This was mitigated by increasing the regularizer term, but even this failed on certain images. The correct fix was to reduce the amount of smoothing for these images, by dropping levels of the image pyramid or reducing the width of the blurring kernel. Ideally, the level of detail should be dynamically selected based on target's area in the image. Small areas should be kept at a high resolution, while large areas should be heavily smoothed.

The practical implementation of the cost function was quite unruly due to all the partial derivatives, and the mathematical intent is mostly lost. This makes the code difficult to understand by other people, but it also makes it tedious to make changes. For example, adding optimization variables would require us to derive several partial derivatives by hand. Optlang [13] is a Domain Specific Language currently in development, and is a small

programming language made specifically for energy minimization algorithms, such as image alignment. It lets the programmer express the cost function in code, and will internally generate symbolic Jacobian expressions and output optimized CUDA code or object files. We hope that this will also support a CPU target, since we think the bottleneck will be the memory transfer from CPU to GPU in each iteration, rather than the time taken per actual iteration. Heterogeneous compute, wherein CPU and GPU memory are shared, could alternatively lower the cost of memory transfer and allow for quite a speedup. For CPU targets, Intel's ISPC lets programmers easily write cross-platform SIMD code, but is not a language focused merely on energy optimization.

We think dense generative methods can give very accurate pose estimates, but only if the image formation model is sufficiently accurate. In particular, it requires a good model of the object and the scene, as well as the geometric and photometric qualities of the camera. It does, however, remain to be seen how much accuracy is actually needed for visual servoing.

5.1.3 Alternative approaches

Not documented in this report is our attempt to detect the elliptical shapes of the targets using the Hough transform. We found a clever way to reduce the problem into a search for circles of a fixed radius. However clever this was, it was optimization for naught, since the main issue was that it, surprisingly, still had many false positives. For example, two intersecting lines look very similiar to the quarter of a circle. Naturally, the circle detector could be combined with the color detector, as was our original intention, but we still struggled to obtain accurate position estimates. The false detections contaminated the true detections in the Hough space, and the target body was quite sparse and not as circular-looking as we would have expected, after we passed it through a Sobel filter. This could perhaps be improved with a different filter, or forgoing a Hough transform altogether.

Convolutional neural networks are rapidly gaining traction in this day and age, and Ascend NTNU has a group working on this. As it stands, evaluating the network takes several hundred milliseconds on a powerful desktop computer. We therefore think this would primarily be interesting for verifying detections from a simpler detector, such as our color segmentation. Other machine learning techniques, such as Support Vector Machines, are also viable alternatives, and have been used in prior IARC work.

Feature-based methods is the traditional branch of object detection, and could be used to detect targets globally from single frames, and even provide accurate 3D pose estimates. We did have a group working on this, who found that SIFT features were infact applicable, although quite slow. Binary features have been gaining interest as of late, and could be an alternative to SIFT features.

A Derivations

A.1 Interaction matrices for pinhole and equidistant camera projections

Denote by $H_o^c = \{R_o^c, T_o^c\}$ the object frame $\{O\}$ decomposed in the camera frame $\{C\}$, and denote by $\boldsymbol{u} = (u, v)$ the projection of a point p^o fixed to the object, obtained by transforming the point to the camera frame $p^c = H_o^c \circ p^o = (x, y, z)$ and projecting the result into the image $\boldsymbol{u} = \pi(p^c)$. Suppose the object frame is undergoing rotation and translation described by the twist $\xi = (\boldsymbol{\omega}, \boldsymbol{v}) \in \mathrm{se}(3)$, i.e. $\dot{H}_o^c = \xi^\times H_o^c$ [5]. The resulting motion of the point on the object, decomposed in the camera frame, is then

$$\begin{aligned}
\dot{p}^c &= \frac{d}{dt}(R_o^c p^o + T_o^c) \\
&= \dot{R}_o^c p^o + R_o^c \dot{p}^o + \dot{T}_o^c \\
&= \boldsymbol{\omega}^\times R_o^c p^o + 0 + \dot{T}_o^c \\
&= \boldsymbol{\omega}^\times (R_o^c p^o + T_o^c) + \dot{T}_o^c - \boldsymbol{\omega}^\times T_o^c \\
&= \boldsymbol{\omega}^\times p^c + \boldsymbol{v}
\end{aligned}$$

which we can write in scalar form as

$$\begin{aligned}
\dot{x} &= \boldsymbol{\omega}_y z - \boldsymbol{\omega}_z y + \boldsymbol{v}_x \\
\dot{y} &= \boldsymbol{\omega}_z x - \boldsymbol{\omega}_x z + \boldsymbol{v}_y \\
\dot{z} &= \boldsymbol{\omega}_x y - \boldsymbol{\omega}_y x + \boldsymbol{v}_z
\end{aligned} \tag{42}$$

The pinhole projection Eq. **??** is

$$\begin{aligned}
u &= u_0 + f\frac{x}{-z} \\
v &= v_0 - f\frac{y}{-z}
\end{aligned}$$

and the time derivative, using the chain rule, is

$$\begin{aligned}
\dot{u} &= f\frac{-\dot{x}z + x\dot{z}}{z^2} \\
\dot{v} &= -f\frac{-\dot{y}z + y\dot{z}}{z^2}
\end{aligned}$$

Substituting Eq. 42 we can write this as

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} f\frac{xy}{z^2} & -f(1+\frac{x^2}{z^2}) & f\frac{y}{z} & -f\frac{1}{z} & 0 & f\frac{x}{z^2} \\ f(1+\frac{y^2}{z^2}) & -f\frac{xy}{z^2} & -f\frac{x}{z} & 0 & -f\frac{1}{z} & f\frac{y}{z^2} \end{bmatrix} \begin{bmatrix} w_x \\ w_y \\ w_z \\ v_x \\ v_y \\ v_z \end{bmatrix}$$

[5] This is the *left-multiplicative* formulation, and leads to left-hand composition: $H_o^c \leftarrow \exp(\xi) H_o^c$. The alternative is the *right-multiplicative* formulation, where $\dot{H}_o^c = H_o^c \xi^\times$, which leads to right-hand composition.

or in concise matrix form

$$\dot{\boldsymbol{u}} = J(x, y, z)\xi$$

The 2×6 matrix J is commonly called the *image Jacobian* or the *interaction matrix* in robotic literature related to vision-based control [8, 44]. A similiar derivation holds for the equidistant projection Eq. 11, where

$$\begin{aligned} u &= u_0 + r\frac{x}{l} \\ v &= v_0 - r\frac{y}{l} \\ r &= f\tan^{-1}\frac{l}{-z} \\ l &= \sqrt{x^2 + y^2} \end{aligned}$$

Computing $\dot{u}$ and $\dot{v}$ is merely a matter of being meticulous, or — for those of us with less time on our hands — being quaint with a symbolic equation processor. Let us investigate how we can compute the derivatives using the symbolic toolbox from MATLAB. First, we implement the above equations as symbolic expressions

```
syms u0 v0 x y z f;
l = sqrt(x*x + y*y);
r = f*atan(-l/z);
u = u0 + r*x/l;
v = v0 - r*y/l;
```

We then apply the chain rule on each coordinate

$$\begin{aligned} \dot{u} &= \frac{\partial u}{\partial x}\dot{x} + \frac{\partial u}{\partial y}\dot{y} + \frac{\partial u}{\partial z}\dot{z} \\ \dot{v} &= \frac{\partial v}{\partial x}\dot{x} + \frac{\partial v}{\partial y}\dot{y} + \frac{\partial v}{\partial z}\dot{z} \end{aligned}$$

using the `diff` function on `u` and `v`, e.g.

```
>> diff(u, x)
ans = f*x^2*atan((x^2 + y^2)^(1/2)/z))/(x^2 + y^2)^(3/2) -
      f*atan((x^2 + y^2)^(1/2)/z)/(x^2 + y^2)^(1/2) -
      f*x^2/(z*(x^2 + y^2)*((x^2 + y^2)/z^2 + 1))
```

Unfortunately, the output is not immediately useful since we would rather simplify

```
f*atan((x^2 + y^2)^(1/2)/z))
```

to $-r$. When we actually compute the derivatives, both the camera-space coordinate p^c and the image-space coordinate $\boldsymbol{u}$ are available. Therefore, we have a cheap way to compute $r = \sqrt{u^2 + v^2}$ with a single norm evaluation, instead of the above expression that has one inverse tangent, a norm, *and* a division. We will spare the reader the tiresome procedure of simplifying each expression, suffice to say it came in handy to have a text editor with find-and-replace functionality. After some minutes of work of labour, we can redefine the partial derivatives in terms of a few (known) auxiliary variables

```
syms r c s l L f;
dudx = r*s*s/l - f*c*c*z/L;
dudy = -c*s*r/l - f*c*s*z/L;
dudz = f*x/L;
dvdx = f*c*s*z/L + r*c*s/l;
dvdy = -r*c*c/l + f*s*s*z/L;
dvdz = -f*y/L;
```

where $c = x/l$, $s = y/l$ and $L = x^2 + y^2 + z^2$. By redefining `r` and `l` we can prevent `MATLAB` from expanding those variables in subsequent expressions. Finally, we can evaluate the total derivatives

```
syms wx wy wz vx vy vz;
dx = wy*z - wz*y + vx;
dy = -wx*z + wz*x + vy;
dz = wx*y - wy*x + vz;
du = dudx*dx + dudy*dy + dudz*dz;
dv = dvdx*dx + dvdy*dy + dvdz*dz;
```

and, if the expressions turn out to be linear, we can extract the Jacobian matrix using the `collect` function

```
collect(du, [wx wy wz vx vy vz])
collect(dv, [wx wy wz vx vy vz])
```

This gives

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} Az + f\frac{xy}{L} & Bz - f\frac{x^2}{L} & -By - Ax & B & -A & f\frac{x}{L} \\ Cz - f\frac{y^2}{L} & Dz + f\frac{xy}{L} & -Cx - Dy & D & -C & -f\frac{y}{L} \end{bmatrix} \begin{bmatrix} w_x \\ w_y \\ w_z \\ v_x \\ v_y \\ v_z \end{bmatrix}$$

where

$$\begin{aligned} A &= csr/l + fcsz/L \\ B &= ssr/l - fccz/L \\ C &= ccr/l - fssz/L \\ D &= csr/l + fcsz/L \\ c &= x/l \\ s &= y/l \\ L &= x^2 + y^2 + z^2 \end{aligned}$$

B Addendum

References

[1] Hatem Alismail. *Direct Pose Estimation and Refinement.* PhD thesis, The University of Adelaide, 2016.

[2] Hatem Alismail, Brett Browning, and Simon Lucey. Bit-planes: Dense subpixel alignment of binary descriptors. *arXiv preprint arXiv:1602.00307*, 2016.

[3] Hatem Said Alismail, Brett Browning , and Simon Lucey. Enhancing direct camera tracking with dense feature descriptors. In *Asian Conference on Computer Vision (ACCV)*. Springer, May 2016.

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

[5] Jose-Luis Blanco. A tutorial on se (3) transformation parameterizations and on-manifold optimization. *University of Malaga, Tech. Rep*, 2010.

[6] Hilton Bristow and Simon Lucey. In defense of gradient-based alignment on densely sampled sparse features. In *Dense Image Correspondences for Computer Vision*, pages 135–152. Springer, 2016.

[7] Changhyun Choi and Henrik I Christensen. Robust 3d visual tracking using particle filtering on the special euclidean group: A combined approach of keypoint and edge features. *The International Journal of Robotics Research*, 31(4):498–519, 2012.

[8] Peter Corke. *Robotics, vision and control: fundamental algorithms in MATLAB*, volume 73. Springer, 2011.

[9] Alberto Crivellaro, Pascal Fua, and Vincent Lepetit. Dense methods for image alignment with an application to 3d tracking. Technical report, 2014.

[10] Alberto Crivellaro and Vincent Lepetit. Robust 3d tracking with descriptor fields. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3414–3421. IEEE, 2014.

[11] Alberto Crivellaro, Mahdi Rad, Yannick Verdie, Kwang Moo Yi, Pascal Fua, and Vincent Lepetit. A novel representation of parts for accurate 3d object detection and tracking in monocular images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4391–4399, 2015.

[12] Bertrand Delabarre. *Contributions to dense visual tracking and visual servoing using robust similarity criteria.* PhD thesis, Rennes 1, 2014.

[13] Zachary DeVito, Michael Mara, Michael Zollöfer, Gilbert Bernstein, Christian Theobalt, Pat Hanrahan, Matthew Fisher, and Matthias Nießner. Opt: A domain specific language for non-linear least squares optimization in graphics and imaging. *arXiv preprint arXiv:1604.06525*, 2016.

[14] Tom Drummond and Roberto Cipolla. Real-time visual tracking of complex structures. *IEEE Transactions on pattern analysis and machine intelligence*, 24(7):932–946, 2002.

[15] Olav Egeland and Jan Tommy Gravdahl. *Modeling and simulation for automatic control*, volume 76. Marine Cybernetics Trondheim, Norway, 2002.

[16] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *arXiv preprint arXiv:1607.02565*, 2016.

[17] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014.

[18] J-M Geusebroek, Rein Van den Boomgaard, Arnold W. M. Smeulders, and Hugo Geerts. Color invariance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(12):1338–1350, 2001.

[19] Håkon Hagen Helgesen. Object detection and tracking based on optical flow in unmanned aerial vehicles. 2015.

[20] Dongze Huang, Zhihao Cai, Xiang He, and Yingxun Wang. A svm embedded particle filter for multi-object detection and tracking. In *Guidance, Navigation and Control Conference (CGNCC), 2014 IEEE Chinese*, pages 2094–2099. IEEE, 2014.

[21] Michal Irani and P Anandan. About direct methods. In *International Workshop on Vision Algorithms*, pages 267–277. Springer, 1999.

[22] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *IEEE transactions on pattern analysis and machine intelligence*, 34(7):1409–1422, 2012.

[23] Juho Kannala and Sami S Brandt. A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. *IEEE transactions on pattern analysis and machine intelligence*, 28(8):1335–1340, 2006.

[24] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007.

[25] Vincent Lepetit and Pascal Fua. *Monocular model-based 3D tracking of rigid objects*. Now Publishers Inc, 2005.

[26] Matthew M Loper and Michael J Black. Opendr: An approximate differentiable renderer. In *European Conference on Computer Vision*, pages 154–169. Springer, 2014.

[27] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981.

[28] Eric Marchand, Hideaki Uchiyama, and Fabien Spindler. Pose estimation for augmented reality: a hands-on survey. 2016.

[29] Carol Martínez, Iván F Mondragón, Pascual Campoy, José Luis Sánchez-López, and Miguel A Olivares-Méndez. A hierarchical tracking strategy for vision-based ap-

plications on-board uavs. *Journal of Intelligent & Robotic Systems*, 72(3-4):517–539, 2013.

[30] Peter Meer. Robust techniques for computer vision. *Emerging topics in computer vision*, pages 107–190, 2004.

[31] Abhimitra Meka, Michael Zollhöfer, Christian Richardt, and Christian Theobalt. Live intrinsic video.

[32] Priyanka Mukhopadhyay and Bidyut B Chaudhuri. A survey of hough transform. *Pattern Recognition*, 48(3):993–1010, 2015.

[33] Takuma Nakamura and Eric N Johnson. Vision-based multiple model adaptive estimation of ground targets from airborne images. In *Unmanned Aircraft Systems (ICUAS), 2016 International Conference on*, pages 598–607. IEEE, 2016.

[34] Richard Newcombe. *Dense visual SLAM.* PhD thesis, Imperial College London, 2012.

[35] Tien Dat Ngo. Template-based monocular 3-d shape reconstruction and tracking using laplacian meshes. 2016.

[36] Mark S Nixon and Alberto S Aguado. *Feature extraction & image processing for computer vision.* Academic Press, 2012.

[37] Jorge Nocedal and Stephen Wright. *Numerical optimization.* Springer Science & Business Media, 2006.

[38] Youngmin Park, Vincent Lepetit, and Woontack Woo. Handling motion-blur in 3d tracking and rendering for augmented reality. *IEEE transactions on visualization and computer graphics*, 18(9):1449–1459, 2012.

[39] Victor A Prisacariu and Ian D Reid. Pwp3d: Real-time segmentation and tracking of 3d objects. *International journal of computer vision*, 98(3):335–354, 2012.

[40] Helge Rhodin, Nadia Robertini, Christian Richardt, Hans-Peter Seidel, and Christian Theobalt. A versatile scene model with differentiable visibility applied to generative pose estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 765–773, 2015.

[41] Gonzalo R Rodríguez-Canosa, Stephen Thomas, Jaime del Cerro, Antonio Barrientos, and Bruce MacDonald. A real-time method to detect and track moving objects (datmo) from unmanned aerial vehicles (uavs) using a single camera. *Remote Sensing*, 4(4):1090–1111, 2012.

[42] Hui-Liang Shen and Zhi-Huan Zheng. Real-time highlight removal using intensity ratio. *Applied optics*, 52(19):4483–4493, 2013.

[43] Geraldo Silveira and Ezio Malis. Real-time visual tracking under arbitrary illumination changes. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–6. IEEE, 2007.

[44] Mark W Spong, Seth Hutchinson, and Mathukumalli Vidyasagar. *Robot modeling and control*, volume 3. Wiley New York, 2006.

[45] Charles V Stewart. Robust parameter estimation in computer vision. *SIAM review*, 41(3):513–537, 1999.

[46] Richard Szeliski. Image alignment and stitching: A tutorial. *Foundations and Trends® in Computer Graphics and Vision*, 2(1):1–104, 2006.

[47] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[48] Justus Thies, Michael Zollhöfer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Face2face: Real-time face capture and reenactment of rgb videos. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1, 2016.

[49] Henning Tjaden, Ulrich Schwanecke, and Elmar Schömer. Real-time monocular segmentation and pose tracking of multiple objects. In *European Conference on Computer Vision*, pages 423–438. Springer, 2016.

[50] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *Acm computing surveys (CSUR)*, 38(4):13, 2006.

[51] Zhengyou Zhang. Parameter estimation techniques: A tutorial with application to conic fitting. *Image and vision Computing*, 15(1):59–76, 1997.

[52] Tao Zhao and Ramakant Nevatia. Tracking multiple humans in complex situations. *IEEE transactions on pattern analysis and machine intelligence*, 26(9):1208–1221, 2004.

[53] Todd Zickler, Satya P Mallick, David J Kriegman, and Peter N Belhumeur. Color subspaces as photometric invariants. *International Journal of Computer Vision*, 79(1):13–30, 2008.