Guidance of a Mobile Robot using Computer Vision over a Distributed System

Oliver M C Williams (JE)

Abstract

Previously, there have been several 4th-year projects using computer vision to follow a robot and thereby control it. The essence of this project is disconnect-edness. The robot is controlled over a wireless link and is no-longer dependent on an umbilical cable. The user control has also been abstracted using CORBA allowing operations to be invoked on the robot from a remote location. Thanks to CORBA, the interface is universal and a small client program can be written to run on any platform connected to the Internet. An aerial camera connected to a server machine monitors the robot. This uses a B-spline snake, updated in an affine manner, to estimate the robot’s position in the image. A plane-to-plane homography is established using four known points and the system can map image coordinates to world coordinates. The server can therefore estimate the robot’s position in world coordinates and a discrete control strategy then moves the robot between positions. Such movement may be initiated from either the server or a client machine using CORBA.

1 Introduction

The idea of a mobile robot to provide assistance either in the home, office or in more hostile environments (e.g. bomb-disposal or nuclear reactors) has existed for many years and such systems are available today. Unfortunately, they are typically expensive and by no means ubiquitous in the way that 1950s and 60s science fiction would have had us believe.

The major limitations to including robots in homes and offices are the infrastructure changes they require. Computer vision means, however, that robots can be monitored from just a few inexpensive cameras and the recent availability of wireless network solutions (IEEE 802.11 and Bluetooth in particular) has decimated the costs of implementation.

A key part of a robot package is how humans are to interact with it. It may be that people wish to work with their robot ‘face-to-face’, via a home or office workstation or even with their mobile telephone. By using a distributed system, the server managing
the robot’s behaviour presents an uncomplicated interface to the Internet. To control
the robot all that is needed is a simple client program (which could be written to operate
on any number of platforms) and a reference to our server. Robot operations may then
be invoked from anywhere in the world and it is this functionality that increases the
possible applications by orders of magnitude. Here is an example:

1.1 The household robot

You possibly have a dog at home. Many people’s pet is able to alert them if there is an
intruder in the house, but how many can do it whilst you are at work? And how many
can pass a message to the kids that you’re going to be late home?

I am not suggesting that people swap Lassie for his electronic equivalent, but hav-
ing a mobile system at home, equipped with cameras and interfaced with household
equipment such as the burglar alarm, oven, or front-door could be very useful. If you
want to set the oven to a certain temperature in time for when you get home, you can
ask the robot to go to the oven and, via a Bluetooth connection, set it to 200°C. A
small LCD screen on our robot’s back could be used to open a videophone-style link
for communication with someone anywhere in the house.

All these toy applications are available now, but the availability of small, inexpen-
sive processors means that our environment is increasingly ‘wired’, and this is set to
continue. Mobile robots will be involved in this connected community, exploiting the
ubiquitous, distributed processing capabilities available to it. Over the coming decades
this could prompt a paradigm-shift in the way humans and machines cooperate.

1.2 This paper

In this paper I present the results of a yearlong project to create a platform on which
a system, as described above, could be built. The hardware is inexpensive, and the
computing requirements are typical of a home PC. This system can track and guide a
robot around its world using an aerial camera, moving it to user-specified locations.
The user-interface to this system is distributed using CORBA and hence a user may
control the robot from a remote location on a variety of platforms.

The coming sections will briefly introduce the major technologies involved before
presenting the complete system and the directions this work could take from here.

2 Plane-to-plane Homography

To a good approximation, our robot lives in a 2D world (i.e. it rides on a table). Frames
grabbed from the aerial CCD are also in 2D. These planes are arbitrary and, if we are
to infer the robot’s position from the image plane, we need to know the image—→world
mapping.
Such a mapping is known as a homography and, in this plane-to-plane case can be represented as a matrix operation on the homogeneous coordinates[4]. We wish to find the projection matrix, $P_{ps} \in \mathbb{R}^{3 \times 3}$ such that:

$$P_{ps} = \begin{bmatrix} k_u & 0 & u_0 \\ 0 & k_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & T_x \\ r_{21} & r_{22} & T_y \\ r_{31} & r_{32} & T_z \\ 0 & 0 & 1 \end{bmatrix}$$

(1)

where $k_u$, $k_v$, $u_0$ and $v_0$ are CCD properties, $f$ is the focal-length of the lens and $r_{mn}$, $T_m$ indicate a rigid-body transformation between the two planes.

$P_{ps}$ has nine elements, however, the homography is invariant to scale and consequently has only eight degrees of freedom. To find $P_{ps}$, eight independent equations are required defining the mapping. These come from four points whose location is known in both coordinate systems (each $(x, y)$ pair encodes two equations). In our application, the user clicks on these four, measured points and the system can convert to and from each plane, as shown in Figure 1.

![Figure 1: System’s interpretation of ground plane](image)

### 3 Tracking and Controlling the Robot

A simple open-loop controller exists that moves the robot using timers and dead reckoning. This has limited accuracy over anything but the shortest journeys meaning closed-loop control is required. For this to be possible we must know the robot’s true position.

#### 3.1 The B-spline snake

A B-spline[5] is a $C^2$ continuous curve defined by $m$ control points. Equation 2 is a parametric representation of curve-segment $i$ of the entire curve defined by control
points \( p_i \).

\[
Q_i(s) = \frac{1}{6} \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3} \end{bmatrix}
\]  

(2)

\( i \oplus n \) implies that the complete curve is a closed one. Most importantly, each segment of the curve depends only on four control points. Hence a B-spline exhibits local control; one segment can be adjusted without affecting the entire curve.

By fitting such a curve around the edges[1] our robot creates in the image, we can track it. To do so requires us to find such edges and comprehensive edge-finding[3] is too expensive to perform in real-time. A local, heuristic method is used[2]. The snake is updated in an affine manner once it is fitted to the robot and thereby maintains the same topology as it moves, in the same way as the robot does in reality.

### 3.2 Closed-loop control

Provided the snake is tracking the robot correctly, we can use the homography to infer the robot’s position in world coordinates. We then use this information to reliably guide the robot to its destination. This was done using a discrete controller that moves the robot by successive open-loop journeys. This proved to be a quick and reliable method of control, as the robot could pause (briefly) between trips to ensure stable tracking.

### 4 Distributed Interface

CORBA[6][7] has been used to present a user-control interface to the Internet. All of the software in this project has been written in C++ and is therefore object-oriented (OO). CORBA is also OO and the interface takes the form of an object. Over this interface, a remote client can retrieve the most recent image plane, the robot location or dispatch the robot to a new location.

Remote invocation works by separating the implementation of the user-control class from its interface. Information is marshalled between machines by an object request broker (ORB). Proxies called the Stub and Skeleton at the client and server sides provide translation between CORBA and our C++ implementation. This is summarized in Figure 2.

### 5 The Complete System

Figure 3 shows the robot moving across the floor in a series of open-loop trips. It was found that tracking is lost on approximately 5% of journeys, but this could be improved by slowing the robot down, or by providing a mask that increases the contrast with the white floor.
One client application exists to operate on a Windows PC. This was tested on an Ethernet network, on a 56kbps PPP dial-up connection and over GPRS using a mobile telephone. Unsurprisingly, the response times vary between these media, however, after months of using the ORB, it has yet to fail.

Objects encapsulate all of the major functionalities of this system. This OO approach to the software engineering satisfies the objective of creating a platform on top of which more complex systems can be built, either through inheritance or by using these simple building blocks as they stand.

### 5.1 Recommendations for future work

As CORBA is transparent to programming language, other client applications should be written to operate on different platforms (e.g. a Java applet or a light-weight application to use on a PDA).

The control strategy used here is effective, but more elegant solutions are possible. Finally, CORBA can provide a naming service for distributed objects, analogous to a URL. This should be instigated to provide even more transparency to the user, who currently has to possess a text file addressing the object.

### References


Figure 3: Robot moving to user-specified destination


