Bootstrapping a robot’s kinematic model

Alan Broun\textsuperscript{a,}\textsuperscript{*}, Chris Beck\textsuperscript{b}, Tony Pipe\textsuperscript{a}, Majid Mirmehdi\textsuperscript{b}, Chris Melhuish\textsuperscript{a}

\textsuperscript{a} Bristol Robotics Laboratory, Bristol, UK
\textsuperscript{b} Visual Information Laboratory, University of Bristol, Bristol, UK

HIGHLIGHTS

\begin{itemize}
  \item We present a system that is able to autonomously build a 3D model of a robot’s hand.
  \item A hand is located and moved to the centre of the robot’s field of view using exploratory motions.
  \item The system and the built models are validated by a number of experiments.
\end{itemize}

ARTICLE INFO

Article history:
Available online 23 October 2013

Keywords:
Kinematic identification
Model building
Body schema

ABSTRACT

We present a system that is able to autonomously build a 3D model of a robot’s hand, along with a kinematic model of the robot’s arm, beginning with very little information. The system starts by using exploratory motions to locate and centre the robot’s hand in the middle of its field of view, and then progressively builds the 3D and kinematic models. The system is flexible, and easy to integrate with different robots, because the model building process does not require any fiducial markers to be attached to the robot’s hand. To validate the models built by the system we perform a number of experiments. The results of the experiments demonstrate that the hand model built by the system can be tracked with a precision in the order of 1 mm, and that the kinematic model is accurate enough to reliably position the hand of the robot in camera space.

\textsuperscript{*}Corresponding author. Tel.: +44 7815773017.
E-mail addresses: abroun@alanbroun.net, alan.broun@brl.ac.uk (A. Broun), csxcb@compsci.bristol.ac.uk (C. Beck), tony.pipe@brl.ac.uk (T. Pipe), majid@compsci.bristol.ac.uk (M. Mirmehdi), chris.melhuish@brl.ac.uk (C. Melhuish).

1. Introduction

When building a robot, or larger robotic system, it is common for designers to explicitly give the robot all of the models it needs to control itself (in the form of kinematic and dynamics models) and all of the models it needs to interact with objects in the world around it (in the form of 3D CAD models). This is a practical approach, but it lacks flexibility in cases where properties of the real world may deviate from the model during the lifetime of the system. For example, an arm may degrade or be replaced with an arm of different dimensions, or novel objects may be encountered by the robot, meaning that existing models may be found wanting. In these cases the models must be extended or replaced. The task of taking the measurements for models may also be complicated by the robot being in hard to reach, or hazardous locations.

Our work focuses on building robots and robotic systems which can autonomously construct models of themselves and the external objects with which they interact. In particular, we explore the use of active 3D vision as a tool that a robot can use to explore itself, and its surroundings, in order to autonomously construct models.

The increasing availability of reasonably priced depth cameras such as the Mesa Imaging SwissRanger or the Microsoft Kinect has made it easier for robotic systems to perceive the world in 3D. These cameras provide depth values for pixels in the image, and so produce a 3D point cloud in camera space. The quality of the point clouds produced by these cameras reduces the need for researchers to set up technically challenging stereo camera systems, which often rely on the presence of highly textured areas in order to achieve reasonably similar results.

A practical problem for a robot with 3D vision is the task of relating the movement of its body to the Cartesian space of its camera system, so that it can interact with objects it sees. More fundamental than that, it may also be a problem for the robot to work out which parts of a 3D image belong to its body and to its hand.

We present a solution to both of these problems in the form of an extended version of the system we presented in [1]. The system allows a robot to reliably identify its hand in its field of view, and then to build a kinematic model of its arm in camera space. Building the kinematic model in camera space implicitly determines the transformation between camera space and the...
model. Once the kinematic model is built, inverse kinematics can be used to accurately move the manipulator of the robot in camera space. In effect, the robot is therefore able to ‘bootstrap’ itself, from a state of fairly limited knowledge, to having a kinematic model of its arm, which it can then use to further interact with, and to explore, the world.

The system works by first using a series of exploratory motions to roughly identify the location, and extents of the robot’s hand in its field of view. It then uses a simple form of visual servoing to move the hand to the centre of the field of view, in order to maximise the quality of the subsequent processes. Once centred, the system builds a model of the robot’s hand by turning the hand in front of a Kinect depth camera, whilst aligning and merging the point clouds obtained from the Kinect into a common reference frame. The model is then used to track future movements of the hand, by aligning the model against incoming point clouds. This system has the useful property of not just providing an estimate for the transformation of the hand in camera space, but also providing a 3D model of the hand which can be useful for other tasks such as planning grasps, or checking for collisions between the robot and the environment.

Once a model of the robot’s hand has been built and is being tracked, we then use it to automatically build a kinematic model of the robot’s arm by tracking the movement of the hand as each revolute joint in the arm is rotated in turn. This allows us to build an accurate model of the arm, starting with very little information. This is an advantage, as a robot that can deduce information for itself, is potentially more robust, and requires less work to commission.

The rest of the paper proceeds as follows. Section 2 describes related work and reviews the techniques which we use to build our system. Section 3 describes the robotic platform we use for our experiments. Section 4 provides a description of how the robotic hand is modelled and tracked along with details of automatically building a kinematic skeleton for the robotic system. Section 5 evaluates the accuracy of the system, and Section 6 presents conclusions along with ideas for future work.

2. Background and related work

2.1. Exploratory motion and active vision

The work of Ballard [2] was amongst the first to look in depth at camera systems which were not simply passive. Ballard observed that more information may be obtained from a visual scene, or obtained at a lower computational cost, through the process of moving the camera system and observing the scene from a number of different viewpoints. Such systems are often termed active vision systems to distinguish them from passive vision systems.

An alternative to moving the camera in a system is to move the object or scene being observed. The idea of using exploratory motions to both identity a robot’s end effector, and also to segment objects of interest from the background was explored in work by Marjanovic et al. [3] and later Fitzpatrick and Metta [4] at MIT as part of the work on the Cog robot. The technique was explored in detail by Broun and Studley [5], who showed that a waving exploratory motion could be reliably detected, even in the presence of a large amount of distractive motion. Work has also been done by Katz and Brock [6] on using exploratory motions to autonomously identify the structure of articulated objects.

2.2. Object modelling and tracking

When building models from range data, such as that obtained from a laser scanner or depth camera, the Iterative Closest Point (ICP) algorithm presented by Chen and Medioni [7] and Besl and McKay [8] is a widely used algorithm for aligning one depth camera frame with either another frame, or with a reference model.

The ICP algorithm has been the subject of much research since its initial presentation. Rusinkiewicz and Levoy [9] identified the key stages that make up the ICP algorithm, outlining a number of techniques for making the algorithm more efficient and speeding up convergence. The ICP algorithm was used as a key part of an object modelling and tracking system built by Weise et al. [10], and a very similar system was used to build models of objects held in a robot’s hand by Krainin et al. [11]. In both of these systems, models were constructed by first aligning point clouds from a depth camera into a common coordinate frame, using the ICP algorithm. Subsequently, corresponding points from the point clouds were averaged together to form surfel (surface element) models. Surfels as described in [12] are orientated 3D points, which can be used to describe complex geometric objects without explicit connectivity information. The advantage of averaging point clouds together to form a surfel model is that it smooths out a lot of noise that would otherwise accumulate as a result of estimating lots of small transformations [13].

Tracking a robot’s hand in camera space is a special case of tracking an arbitrary 3D object in camera space, and this is an active area of research, with Lepetit and Fua [14] providing a comprehensive survey of the main techniques. Fiducial tracking [15] involves tracking markers attached to the object of interest. Model-based tracking involves posing a 3D model of the object of interest to best match the information coming from the camera. This method has been used extensively in human hand tracking applications, such as [16].

2.3. Kinematic identification

There are a number of methods available for identifying a robot’s kinematic model. Early work from the 1980s includes Circle Point Analysis (CPA) used by Stone et al. [17], and described in detail by Mooring et al. [18]. CPA involves fitting circles to observed endpoint locations in order to identify the axis of revolution for a revolute joint. Another method for identifying the joint axes of a robot is the Jacobian Matrix Method of Bennett and Hollerbach [19]. This method requires either joint torque sensors or a method of estimating the linear and angular velocity of the robot’s end link [19].

More recently, the field of developmental robotics has taken an interest in kinematic identification. Here, it has been explored as part of more general efforts to enable robots to build and maintain a body schema for themselves. In the context of robotics, Hoffmann et al. [20] describe a body schema as a group of body representations, which allow an embodied agent to control its actions, and to integrate sensory information such as vision or touch into common frames of reference. These representations may include kinematic and dynamic models, and the emphasis is on building the models autonomously. The aim is to give a robot the ability to adapt to changes in its body due to damage, or to dynamically extend its body schema to allow the use of tools. Hersch et al. [21] built a kinematic model of a robot by observing end effectors using an iterative gradient descent approach. Sturm et al. [15] presented a system that uses a Bayesian network to learn arbitrary kinematic chains, which can also cope with changes in the kinematic chains as the system runs. This system however, requires observations of all joint positions to build the kinematic chain, whereas our system only needs to observe the movement of the end of the chain.

Finally, recent work by Hart and Scassellati [22] takes a similar approach to the one presented here. The difference lies in the fact that the method of Hart and Scassellati requires an Augmented Reality (AR) marker to track the hand, whereas our method builds and tracks a complete model of the robot’s hand, and so can operate without AR markers.
3. Robotic platform

The work presented in this paper was done with a Bristol and Elumotion Robotic Torso (BERT) robot (Fig. 1). This is an upper torso humanoid robot, which uses harmonic drives, for accurate, low backlash movement, and brushless DC motors. The motors are controlled with EPOS motor controllers, and these in turn are driven by higher level software running on a PC with communication over a Controller Area Network (CAN) bus. A Microsoft Kinect acts as BERT’s vision system providing both colour and depth images, at a resolution of 640 × 480 pixels, and a frame rate of 30 frames per second. Our robot control software makes extensive use of the open source Robotic Operating System (ROS) libraries [23], which provide a convenient way to integrate our algorithms with a variety of open source functions.

4. Method

Since we are interested in a self-learning robot, we need to reduce the amount of a priori information available, while making certain assumptions to make the task tractable. We assume that an attainable starting pose for the robot is given, such that the robot’s hand is visible to the robot’s vision system. We also expect that the range of motion and gearing ratio (used to convert from encoder ticks to angles), of all of the robot’s actuators are known. Finally, we assume that the relative order of the robot’s actuators is available, i.e., we know that the elbow actuators come after the shoulder actuators etc. Everything else the robot needs to know, it deduces in a series of steps. We are interested in reducing this initial set of assumptions still further, but postpone discussion of how this may be done until Section 6.

The complete system for building a kinematic model of the robot’s arm involves a number of stages. The first thing the robot must do is to work out where in its field of view its hand is located, as described in Section 4.1. Once the robot has located and identified its hand, we centre the hand in its field of view to obtain a good clear view of it, and to avoid any inconsistencies that could arise if the later hand modelling step was to be performed from a variety of different viewpoints. The problem is that at this stage the robot has no kinematic model, and so therefore, no way of working out how moving its actuators will change the position of its hand, in its field of view. In Section 4.2, we describe how this problem can be overcome, and centre the robot’s hand in its field of view. Once the robot’s hand is centred, we use the Kinect to construct and track a 3D model of it. This work is described in Section 4.3. Finally, Section 4.4 describes how we use the ability to track the robot’s hand to deduce the position and orientation of the actuator joint axes, and therefore, to finally build the kinematic model of the robot’s arm.

4.1. Finding the hand

Building on the work in [4] and our previous work in [5], we use a waving motion as the exploratory clue to locate the hand. To generate this motion, we pass a sine wave through the wrist actuator, which means that the angle of the wrist actuator \( \alpha \) as a function of time \( t \) is

\[
\alpha(t) = A \sin(2\pi ft),
\]

where \( A \) is the amplitude of the wave, and \( f \) is the frequency of the wave. The instantaneous linear velocity \( v \) of a point on the hand at distance \( r \) from the actuator axis is

\[
v(t) = \rho \omega(t),
\]

where \( \omega \), the angular velocity of the actuator, i.e., the derivative of the wrist actuator angle, is

\[
\omega(t) = \frac{d\alpha}{dt} = 2\pi Af \sin \left( \frac{\pi}{2} - 2\pi ft \right).
\]

The proportion of the linear velocity that will be seen on the image plane \( v_{\text{image}} \), when the moving hand is viewed by the Kinect is

\[
v_{\text{image}}(t) = \sin \beta v(t),
\]

where \( \beta \) is the angle between the Kinect camera axis and the instantaneous velocity vector at the observed point. Combining (2), (3), and (4), we see that the sine wave we put into the actuator will be phase shifted, and have a very different amplitude by the time it is observed on the image plane. However, it will still be recognisable as a sine wave, and it will still have the same frequency.

In order to determine the velocity of points in the Kinect image, we use a block matching algorithm [24]. This algorithm works by dividing up the image with a grid of blocks (we use \( 8 \times 8 \) blocks). Then, for each block, the algorithm estimates the optical flow to the next frame by searching in a small window about the centre of the block, for the best match in the next frame in a SAD (Sum of Absolute Differences) sense. An example of the optical flow obtained from this algorithm is shown in Fig. 2.

To detect its hand, the robot starts a detection episode by briefly holding its hand still, and then passing a repeated sine wave through its wrist actuator (the exploratory motion). Whilst the sine wave plays out of the wrist actuator, the Kinect records images of the hand, and optical flow is calculated for each recorded image. A small delay follows the end of the sine wave, to allow all of the motion to be recorded by the camera, and then the detection episode is concluded. The optical flow from all of the recorded

Fig. 1. The BERT robot observing its hand.

Fig. 2. An example of the optical flow obtained from the block matching algorithm, in response to an exploratory waving motion.
images is then concatenated, to give a time-dependent optical flow series for each block.

In order to identify the input sine wave in the output optical flow signal, we use Normalised Cross Correlation (NCC). This involves treating the input sine wave and the output optical flow as two random processes, X and Y, and then calculating the correlation coefficient \( \rho(X, Y) \), defined as

\[
\rho(X, Y) = \frac{\text{cov}(X, Y)}{\sqrt{\text{var}(X)\text{var}(Y)}},
\]

where \( \text{var}(\cdot) \) and \( \text{cov}(\cdot, \cdot) \) are variance and covariance, respectively. The correlation coefficient will be a value in the range \([-1, 1]\), with 1 signifying a perfectly matching pair of signals, -1 signifying that X is an inverted version of Y, and 0 signifying that the two signals are completely uncorrelated.

In order to detect the hand at a particular point in the Kinect image, we therefore calculate (5) between the optical flow and repeatedly delayed versions of the input signal. If the correlation coefficient goes above a given threshold \( \lambda \), for both the x and y components of the optical flow, then we identify the block as having motion that matches the exploratory wave. Fig. 3 illustrates a typical application of NCC to the optical flow of a block containing the hand.

Once NCC has been run on all blocks, the hand is declared to be the largest connected set of positively identified blocks, and a bounding box gives the expected extents of the hand. An example of this is shown in Fig. 4.

It is important to realise that with this technique, we are not tracking points on the robot’s hand as it waves, instead, we are looking at the instantaneous linear velocity which is observed at a grid of points on the image plane, and comparing it to the signal put into the wrist actuator. The analysis in this section shows that a sine wave will be preserved during the transform from joint space to image space. However, it will only be preserved perfectly if we are observing a solid disc rotating back and forth about the wrist actuator. This is because, if there are gaps in the disc, then the observed velocity will drop to 0 as the gaps pass in front of the observation point on the image plane.

The robot’s hand is obviously not a disc. However, we were still able to get good results with this technique by keeping the amplitude of the waves fairly small, so that during the waving motion, the long, thin, fingers of the hand spent most of their time close to a small group of observation points.

\[ x \leftarrow \text{LocateHandWithWave} \]

\[ \text{while } |x_c - x| > \epsilon \text{ do} \]

\[ \text{BuildJacobian} \]

\[ \Delta q \leftarrow J(q)^\dagger (x_c - x) \]

\[ x \leftarrow \text{LocateHandWithWave} \]

\end while
4.3. Building a model of the hand

The method we propose for building a model of the robot’s hand draws inspiration from the work presented in [10,11]. Essentially, we build and track a surfel model \( S = \{ s_1, s_2, \ldots, s_M \} \) where each surfel \( s_i \in S \) has position \( \mathbf{v}(s_i) \), normal \( \hat{n}(s_i) \), radius \( r(s_i) \), and colour \( c(s_i) \) attributes.

The concept of a surfel is illustrated in Fig. 5. Surfels allow the properties of an object’s surface to be captured, without having to maintain connectivity information between parts of the surface. Approximating a surfel as a hexagon allows it to be drawn quickly and efficiently, with two quads.

Following on from the previous two sections, the robot’s hand is now roughly centred in its field of view, and we have obtained a rough 2D bounding box, describing the image extents of the hand, by using an exploratory waving motion.

Prior to building a model of the hand, the pose of the robot and the corresponding joint angles at this stage are defined as the initial position for the hand model, and also for the subsequent kinematic model. To build the model of the hand, we convert the 2D bounding box in image space to a 3D bounding box in Kinect camera space. The \( x \) and \( y \) dimensions of the 3D bounding box are easily obtained from the 2D bounding box. The \( z \) dimension of the 3D bounding box is set by assuming that the depth of the hand will not be greater than the diagonal of the 2D bounding box. Once built, the 3D bounding box is used to filter out any 3D points which lie outside the box. Fig. 6 shows a box positioned around the points that represent the robot’s hand. All points outside the box have been filtered out. The box is moved along with the hand, as it is tracked in camera space, to filter each frame as it comes from the Kinect.

Once a point cloud has been filtered, we estimate normals for the remaining points by treating the cloud as a polygonal height map and setting the normal of each point to be the weighted average of the normals of the neighbouring 4 polygons. This is a simplistic method, which gives unreliable normals close to depth discontinuities. To address this, we remove the points close to depth discontinuities by treating the Kinect depth-map as a greyscale image, and applying morphological erosion. The filtering and normal estimation step gives a point cloud \( P = \{ p_1, p_2, \ldots, p_N \} \) where each point \( p_i \in P \) has a position \( \mathbf{v}(p_i) \), normal \( \hat{n}(p_i) \), and a colour \( c(p_i) \) attribute.

To initialise the model of the hand, we convert the first point cloud directly into a surfel model. So, for each point \( p_i \), we create a surfel \( s_i \) with the same position, normal, and colour as the point. The radius \( r(s_i) \) is set as

\[
r(s_i) = \frac{\mathbf{v}_i(p_i)}{\sqrt{2} \cos \gamma},
\]

where \( \mathbf{v}_i(p_i) \) is the depth of \( p_i \), \( l \) is the focal length of the Kinect, and \( \gamma \) is the angle between \( \hat{n}(p_i) \) and the normal to the image plane.

This sets the radius for a surfel so that it roughly covers one pixel when it is observed.

After that, the robot rotates its wrist joint through 180°, in order to turn the hand in front of the Kinect. The Kinect produces new point clouds roughly every 2.5°, and as each point cloud comes in, the existing model is aligned with the observed point cloud using the ICP algorithm.

4.3.1. Point cloud alignment with ICP

The process of finding a transformation which aligns the existing model with the observed point cloud involves repeatedly finding the closest point in the point cloud, for each surfel in the model. A naive algorithm for finding the closest point would have a runtime complexity of \( O(NM) \), which would slow the alignment process down to the point of making a real time implementation infeasible. Therefore, we make use of projective matching as detailed in [9], in order to get a quick approximation of the closest point for each surfel.

Projective matching works by first positioning the surfel model in the camera space of the observed point cloud, using the current estimate for the transformation between the model and point cloud. Then, the normal of each surfel is tested in turn, against the camera axis, to see if the surfel is visible. If it is, then the surfel is projected onto the image plane. If a point from the point cloud occupies the same pixel on the image plane, then the point is declared to be a match for the surfel. We cope with the case of multiple surfels matching the same point, by using a z-buffer, so that only the surfel closest to the image plane gets a match. This allows the matching to be done with a runtime complexity of \( O(M) \).

Once the matches have been found, we look for the transformation that minimises the point-to-plane metric [9]. This involves looking for the transformation that minimises the sum of squared distances between each surfel, and the plane on which its corresponding point lies. The orientation of a point’s plane is obtained from its normal, which was estimated above.

Formally, this means that, on each iteration, the optimal transform \( \mathbf{T}_{\text{opt}} \) between the surfel model and the point cloud is

\[
\mathbf{T}_{\text{opt}} = \arg \min_{\mathbf{T}} \sum_{i=1}^{M} \eta(i) \left( (\mathbf{T} \mathbf{v}(s_i) - \mathbf{v}(p_{m(i)})) \cdot \hat{n}(p_{m(i)}) \right)^2,
\]

where \( \eta(i) \) returns 1 if the surfel \( s_i \) has a matching point and 0 otherwise. Also, the function \( m(i) \) gives the index of the point that...
matches the surfel $s$. To minimise Eq. (9) we follow the lead of [27] and approximate the rotation part of $T_{opt}$ as a linear transformation. This approximation is reasonable, as there will only be a small difference in the pose of the hand from one frame to the next. It also enables us to minimise Eq. (9) efficiently, by allowing us to use linear optimisation, rather than non-linear optimisation.

### 4.3.2. Updating the surfel model

Once the surfel model has been aligned with the observed point cloud, the point cloud is merged into the surfel model, using a simplified form of the process presented in [10]. The process we use is to first find matches between surfels and points using the projective matching procedure given above. Then we either add new surfels to the model, or else update existing surfels.

New surfels are created from points which do not have a matching surfel in the model. When a point is matched to an existing surfel, we compare the depth of the surfel $\mathbf{v}(s_i)$ to the depth of the point $\mathbf{v}(p_{m(i)})$.

If $\|\mathbf{v}(s_i) - \mathbf{v}(p_{m(i)})\| < d_{\text{max}}$ where $d_{\text{max}}$ is a distance threshold (set to 10 mm in our system), then the observed point is far behind where the surfel predicts the surface should be. The surfel is assumed to be incorrect, deleted, and replaced with a new surfel created from the point.

If $\|\mathbf{v}(s_i) - \mathbf{v}(p_{m(i)})\| > d_{\text{max}}$ then the observed point is far in front of the surfel. This may not be a problem, as we may be viewing an occluding surface, so for now we ignore this case as a surfel may be added later on in response to a view of the surface from a different angle.

Finally, if $|\mathbf{v}(s_i) - \mathbf{v}(p_{m(i)})| < d_{\text{max}}$ then we assume that the point is a new observation of an existing surfel. The position and normal of the surfel is updated from the position and normal of the point using a running average. If the radius suggested by the point is smaller than the radius of the surfel (due to an observation being made closer to the camera) then the radius of the surfel is replaced. Also, if the normal of the point is closer to the camera axis than previous observations, we assume that we are getting a more direct view of the surface, and thus update the colour of the surfel as well. The process of averaging the point clouds into a surfel model has the effect of filtering out a lot of the noise, which would be present if the point clouds were used directly as the model.

### 4.3.3. Tracking the hand

After the model of the hand has been built, the system continues to track the hand by continually aligning the hand model against incoming point clouds. Fig. 7 shows a typical surfel model of the robot’s hand, built and tracked using this technique. We have produced GPU implementations for key parts of this model building and tracking system. As such, once the hand has been detected and centred, it takes just over a minute for the system to autonomously build a 3D model of the robot’s hand. It is then able to reliably give a reading for the position, and orientation of the hand, in camera space, at a rate of 15 Hz. Further evaluation of the accuracy and precision of the tracking system is presented in Section 5.

### 4.4. Building a kinematic model

Having built a 3D model of the robot’s hand, the tracking system is now used to build a kinematic model of the connecting links between the Kinect and the robot’s hand.

To build the kinematic model, the robot moves each of its joints back and forth through their range of motion in order to deduce their axis of rotation. The joints in our system are all revolute joints, but this system could also be used to identify prismatic joints. The order of the joints is given, but it could equally well be deduced by rotating sequences of joints in different permutations, and observing which joints have their axis moved by other joints. The position of a joint’s axis will not be affected by the position of a child joint’s axis, but it will be affected by the position of a parent joint’s axis. For the range of motion that the hand is visible to the camera, the motion of the hand can be tracked. Tracking the hand gives a rigid transformation, which describes how the hand model moves from the point where it is first visible, to the point where tracking is lost. To get useful information from this transformation, we make use of Charles’ theorem [28], which states

Any rigid transformation can be represented by a rotation about a fixed axis, followed or preceded by a translation along that axis.

We take advantage of this fact by deducing the screw decomposition of the tracked transformation. This gives us the direction of the joint axis, and a point on the joint axis. We also get the rotation around the axis, and the displacement along the axis. For a perfect revolute joint we should have only rotation around the axis; and likewise, a perfect prismatic joint should exhibit only displacement.

#### 4.4.1. Constructing a screw transformation

Before we tackle the problem of finding the screw decomposition, we look at how a transformation matrix $A$ would be constructed from $(\mathbf{s}, \mathbf{s}_n, \theta, k)$, where $\mathbf{s}$ is the axis normal, $\mathbf{s}_n$ is a point on the axis, $\theta$ is the angle rotated around the axis, and $k$ is the distance translated along the axis. Fig. 8 shows the component parts of a screw transformation. A screw transformation can
be constructed from simpler transformations, in the sense that it can be viewed as the composition of

1. a translation \(-s_o\) to move the screw axis so that it passes through the origin
2. a rotation \(\theta\) around the screw axis at the origin
3. a translation \(s_o\) to restore the position of the screw axis
4. a displacement \(k\) along the screw axis \(\hat{s}\).

This gives

\[
A = \begin{pmatrix} 1 & k\hat{s} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & s_o \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R_t(\theta) & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -s_o \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} R_t(\theta) & 0 \\ 0 & 1 \end{pmatrix} - (s_o - R_t(\theta)s_o + k\hat{s}). \tag{10} \]

Now \(R_t(\theta)\) can be built using Rodrigues’ rotation formula [29]

\[
R_t(\theta) = I + [\hat{s}]_x \sin \theta + [\hat{s}^2]_x (1 - \cos \theta) \tag{11} \]

where \([\hat{s}]_x\) is the skew symmetric matrix

\[
[\hat{s}]_x = \begin{pmatrix} 0 & -s_z & s_y \\ s_z & 0 & -s_x \\ -s_y & s_x & 0 \end{pmatrix}. \tag{12} \]

4.4.2. Obtaining the screw decomposition

Now, given a Euclidean transform

\[
A = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \tag{13} \]

we tackle the problem of deducing the screw decomposition. Matching the elements of (13) to those of (10) we have

\[
R = R_t(\theta) \tag{14} \]

\[
t = s_o - R_t(\theta)s_o + k\hat{s}. \tag{15} \]

As \(R\) is a rotation matrix, the axis of rotation \(\hat{s}\) is the eigenvector corresponding to the unit eigenvalue. Also, if (11) is expanded, it can be shown that \(\theta\) is easy to find as

\[
\text{trace}(A) = 1 + 2 \cos \theta. \tag{16} \]

Using the dot product we obtain \(k\), the distance travelled in the direction of \(\hat{s}:

\[
k = \hat{s} \cdot t. \tag{17} \]

Now, if \(\theta = 0\) we are unable to determine a value for \(s_o\), although if we were tracking a prismatic joint, we could use any of the tracked positions for \(s_o\). When \(\theta \neq 0\) we find \(s_o\) by creating the points \(x'\) and \(x''\) from an arbitrary point \(x\) which is not on the axis of rotation:

\[
x' = Ax - k\hat{s} \tag{18} \]

\[
x'' = Ax' - k\hat{s}. \tag{19} \]

This applies \(A\) to \(x\) and removes any translation in the direction of \(\hat{s}\) leaving just the rotation around \(\hat{s}\). The three points are used to construct intersecting planes as shown in Fig. 9, and using these planes we calculate \(s_o\) as a point where the planes intersect. For the special case of \(\theta = \pi\), \(s_o\) is set as the midpoint between \(x\) and \(x''\).

When implementing the screw decomposition, we ensured that \(x\) did not lie on the axis of rotation, by testing the distances between \(x\) and \(x'\) to ensure that they were greater than 0. A maximum of 3 non co-linear points need to be tested to guarantee that we find a suitable \(x\).

5. Validating the system

Our robot is able to locate its hand in its field of view using exploratory waves. It can then centre its hand in its field of view using a simplified form of visual servoing. and once centred, it can build and track a model of its own hand. Finally, it can also make use of the tracking ability to construct a kinematic model of its arm. We now present experiments which were carried out to assess the system performance.
5.1. Hand location and centring

We explore the reliability of our proposed method for locating and centring the hand, by starting the robot in 6 different poses, which give an initial position for the hand, at various points in the robot’s field of view. The hand location and centring process was run for 3 iterations, and at each step, the position of the hand was measured roughly, by manually picking out a fixed point on the hand.

Fig. 11 shows the result of this experiment. It can be seen that the process works well for a number of different starting points, with the hand brought within 20 pixels for all start points in 3 steps, and in some cases just 2 steps.

5.2. Hand tracking accuracy and precision

We assess the accuracy and precision of our tracking system using the standard definitions given in [30]. Accuracy is defined as the average absolute error when measuring known values. Precision is defined as the standard deviation of repeated measurements of a fixed value.

Precision was the easiest quantity to assess. We first confirmed that the measurements of the hand’s pose were normally distributed, using Pearson’s chi-squared test. Then, the robot’s hand was moved around a 3D grid of points in camera space (see Fig. 12), with the necessary joint angles being calculated using the constructed kinematic skeleton, and inverse kinematics. At each point, 10 measurements of the position and orientation of the hand were taken, and the precision at each point was taken as the standard deviation of the measurements.

The precision of the hand tracking was found to be good for a large number of the test positions, with the average translational precision, over all the test points, being in the order of 1 mm, and the average rotational precision being in the order of 0.1° (see Table 1).

The accuracy of the hand tracking was harder to assess because it requires knowledge of the hand’s true location, obtained from some more accurate measurement system. We had no easy way of measuring the position of the robot’s hand when it was being moved by the robot, so we made use of a spare robot hand, mounted it on a separate base, and moved it between 8 fixed test points on a table in front of the robot.

A coordinate system was defined on the table-top, and the positions of the test points were measured to the closest half millimetre with a ruler. The table-top was confirmed to be flat with a spirit level. The hand was moved between the test points, taking care to keep the orientation of the hand constant, and at each test point the position and orientation of the hand was measured by the hand tracking system.

It was not possible to compare the hand tracking measurements directly with the ruler measurements, because the coordinate systems were related by an unknown rigid transformation. However, as the orientation of the hand was held constant, it was possible to compare the distances between the test points. The distances between all pairs of test points measured by the hand tracking system were compared with the corresponding distances from the ruler measurements, and were found to agree with an average absolute error of just 2 mm, and standard deviation of 1.8 mm.

5.3. Assessing the hand model

In an attempt to quantitatively assess the quality of the hand model built by our modelling system, we detached the hand from the robot, and built a high resolution model of the hand using a 3DMD scanner [31] (see Fig. 13). Using the MeshLab program [32], the high resolution model was then aligned using ICP, with a model of the hand produced by our system. We then calculated the Hausdorff distance from the high resolution model, to our hand model, again using the implementation by MeshLab. Essentially, this samples a large number of points on the high resolution model, and measures the distance from each of those points, to the closest corresponding point on our hand model.

The Root Mean Square (RMS) of these distances was found to be 4 mm. Obviously, this only evaluates the accuracy of our hand model in terms of how well it agrees with the high resolution model. However, 3DMD quote their scanner as having a geometry accuracy of 0.2 mm RMS or better. Therefore, it seems reasonable to
We believe that the inaccuracies are due to a combination of errors in the measured positions of the joint axes, and unmodelled effects such as joint compliance.

6. Conclusion

With the advent of affordable depth cameras, it becomes far easier to give robots true 3D vision, which enables them to build detailed models of themselves, their environment and the objects that they interact with. In this paper, we have presented and evaluated a system in which a robot turns a depth camera on itself, and is therefore able to progressively build a model first of its hand, and then of the kinematic structure that links its camera to its hand.

The system is able to identify and locate the robot’s hand using exploratory motions, and therefore, it is able to build the 3D and kinematic models with very little prior information.

There is nothing new about kinematic identification per se [19,17], and other researchers have already programmed robots augmented with fiducial markers to build models of themselves [22,21]. However, the contribution of our work is to show how a depth camera can remove the need for fiducial markers, by building a model of the robot’s end effector, and tracking that directly. The use of exploratory motions in this context, to detect and centre the robot’s hand prior to model building, is also novel, and it reduces further the a priori knowledge which must be supplied to the robot by a system designer.

A system such as the one presented in this paper, would be of potential use for a robot designed for service in a domestic environment, or for an assisted living application. This is because such a robot may have to work for extended periods without being serviced, and so the ability to rebuild its kinematic model in response to a change or a degradation in its hardware could greatly improve the reliability of the robot.

The system would also be very useful in situations where a new kinematic model had to be built for a robot, but where it would be difficult, or impractical, for a human to gain access to take the required measurements. For example, this may be because the robot is in a remote or hazardous location.

One of the more exciting applications of this work is that it makes progress towards software that could dramatically reduce the time and effort required to commission a new robot. Rather than a group of system designers having to produce models specific to the new robot, an advanced software agent could instead be loaded onto the robot. It would then undertake a learning process to discover the capabilities of the robot, possibly mimicking the process that a human uses to learn the capabilities of its body when a child.

It is interesting to consider what might be the minimal amount of information, and set of sensors, that a robot would require in order to build a kinematic model of itself. For our BERT robot, the main problem is that it is not possible to make unconstrained exploratory motions with the robot. This is because for some possible motions, there is a risk that the robot’s arms and hands will collide with each other, or with the robot’s torso, damaging the robot. Therefore, we always need to provide our robot with a ‘safe’ range of angles that it can drive its actuators through. This requirement could be avoided if the robot was equipped with a touch sensitive skin, which allowed it to detect self collisions. Projects such as ROBOSKIN [33], show that the technology is readily available. Incorporating a sense of touch with the vision-based techniques presented here could provide a range of interesting options for the autonomous kinematic identification of robotic systems.

In future work, we intend to incorporate a kinematic calibration step into our system to refine the autonomously built kinematic model using methods such as those in [18]. We also hope to use the model building techniques presented in this paper to enable the robot to extend its perception beyond its own body, and to build models of objects it interacts with.
Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable comments during the preparation of this paper.

This work was supported by funding from the Leverhulme Trust, project number F/00182/C.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.robot.2013.09.011.

References


