A Simple but Useful Approach to Monocular Eye-in-Hand Robotic Orientation Calibration

Seth Hensley and Sebastian van Delden
Division of Mathematics and Computer Science
University of South Carolina Upstate
Spartanburg, SC 29303
{sehensley and svandelden}@uscupstate.edu

Abstract – In this paper we present a way to automatically recover camera orientation in an eye-in-hand system. The algorithm is completely automated performing a sequence of rotations and translations iteratively until the camera frame has been successfully aligned with the manipulators world frame. The system we have developed has been fully implemented and tested on a Staubli RX60 robotic arm using an off-the-shelf Logitech USB camera. The algorithms were developed in both the Java and V+ programming languages, which for our purposes needed to communicate together. In our tests we use vision algorithms to snap a series of pictures of a black blob on a white background, working to center the object. Data from these algorithms are processed using manipulator algorithms developed in V+. These data indicate what movements should be made by the end effector. These movements are made incrementally until the camera and the Robots world frame are aligned. In our experimental results the algorithms successfully converged for each test and the unknown angles were successfully recovered. Our experiments and results in this paper present a novel way to recover camera orientation during recalibration. The system presented in the following sections can provide an efficient way to automatically allow a manipulator to maintain precision throughout operation.

1. INTRODUCTION

Many robotic manipulators utilize cameras and vision algorithms to accomplish factory automation tasks. These cameras allow for a more flexible work environment by extracting key
features and important image regions from the manipulators work area, from which position and orientation information of a desired pose is determined. The vision algorithms used by a visually guided manipulator are usually very environment and problem oriented, that is, they are engineered to solve a very specific problem.

Even though closed-loop visually guided robotics([1]-[6] are a few of the papers in the literature that present good overviews of visually guided robotics research with a focus on closed loop systems, or “visual servoing”) is a popular area of research, many current industrial applications also employ calibrated open-loop systems. At Staublis Fast Moving Technology Days event in September 2006, several calibrated open-loop visually guided applications were being demonstrated by industry leaders. In one demonstration a vision system determined the location and orientation of a bag of potato chips moving down a conveyer belt, and then instructed the arm on how the bag needed to be picked using a suction cup gripper. In another demonstration, bolts that needed to be picked and placed lay randomly on a surface, possibly occluding each other. The vision algorithms would identify isolated bolts and then use a sequence of surface vibrations to alter the pose of occluded bolts, eventually isolating out single bolts and picking them. [10] is an excellent article that describes several other current vision-based object handling industrial applications in use.

In a calibrated system, the camera and robot kinematics are calibrated relative to a fixed 3D frame. The classical approach is to move the end-effector and observe/perceive the movement of the eye: or $AX = XB$, where $A$ is the robot end-effector motion $\tau^2$, $B$ the induced camera motion $\tau^c$, and $X$ is the hand-eye transformation $\tau^h$ to be determined.
In a visual servoing system, visual feedback is used to minimize the image plane error of the manipulator’s actual and desired positions. The vision system looks at the current pose of the manipulator and estimates how its joints should be moved so that the manipulator draws closer to the desired pose. Typical tasks like tracking and positioning are performed by reducing the image distance error between a set of current and desired image features in the image plane.

Our research makes two contributions to the field of visually guided robotics. The first contribution deals with a traditional problem that calibrated systems face [11]: over time the precision of the robot/camera coordinate system calibration degrades due to movement, vibration and other forces. This creates the need for the system to constantly be re-calibrated. Thus we present a completely automated re-calibration algorithm that recovers the orientation of the camera frame with respect to (w.r.t) the robot world frame.

The algorithms are designed for a monocular eye-in-hand system where the robot controller is capable of rotating and translating the tool frame w.r.t. the world frame. Since this method is completely automated, there is no need for a human operator to re-calibrate the system. The re-calibration procedure could be run periodically and automatically by the system to ensure that precise calibration is maintained.

The second contribution addresses how visually guided robotic systems are presented in the literature. Modern robotic manipulators are equipped with sophisticated programming environments, for example: Staubli’s Robotics Studio software and VAL3 programming language, FANUC’s Proficy, and ABBs Robot Application Builder. Modern robotic controllers are programmed with high level programming languages very similar to modern general purpose languages like C++ or Java. However, the literature on visually guided robotics manipulator
lacks contributions in which the methodologies are presented in an algorithmic fashion which would enable a robotic programmer in industry to easily reproduce the vision and control algorithms. Here we seek to partially fill this void by presenting the majority of our work in an algorithmic fashion that is more intuitive for robotics programmers to implement.

II. ASSUMPTIONS AND INITIALIZATIONS

The following assumptions and initializations are required by the algorithms in this paper:

- The manipulator must be able to translate and rotate its tool frame \{T\} w.r.t. its world frame \{W\}. As the location of the tool flange changes in times, the transformation \( t^W \) between points in \{T\} and \{W\} is automatically maintained internally by the robot controller.
- The camera must be mounted to the end effector. The pose of the camera frame \{C\} is not known w.r.t. \{T\} or \{W\}.
- There must be a flat surface in the robot work area that has a solid color background and a blob located in the center whose color contrasts greatly with the background, for example, a black blob on a white surface as shown in Figure 1. The blob does not necessarily have to be round.
- A pixel-value threshold that separates blob and background pixels must be determined. This could be manually done, but section A describes an algorithm that automates this initialization process.
- A rough alignment of the \{C\} w.r.t. \{W\} must be initially determined. The closest axis \( X_W \), \( Y_W \), and \( Z_W \) in \{W\} (within 45° or -45°) must be mapped to \( X_C \), \( Y_C \), and \( Z_C \) in \{C\}. For example, \( +X_C \) is within 45° or -45° to \( -Y_W \), \( +Y_C \) is within 45° or -45° to \( -X_W \), and \( +Z_C \) is within 45° or -45° to \( -Z_W \). This could be done manually, but section B describes an algorithm that automates this initialization process.
Finally, communication between the vision software and the controller must be established. Our software was written from scratch in Java and had to communicate with Stäubli’s V+ control language. To accomplish this, we used a TCP/IP communication program that we recently developed for another application [12].

A. Bi-Modal Image Thresholding

The camera must be positioned w.r.t this surface so that nothing else is in its field of view (FOV). Initially, the camera must be close enough to the surface so that the blob is imaged large enough (approximately 30-70% of the pixels in the image) to produce a true bi-modal histogram which is required for the automated thresholding algorithm to determine a correct threshold. This is a very important step which avoids the need to hard code a threshold and enables the rest of the algorithms to work even if lighting conditions fluctuate over time in the work cell.

![Image of robot and camera](image.png)

Fig. 1. The initial configuration of the robot and camera w.r.t to the blob on solid background. The orientation of the camera frame is not known w.r.t. the tool frame and will be recovered by the algorithms in this paper.

Figure 2 shows a greyscale input image that was captured by our eye-in-hand manipulator. Even if faint shadows are presents in the image, as in this input image, they will not have an impact on the thresholding algorithm.
Fig 2. Camera input view. The blob occupies approximately 30% of the pixels in the images in order to produce a true bimodal input image. Nothing else can be seen in the camera’s field of view.

We implemented the Otsu bimodal thresholding algorithm [13] which selects the threshold based on the minimization of the within-group variance of the two groups of pixels separated by the thresholding operator. By obtaining this threshold we are able to separate the foreground from the background in the image. Let $\sigma^2_1(t)$ be the variance for the group with values less than or equal to t and $\sigma^2_2(t)$ be the variance for the group with values greater than t. Let $q_1(t)$ be the probability for the group with values less than or equal to t and $q_2(t)$ be the probability for the group with values greater than t. Let $\mu_1(t)$ be the mean for the first group and $\mu_2(t)$ the mean for the second group. Then the within group variance $\sigma^2_w$ is defined by:

$$\sigma^2_w(t) = q_1(t)\sigma^2_1(t) + q_2(t)\sigma^2_2(t)$$

where

$$q_1(t) = \sum_{i=1}^{t} P(i)$$
$$q_2(t) = \sum_{i=t+1}^{n} P(i)$$
$$\mu_1(t) = \frac{\sum_{i=1}^{t} iP(i)}{q_1(t)}$$
$$\mu_2(t) = \frac{\sum_{i=t+1}^{n} iP(i)}{q_2(t)}$$
Each potential pixel threshold value (usually in the range [0-255] for a typical greyscale image) is plugged into these formulas and the value that minimizes the within-group variance is chosen as the threshold. These formulas are easily implemented and will be not presented here in algorithmic fashion. Figure 3 shows the histogram created from the input image in Figure 2 and threshold value of 124 that was recovered by the above equations.

![Figure 3](image.png)

Fig 3. The histogram that was produce from the input image in Figure 2. The horizontal axis represents pixel values [0…255] and the vertical axis represents pixel quantity.

The camera must be relatively close to the blob in order to initially determine the threshold, given that a bi-modal histogram was needed. However, after the threshold has been determined, the camera can be moved away from the surface, causing the blob to shrink, and the initial threshold will still be valid. This capability is required for the algorithms in section III.

**B. Initial Rough Alignment**

An initial rough alignment of the robot world \{W\} and camera \{C\} frame axes must be determined so that an approximate correlation between movements in the robot world frame and the blob can be established. Figure 4 depicts an initial rough alignment that was used in our experiments.
Fig 4. An example initial alignment of camera and robot frames. The unknown angle differences are recovered by the algorithms in the following sections.

Note that the $+Y_C$ axis is inverted since the camera origin is in the upper left corner of the image and row number increases as you move down through the image. The angle differences between camera and robot axes are unknown and will be recovered by the algorithms in the following sections.

The initial axes correlations are recovered by moving the end-effector (and thus the camera) along each of the robot’s world axes and observing the greatest blob centroid change in the camera coordinate system. For example, in Figure 4, a translation of the end-effector in $+Y_W$ resulted in maximum blob centroid movement along $-X_c$.

The initial alignment is determined as follows:
- Translate some distance in $+X_W$, $+Y_W$, and $+Z_W$.
  - The distance is arbitrary, but the blob should not move out of the FOV of the camera.
- Note blob centroid movement in $X_c$ and $Y_c$ after each translation.
  - Each translation results in $X_c$ and $Y_c$ blob centroid movements (six values in total).
- The top two blob movements in Xc and Yc indicate the alignment of two of the robot axes, and the third alignment can then be automatically determined.

### III. MANIPULATOR CONTROL ALGORITHMS

The vision algorithms communicate with the manipulator control algorithms by sending a three tuple of information that indicates what type of incremental movement should be made to the end-effector: \((\{\text{rotation, translation}\}, \{X_W, Y_W, Z_W \text{ axis}, \{\text{positive or negative decimal number}\}\})\). The positive or negative integer indicates the direction of the translation or rotation, and also how many mm or degrees should be moved. Only incremental movements are made until the camera and robot world frames are aligned, avoiding the need to determine a mm per pixel relationship which would be application specific.

The algorithm is summarized below in a V+ type syntax which is used by Stäubli RX series manipulators.

```plaintext
WHILE (NOT ALIGNED) DO
  (TYPE, AXIS, VALUE) \(<\) THREE TUPLE
  CUR_POS \(<\) CURRENT END-EFFECTOR POSITION
  \{CUR_X, CUR_Y, CUR_Z, CUR_YAW, CUR_PITCH, CUR_ROLL\} \(<\) DECOMPOSE(CUR_POS)
  IF (TYPE == TRANSLATE) THEN
    IF (AXIS == X) THEN
      MOVE TRANS(VALUE, 0, 0, 0, 0, 0): CUR_POS
    END
    ELSE IF (AXIS == Y) THEN
      MOVE TRANS(0, VALUE, 0, 0, 0, 0): CUR_POS
    END
    ELSE (IF AXIS == Z) THEN
      MOVE TRANS(VALUE, 0, 0, 0, 0, 0): CUR_POS
    END
  END
  IF (TYPE == ROTATE) THEN
    IF (AXIS == X) THEN
      MOVE TRANS(CUR_X, CUR_Y, CUR_Z): RX(VALUE):
      TRANS(0, 0, 0, CUR_YAW, CUR_PITCH, CUR_ROLL)
    END
    IF (AXIS == Y) THEN
      MOVE TRANS(CUR_X, CUR_Y, CUR_Z): RY(VALUE):
      TRANS(0, 0, 0, CUR_YAW, CUR_PITCH, CUR_ROLL)
    END
    IF (AXIS == Z) THEN
      MOVE TRANS(CUR_X, CUR_Y, CUR_Z): RZ(VALUE):
      TRANS(0, 0, 0, CUR_YAW, CUR_PITCH, CUR_ROLL)
    END
  END
END
```

The control algorithm receives the three tuple of information and makes the movement relative to the current location of its end-effector. Each time a movement is made, the location of the
end-effector must be updated. Communication and movement must be synchronized so that the robot completes its current motion before the vision algorithms compute that next motion.

The DECOMPOSE function recovers the X, Y, Z, Yaw, Pitch, and Roll values of CUR_POS, the current location of the end-effector. This is significant for end-effector rotations because the rotation must be made w.r.t. to {W} and not {T}. The X, Y, and Z components are extracted from CUR_POS and then combined with the robot’s world Yaw, Pitch, and Roll values so that rotations are centered around the translation values of CUR_POS but are made around the world axes. The trans(X,Y,Z,Yaw,Pitch,Roll) function returns a transformation created from its parameters. The RX(p), RY(p), and RZ(p) functions returns pure rotation transformations of p degrees around the world X, Y, and Z axes, respectively. A colon denotes transformation multiplication.

Notice for the rotations portion of the algorithm, that a pure translation transformation is first created from the end-effector’s X, Y, and Z values. The yaw, pitch and roll values of this transformation are equal to the world frame. This transformation is then multiplied by a pure rotation transformation around the desired world axis. Finally, the result is multiplied by a pure rotation transformation created from the original Yaw, Pitch and Roll from CUR_POS. This ordering is essential for rotating the end-effector around {W} and not {T}. Using the X, Y, and Z components from CUR_POS ensures that the rotation is made from a point close to the camera which will prevent a large end-effector movement which could move the blob out of the camera’s FOV.
III. VISION ALGORITHMS

A. Centering

We need a mechanism for constantly centering the blob in the image in order to ensure that the blob remains in the cameras FOV. The algorithm locates the image quadrant, where the centroid of the blob is located, and then incrementally translates in the appropriate direction until the blob is centered.

Fig. 5. A depiction of blob movements during the centering process.

In the centering algorithm, IMG_CENTER refers to the pixel center of the image and BLOB_CENTER refers to the centroid of the blob. X_TRANS and Y_TRANS are small user defined mm distances that the robot should translate in the X_C and Y_C directions, respectively, and the sign indicates the direction along that axis. The mapping function returns the corresponding world axis that its parameter has been mapped to during the initialization step.

```plaintext
WHILE (|BLOB_CENTER - IMG_CENTER|>MIN_ERROR) DO
  IF (CENTROID.X > IMG_CENTER.X) THEN
    VALUE = -X_TRANS
  ELSE
    VALUE = +X_TRANS
  END
  SEND(TRANSLATE, MAPPING(X_C), -X_TRANS)

  IF (CENTROID.Y > IMG_CENTER.Y) THEN
    VALUE = -Y_TRANS
  ELSE
    VALUE = +Y_TRANS
  END
  SEND(TRANSLATE, MAPPING(Y_C), -Y_TRANS)
END
```
B. Orientation Recovery

Three separate rotation steps are needed in recovering the orientation of the camera frame w.r.t. the robot frame.

- **First**, move back and forth along \( \text{MAPPING}(X_C) \), note the movement of the blob, and then incrementally rotate around \( \text{MAPPING}(Z_C) \) until the centroid row error is minimized.
  
  o This aligns \( X_C \) to the plane created by \( \text{MAPPING}(X_C) \) and \( \text{MAPPING}(Z_C) \) axes.

- **Second**, move back and forth along \( \text{MAPPING}(Z_C) \) direction, note the movement of the blob, and then incrementally rotate around \( \text{MAPPING}(X_C) \) until centroid row error is minimized.

  o This aligns \( Y_C \) perfectly with \( \text{MAPPING}(Y_C) \).

- **Third**, move back and forth along \( \text{MAPPING}(Z_C) \), note the movement of the blob, and then incrementally rotate around \( \text{MAPPING}(Y_C) \) until centroid column error is minimized.

  o This results in all three camera axes being aligned with their corresponding world axes.

The following algorithm shows how the first step can be implemented. The solutions for the second and third steps are very similar to this algorithm and so are not shown here.

DISTANCE is some arbitrary mm distance that cannot be too large which would cause the blob to go outside of the camera’s FOV. DEGREES is set to a small rotation value. In our experiments, it was set at \( \tfrac{1}{2}^\circ \); The sign preceding it indicates whether or not a positive or negative rotation should be performed. This exact order of the rotations as explained here is not necessarily required. The requirement, of course, is a sequence of three Euler angle rotations to recover the three angles. MIN_ERROR is an integer corresponding to the pixel error that we are willing to tolerate. Due to rounding errors when calculating blob centroids, a MIN_ERROR of
zero may not be possible. In our experiments, we tolerated a pixel error of one. The mm distance of this error value depends on the distance of the surface area from the blob and so will vary across implementations.

\[
\text{WHILE (NOT ALIGNED) DO} \\
\text{CENTER_BLOB( )} \\
(R1, C1) \leftarrow \text{INITIAL_CENTROID( )} \\
\text{SEND (TRANSLATE, MAPPING(X), DISTANCE)} \\
(R2, C2) \leftarrow \text{NEW_CENTROID( )} \\
\text{IF ( } |R2-R1| < \text{MIN_ERROR} \text{) THEN} \\
\text{ALIGNED = TRUE} \\
\text{ELSE} \\
\text{IF ( } R2 < R1 \text{ ) THEN} \\
\text{VALUE = } -\text{DEGREES;} \\
\text{ELSE} \\
\text{VALUE = } +\text{DEGREES;} \\
\text{END} \\
\text{SEND (ROTATE, MAPPING(Z), VALUE)} \\
\text{SEND (TRANSLATE, MAPPING(X), DISTANCE)} \\
\text{END}
\]

IV. RESULTS

We have implemented and tested the algorithms described in the previous sections on a Stäubli RX60 robotic manipulator which is controlled by the V+ programming language. The vision algorithms were written in Java and use the Java Media Framework (JMF) API to communicate with the camera, an off-the-shelf Logitech USB camera. We chose ten random starting configurations and then executed the algorithms. The algorithms converged for each test case and the unknown angle offsets were always correctly recovered. The rotation algorithms iteratively recovered the unknown angles in a linear fashion, so convergence speed of the algorithm varied depending on the size of the angles. We are currently working to improve convergence speed however.

One of the initial configurations consisted of the following unknown angles which were correctly recovered:

**Mapping:**

<table>
<thead>
<tr>
<th>Mapping</th>
<th>Initial Offsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAPPING(ZC) = −ZW</td>
<td>20.5°</td>
</tr>
<tr>
<td>MAPPING(YC) = −XW</td>
<td>8.5°</td>
</tr>
<tr>
<td>MAPPING(XC) = −YW</td>
<td>7.5°</td>
</tr>
</tbody>
</table>
Figure 6 shows the input images from this test case before the algorithms were executed. The upper leftmost image shows the blob centered and the lower leftmost shows the input image after a translation along $Z_W$. Notice that the blob moves towards the upper right corner of the image since $Z_W$ and $Z_C$ are not aligned. The other two pairs of images show movements along $Y_W$ and $X_W$ and the corresponding blob movements are not coincident to any camera axis.

Fig. 6. Test example initial configuration. Movements along world axes do not correspond to movements along camera axes.

Figure 7 shows the same sequences of input images after the algorithms have recovered the unknown angles. Notice now that a translation along $Z_W$ causes the blob to stay in place while it shrinks. Also, translations along $Y_W$ and $X_W$ cause perfect horizontal and vertical blob movements in the input image, as expected.
V. CONCLUSIONS

We’ve presented a completely automated algorithm to recover the camera orientation during recalibration of an eye-in-hand manipulator. These algorithms that we’ve presented could be executed by the manipulator periodically in order to maintain precise calibration over time. As it is currently, the algorithms require that a blob is placed on a solid color background surface in a clear workspace so that no other objects affect the cameras FOV. This work can eventually be extended to recover not only the orientation but also the translation offsets. We are also attempting to implement depth extraction [14]. We also wish to adjust the algorithm so that a blob is no longer required initially. Convergence speed is also another improvement that is being sought. This work provides a way to bypass tedious recalibrations on the operators part, in a way that is relatively quick in terms of overall time and maintenance, with non-specific off-the-shelf parts.

ACKNOWLEDGEMENTS

We would like to express our sincere thanks to the Stäubli Corporation for making this research possible by generously donating six RX60 manipulators to our institution, and for providing the Stäubli Robotics Studio software package to us which was used to create the 3D figures in this paper.
A similar version of this paper has been published in the 5th IEEE International Workshop on Robotic and Sensors Environments [15].

REFERENCES


