A High Performance Robot Vision Algorithm Implemented in Python

S. Chris Colbert\textsuperscript{1}, Gregor Franz\textsuperscript{2}, Konrad Wöllhaf\textsuperscript{2},
Redwan Alqasemi\textsuperscript{1}, Rajiv Dubey\textsuperscript{1}

\textsuperscript{1}Department of Mechanical Engineering, University of South Florida
\textsuperscript{2}University of Applied Sciences Ravensburg-Weingarten, Germany

SciPy 2010
Why the Need for Autonomous Robot Vision?

- It has broad applicability.
  - Industrial automation
  - Nuclear waste handling
  - Assistive and service oriented robots

- According to a 2006 US Census Bureau report, 51.2 million Americans suffer from some form of disability and 10.7 million of them are unable to independently perform activities of daily living (ADL).
  - Assistive robots can help with this.
Autonomous object recognition algorithms come in two forms:

- *A priori* knowledge based
- Novelty based
The State of the Art

How do the *a priori* knowledge based systems work?

- It starts with a model(s).
- 3D models, images, features...
- Match object against the database.
- Retrieve information.

**Example: Schlemmer et al.**

1. Store shape and appearance.
2. Find shape in range data.
3. Match appearance data.
4. Grasp via visual servoing and matched SIFT points.
How do novelty based systems work?

- With lots, *and lots* of data.
- Stereo, shape-from-silhouettes, laser ranger.
- Full access to object typically required.
- Long computation times.

**Example: Yamazaki et al.**

1. Drive robot around object.
2. Capture more than 130 images.
3. Perform dense disparity reconstruction.
4. Wait around 100s for the computed results.
Objectives

- Reconstruct the shape and pose of a novel object to a sufficient degree of accuracy such that it permits grasp and manipulation planning.
- Require no *a priori* knowledge of the object with the exception that a given object is the object of interest.
- Require only a minimal number of images for reconstruction; significantly less than the status quo.
- Operate efficiently, such that the computation time is negligible in comparison to image capture times.
Algorithm Overview

The algorithm has three main phases:

1. Capture three images of the object and generate a silhouette of the object for each image.
2. Use the silhouettes to generate a point cloud that approximates the surface of the object.
3. Improve the approximation by fitting a parametrized shape to the points. The parameters of this shape serve as the model of the object.
Three images are captured from disparate locations.

Two frontal, one overhead.

Mutually orthogonal is preferred.

But, this can be relaxed due to kinematic constraints.

Store reprojection matrix $C^W_T$ for each image.
Image distortion is corrected according the equations

\[
\begin{align*}
x_p &= (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) x_d + (2p_1 x_d y_d + p_2 (r^2 + 2x_d^2)) \\
y_p &= (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) y_d + (2p_2 x_d y_d + p_1 (r^2 + 2y_d^2))
\end{align*}
\]

Where \((x_d, y_d)\) are the distorted image points and \((k_1, k_2, k_3, p_1, p_2)\) are the five distortion coefficients.
Color based segmentation is used to generate the silhouette of object from each image.
Once the images have been captured and preprocessed, the 3D surface of the object is approximated in the form a point cloud. This comprises two major steps:

1. Create a sphere of points that completely bounds the object.
   - Requires the calculation of a centroid and radius.

2. Modify the position of each point such that the projection of the point intersects or lies on the edge of the silhouette.
Bounding Sphere Construction
Finding the Centroid

1. For each silhouette, project a ray from the camera center through the imaged silhouette centroid.
2. Find the single point which minimizes the sum of squared distances to each ray.
3. This point is the approximate centroid of the object and is used as the centroid of the sphere.
For each silhouette, find $r_{max}$ as shown in the figure.

Select the silhouette with the largest $r_{max}$ for further processing.
Bounding Sphere Construction
Finding the Radius

1. Project two rays from the camera center: one through the centroid, and one through \( r_{\text{max}} \).

2. Construct a plane at the centroid that is perpendicular to the centroid ray.

3. Find the point that lies on this plane and the ray containing \( r_{\text{max}} \).

\[
\begin{align*}
p_4 &= p_3 + (p_3 - p_1)t \\
t &= \frac{-(p_1 - p_2) \cdot (p_3 - p_2)}{(p_1 - p_2) \cdot (p_3 - p_1)}
\end{align*}
\]
Example: A simulated cylinder bounded by the computed sphere.

- Points are generated using a simple routine based on the golden ratio.
- The radius of the sphere is generally increased by a factor to insure complete bounding of the object.
Point Cloud Manipulation

1. Project $x_i$ into the silhouette image to get $x_i'$.
2. If $x_i'$ intersects the silhouette, do nothing.
3. Find the pixel point $p'$
4. Let the line $c_0p$ be $L_1$.
5. Let the line $x_0x_i$ be $L_2$.
6. Let $x_{i_{new}}$ be the point of intersection of lines $L_1$ and $L_2$.

$x_0 = $ Sphere Center
$C_0 = $ Cam Center
$x_i = $ Sphere Point
$x_{i_{new}} = $ modified $x_i$
Point Cloud Manipulation

Results

- The procedure is applied to each point \textit{once} in each silhouette image.
- A significant improvement over other algorithms.
- The result is a rough approximation of the objects surface.

- Given an infinite number of images, the approximation would converge to the \textit{visual hull}.
- Rather than more images, we improve the approximation by fitting a superquadric to the point.
The final phase of the algorithm is to find a shape that best fits the point cloud. We use superquadrics as our modeling tool for a variety of reasons:

- They have a convenient parametrized form which can be directly used for grasp planning.
- Their closed form expression provides a nice base for non-linear minimization.
- Their nature makes them robust to small sources of error.
- They are capable of accurately approximating the shape of many objects used in Activities of Daily Living.
Superquadrics
Some Possible Shapes
Implicit Superquadric Equation

\[
F(x_w, y_w, z_w) = \left( \frac{n_x x_w + n_y y_w + n_z z_w - p_x n_x - p_y n_y - p_z n_z}{a_1} \right)^{\frac{2}{\epsilon_2}} + \\
+ \left( \frac{o_x x_w + o_y y_w + o_z z_w - p_x o_x - p_y o_y - p_z o_z}{a_2} \right)^{\frac{2}{\epsilon_2}} \frac{\epsilon_2}{\epsilon_1} + \\
+ \left( \frac{a_x x_w + a_y y_w + a_z z_w - p_x a_x - p_y a_y - p_z a_z}{a_3} \right)^{\frac{1}{\epsilon_1}} \frac{\epsilon_1}{\epsilon_2}
\]

Evaluates to 1 if a point \((x_w, y_w, z_w)\) lies on the superquadric. \(F(x_w, y_w, z_w)\) is also called the inside-outside function.

17 parameters at first glance; 6 are redundant.
Superquadrics

The parameters \((n_x, n_y, n_z, o_x, o_y, o_z, a_x, a_y, a_z, p_x, p_y, p_z)\) make up the 4x4 transformation matrix that relates the superquadric centered coordinate system to the world coordinate system.

\[
W_Q^T = \begin{bmatrix}
  n_x & o_x & a_x & p_x \\
  n_y & o_y & a_y & p_y \\
  n_z & o_z & a_z & p_z \\
  0 & 0 & 0 & 1
\end{bmatrix}
\]

The 3x3 rotation portion is orthonormal and can be decomposed into the ZYZ-Euler angles \((\phi, \theta, \psi)\).

Thus, the superquadric is parametrized by 11 parameters

\[
\Lambda = (\lambda_1, \lambda_2, ... \lambda_{11}) = (a_1, a_2, a_3, \epsilon_1, \epsilon_2, \phi, \theta, \psi, p_x, p_y, p_z)
\]
Superquadrics

What do the 11 parameters represent?

- \((a_1, a_2, a_3)\) are the dimensions of the superquadric in the \(x\), \(y\), and \(z\) directions.
- \((\epsilon_1, \epsilon_2)\) are the shape exponentials.
- \((\phi, \theta, \psi)\) are the ZYZ-Euler angles which define orientation.
- \((p_x, p_y, p_z)\) are the \((x, y, z)\) world coordinates of the centroid of the superquadric.
Classical Cost Function

Inside-Outside function

\[ F = F(x_w, y_w, z_w, \lambda_1, \lambda_2, \ldots, \lambda_{11}) \]

Cost function

\[
\min_{\Lambda} \sum_{i=1}^{n} (\sqrt{\lambda_1 \lambda_2 \lambda_3} (F^{\epsilon_1} - 1)^2)
\]

- Standard cost function as derived by Jacklic, Leonardis, and Solina.
- \( \sqrt{\lambda_1 \lambda_2 \lambda_3} \) recovers smallest superquadric.
- \( \epsilon_1 \) exponential promotes rapid and robust convergence.
- Use non-linear gradient descent algorithm to find \( \Lambda \).
- There are limits placed on certain \( \lambda_i \) to restrict the range of recoverable shapes.
Error Rejecting Cost Function

Modified Cost function

$$\min_{\Lambda} \left[ w \sum_{i=1}^{n} (\sqrt{\lambda_1 \lambda_2 \lambda_3} (F_{\epsilon_1} - 1))^2 + \right.$$

$$\left. \left( (1 - w) \sum_{i=1}^{n} (\sqrt{\lambda_1 \lambda_2 \lambda_3} (F_{\epsilon_1} - 1))^2 \in F_{\epsilon_1} < 1 \right) \right]$$

- Penalizes points that lie inside the superquadric.
- Forces the superquadric to reject perspective projection errors.
- The superquadric will be as large as possible, without exceeding the visual hull.
- Empirically determined $w = 0.2$. 
Cost Function Comparison

Surface Approximation

Standard Cost Function

Modified Cost Function

↑ 20% greater accuracy
Example Reconstruction
Example Reconstruction
Simulate Trials
Overview

- Tested the algorithm against simulated sphere, cylinder, prism, and cube.
- These shapes represent a range of common convex shapes and can be modeled accurately by a superquadric.
- The results are reported by comparing recovered superquadric parameters against the known ground truth.
- The volume of the superquadric is also compared against the volume of the object in the form of a fraction.
- The volume fraction $v_f$ is a quick and intuitive measure of accuracy.
Simulation Trials

Results

$v_f = 1.087$

$v_f = 1.088$

$v_f = 1.077$

$v_f = 1.092$
Hardware Setup

Robot

- Kuka KR 6/2
- Six axis, low payload, industrial manipulator.
- High repeatability: $\pm 0.1\,mm$
Due to kinematic limitations of the robot, viewing directions were not perfectly orthogonal, but they approached such a condition.
Four test objects, all red in color, which represent a range of frequently encountered shapes.
Several sources of error are introduced in the hardware implementation that are not present in the simulation environment:

- Imprecise camera calibration: intrinsics and extrinsics
- Robot kinematic uncertainty
- Imperfect segmentation
- Ground truth measurement uncertainty (must be measured with the robot).
Experimental Trials
Battery Box

\[ v_f = 1.18 \]
Experimental Trials
Cup Stack

\[ v_f = 1.13 \]
Experimental Trials

Yarn Ball

$v_f = 1.14$
Experimental Trials
Cardinal Statue

\[ v_f = \frac{N}{A} \]
Performance Evaluation

With respect to the stated objectives:

- Most parameters of the reconstruction differ from the ground truth by no more than a few percent. This should be well within the margin of error for most household retrieval tasks.
  - Contrast this with an error of 10% in the work by Yamazaki et al.

- On an Intel QX-9300 at 2.53 GHz, the algorithm executes in ~0.3 seconds on average. This time depends highly on the time required for the non-linear minimization routine to converge.
  - The work by Yamazaki et al. required in excess of 100 seconds to converge.
Python Implementation

Core Algorithm

- **NumPy**
  - Basic image data structure
  - Linear algebra

- **Cython**
  - Image processing and superquadric gradient

- **Scikits.Image**
  - OpenCV bindings

- **SciPy**
  - Non-linear minimization
Python Implementation

Simulation

• Mayavi
  - Simulator engine
  - 3D renderings

• Traits UI
  - Simulator UI
Python Implementation

Hardware Networking

- **OpenOPC**
  - Robot control and communication

- **xmlrpclib**
  - Exposes core algorithm to the network

- **PIL**
  - Camera JPEG -> NumPy conversion
This work has presented an algorithm for the shape and pose reconstruction of novel objects using just three images.

The algorithm requires fewer images than other algorithms in the published literature, and provide sufficient accuracy for grasp and manipulation planning.

The algorithm provides higher performance than the other algorithms in the published literature.

The algorithm is entirely implemented in Python using libraries such as NumPy, SciPy, Cython, Mayavi, Traits, OpenOPC, and PIL.