# Matching 2-D Ellipses to 3-D Circles with Application to Vehicle Pose Detection

Marcus Hutter and Nathan Brewer

# Introduction

- What are we Doing?
- Detecting and Identifying Wheels
- Mapping 3-D circles to 2-D Ellipses
- Determining Pose

# What?

- We want to find the pose of a given 3D model of a vehicle that will match the pose of a similar vehicle in an image

# How?

- We do this by extracting information from the image that gives us clues about the location and orientation of the car in 3d space

# How?

- What Information?







# What do we know about wheels?

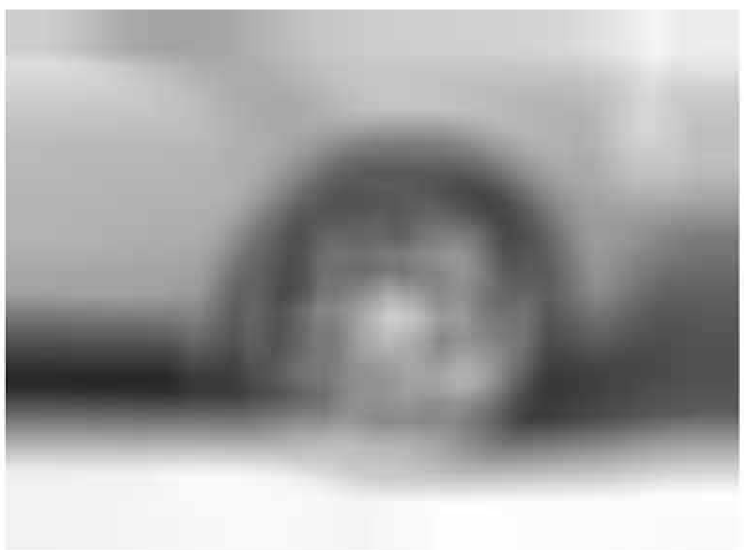
- Wheels are always circular in the real world, and elliptical in images
- Wheels are generally imaged as bright ellipses within a dark tyre

# Wheel Detection Algorithm

1. Generate Local Average Image
2. Normalize Image by Removing Average
3. Threshold Normalized Image
4. Find Connected Regions
5. Star Fill Connected Regions
6. Extract Ellipse Parameters from Blobs
7. Filter out Non-elliptical Blobs
8. Identify Wheels
9. Determine Wheel Normal

# Finding Comparatively Bright Areas (1&2)

- To find wheels, we first manipulate our image to find which areas appear brighter than their local surroundings.



Average of surrounding area

Difference between average and actual value

# Wheel Detection Algorithm

1. Generate Local Average Image
2. Normalize Image by Removing Average
3. Threshold Normalized Image
4. Find Connected Regions
5. Star Fill Connected Regions
6. Extract Ellipse Parameters from Blobs
7. Filter out Non-elliptical Blobs
8. Identify Wheels
9. Determine Wheel Normal

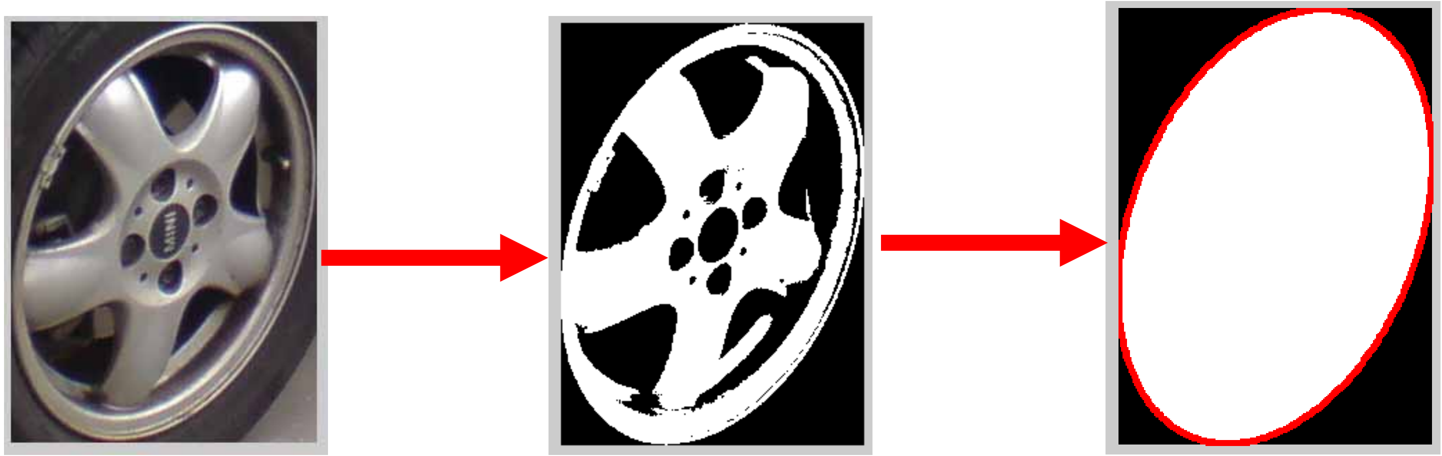
# Extracting bright regions (3&4)

- Given the comparative brightness of each object in the image, we extract and label only the brightest areas

# Wheel Detection Algorithm

1. Generate Local Average Image
2. Normalize Image by Removing Average
3. Threshold Normalized Image
4. Find Connected Regions
5. Star Fill Connected Regions
6. Extract Ellipse Parameters from Blobs
7. Filter out Non-elliptical Blobs
8. Identify Wheels
9. Determine Wheel Normal

# Filling Objects (5)



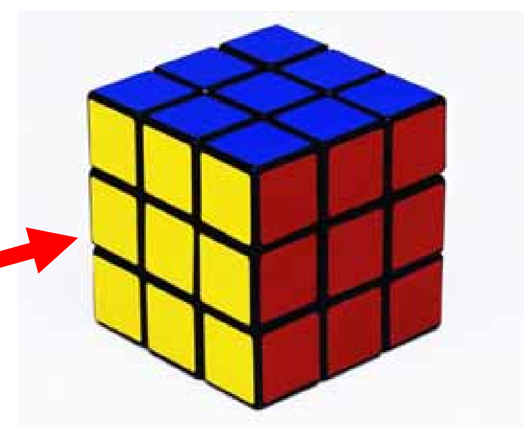
- We need to fill our objects, turning them into solid blobs
- Flood fill doesn't work, as there are often gaps, so instead we fill from each perimeter pixel to the centre

# Wheel Detection Algorithm

1. Generate Local Average Image
2. Normalize Image by Removing Average
3. Threshold Normalized Image
4. Find Connected Regions
5. Star Fill Connected Regions
6. Extract Ellipse Parameters from Blobs
7. Filter out Non-elliptical Blobs
8. Identify Wheels
9. Determine Wheel Normal

# Extracting Properties of Objects (6&7)

- We find the covariance matrix, *C*, corresponding to these filled objects
- We can then find the equivalent ellipse, which shares this covariance
- Comparing this ellipse to the object lets us determine how elliptical it is

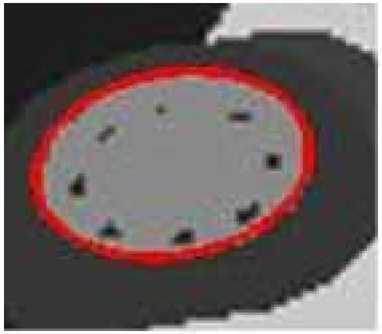
# Wheel Detection Algorithm

1. Generate Local Average Image
2. Normalize Image by Removing Average
3. Threshold Normalized Image
4. Find Connected Regions
5. Star Fill Connected Regions
6. Extract Ellipse Parameters from Blobs
7. Filter out Non-elliptical Blobs
8. Identify Wheels
9. Determine Wheel Normal

# Finding Most Probable Wheels (8)

- We are then left some elliptical objects, of which two correspond to the wheels
- Knowledge of where wheels lie on a car allows us to choose the most probable wheels

# Experimental Results





# Mapping a 2D Ellipse to a 3D circle

- To complete our algorithm, we need to find the 3D normal of a circle which would project to this ellipse

$$E(\mu, C) = \{ p \in R^2 : (p-\mu)^{\mathrm{T}} C^{-1} (p-\mu) \leq 4 \}$$

$$\text{Circle} = \{ x \in R^3 : \|x\| \leq r \; \& \; x^{\mathrm{T}} \varphi = 0 \}$$

# Mapping a 2D Ellipse to a 3D circle

- From the ellipse covariance matrix, we can determine an ellipse normal direction, which in this case corresponds to the direction of the axle with respect to the wheel

$$\boldsymbol{\varphi} \equiv \begin{pmatrix} \varphi_x \\ \varphi_y \\ \varphi_z \end{pmatrix} = \frac{1}{a_1} \begin{pmatrix} \pm\sqrt{a_1^2 - 4C_{xx}} \\ \pm\sqrt{a_1^2 - 4C_{yy}} \\ \pm a_2 \end{pmatrix}$$

# Determining the mapping from a Circle to an Ellipse

- We now want to find the projection that maps a general circle in 3D to a general ellipse in 2D
- Let $v, \phi \in R^3$ and R>0 be the centre, normal and radius of the circle to be projected
- Let μ and *C* be the centre and covariance matrix of the ellipse
- Finally, let $x' = \sigma Q x + q$ be the projection to be determined

# Determining the mapping from a Circle to an Ellipse

- To find this projection, we must find each component:

$Q \in R^{2\times 3}$: A rotation parameter, given by the first two rows of an orthogonal matrix $\tilde{Q}$

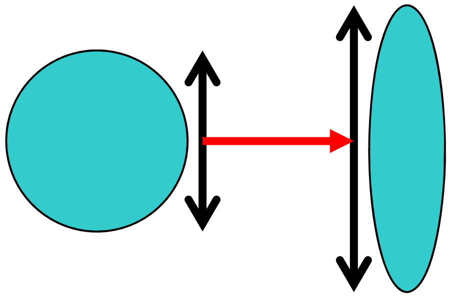
σ>0: A scale factor

$q \in R^2$: A shift vector

# Scale

- The relationship between the radius of the circle and the major axis of the ellipse is simple to calculate.

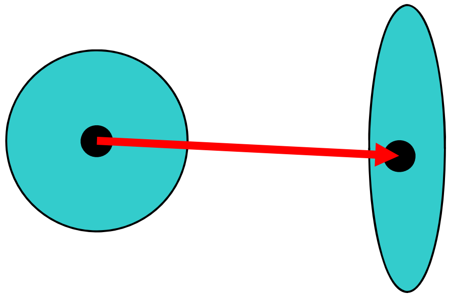
$$\sigma = \frac{a_1}{R}$$



# Shift

- We know that the centres of the ellipse and the projected circle must match, hence:

$$\mathbf{q} = \boldsymbol{\mu} - \sigma Q \mathbf{v}$$



# Rotation

- Let $\varphi$ be the normal of the ellipse, as extracted earlier
- We then know that our transformation must rotate $\Phi$ to align with $\varphi$, giving us the constraint

$$\varphi = \tilde{Q}\,\phi$$

# Rotation

- We make use of the quaternion representation of a rotation *α* about an axis **u**:

$$\mathbf{q} = \cos\frac{\alpha}{2} + \mathbf{u}\sin\frac{\alpha}{2}$$

- In our case, a rotation about $\phi \times \varphi$ by $\alpha = \cos^{-1}(\phi \circ \varphi) \in [0;\pi]$ rotates *Φ* to align with *φ*

# Rotation

- From this, we can define:

$$\tilde{Q} = \tilde{Q}_2 \tilde{Q}_1$$

Where $\tilde{Q}_1$ is given by the matrix form of the quaternion with an angle and axis defined by the circle and ellipse parameters, and $\tilde{Q}_2$ a rotation about φ by an arbitrary angle β

# Experimental Results

# Using Both Wheels

- We are able to resolve this arbitrary rotation, and improve the scale factor calculation, by making use of both wheels in an image
- We can find a 3D vector Δ between the front and rear wheels in the 3d model, and a vector δ between the wheels in the image.
- The z-component of δ can be estimated by assuming the line between wheels is orthogonal to the axle

# Using Both Wheels

- We can fix the previously arbitrary value β using the constraints:

$$\cos\beta = \frac{\delta \circ \Delta}{\|\delta\|\|\Delta\|}, \sin\beta = \frac{\det(\delta, \Delta, \varphi)}{\|\delta\|\|\Delta\|}$$

- We can also improve the scale factor by setting:

$$\sigma = \frac{\|\delta\|}{\|\Delta\|}$$

# Using Both Wheels

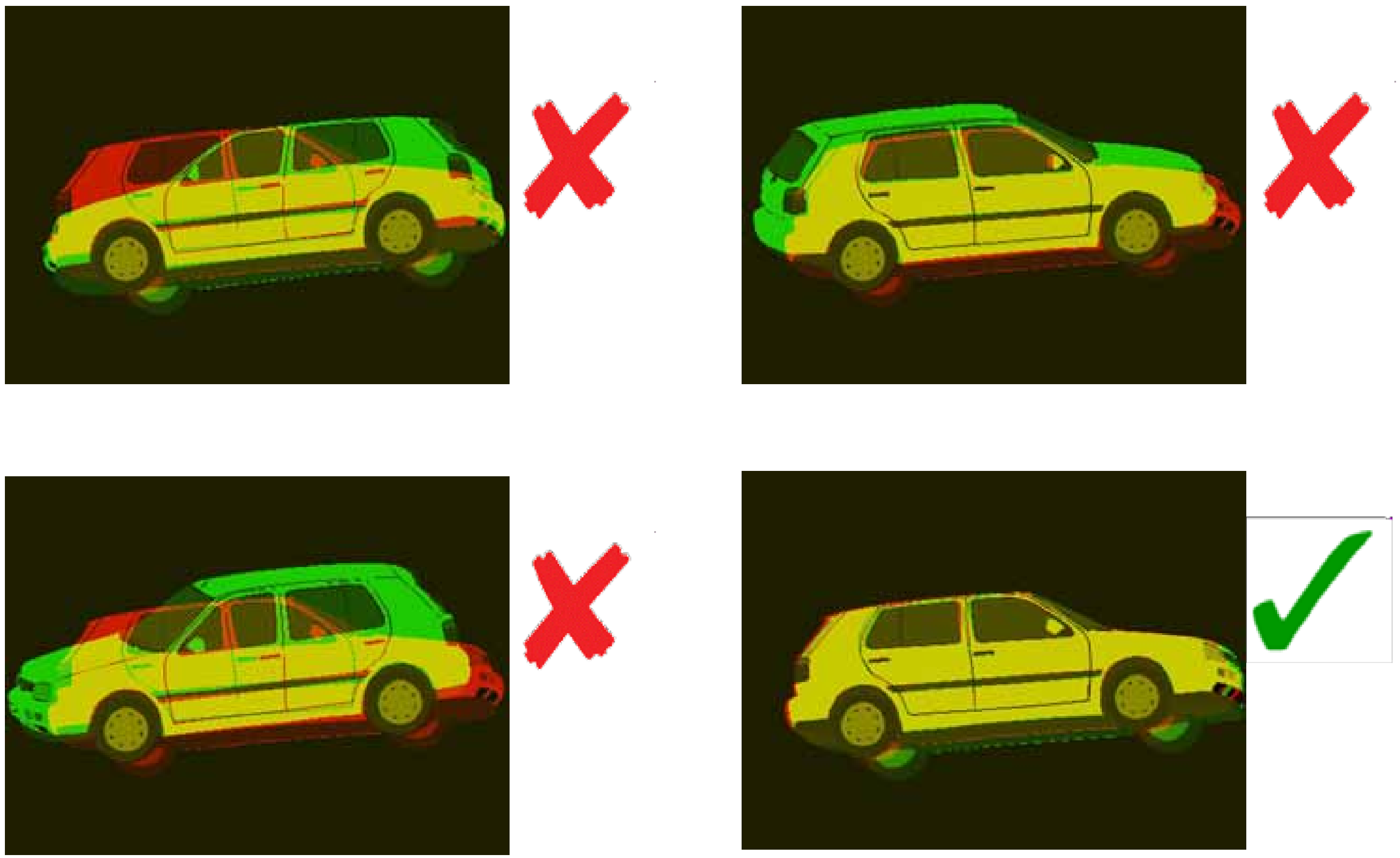
# Ambiguities

- There are, unfortunately, some ambiguities that cannot be resolved explicitly in this framework:

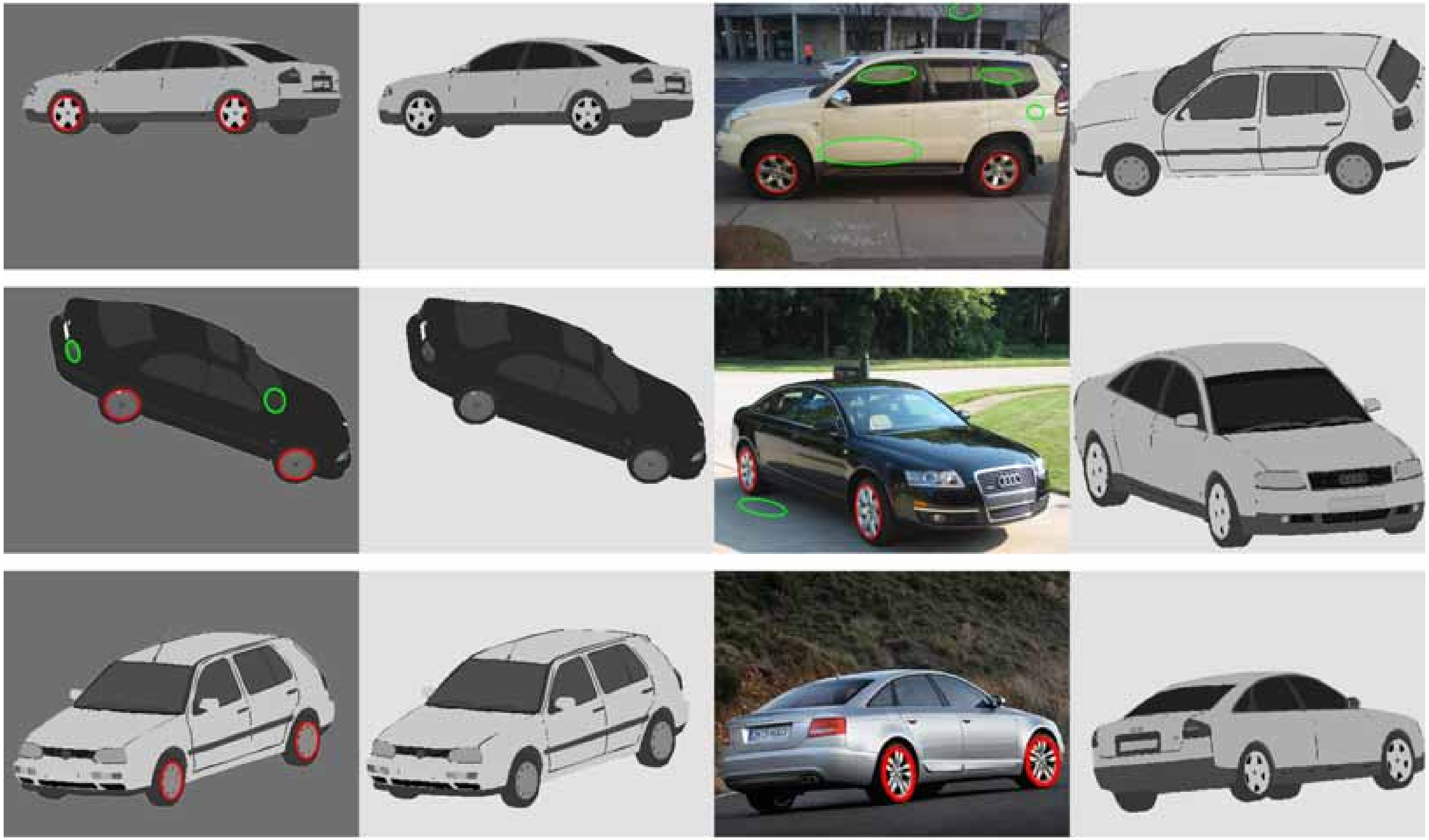
We don’t know which way the car is facing in the image

We don’t know if the car is pointing into or out of the image

# Ambiguities



# Experimental Results



# Questions?