Machine vision object tracking for automatic metal sheet cutting

Matouš Cejnek*[1], Cyril Oswald[1]

[1] *ČVUT v Praze, Fakulta strojní, Ústav přístrojové a řídící techniky, Technická 4, 166 07 Praha 6, Česká republika*

Abstract

Automation of metal sheet cutting increase the effectivity of the metal sheet manufacturing process. The goal of this project is object tracking algorithm that leads to optimalization of the metal sheet cutting process. The suggested object detection and tracking method is composed from multiple machine learning state of the art algorithms. The obtained results are close to the expert human operator guidelines.

Key-words: machine vision; evolutionary algorithms; metal sheet manufacturing

1. Introduction

The automation of manufacturing processes is huge trend nowadays. Such an automation might save time, money and reduce human error. The goal of this work is to provide machine vision [1] assistance for automated metal sheet cutting. The software we develop should be later used by our industry partner. The ultimate goal of this study is to use camera to measure the exact position where the metal sheet should be cut. The cut of the metal should remove the end of the metal sheet, because the end has irregular shape and quality. To find out the optimal position to cut in an automated way requires the following information:

- location of the very tip of the metal sheet
- position where the metal sheet end arc has a connection with sides of the metal sheet
- optionally also the full contour of the metal sheet object could help

The target positions that should be measured from image are shown in Fig. 1.

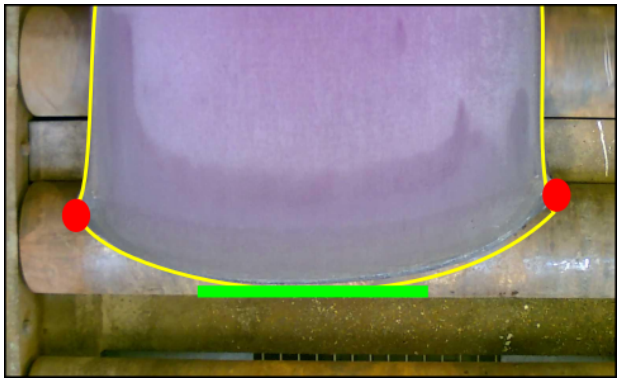

Fig. 1. *The target positions that should be measured from image: green - tip of the sheet, red - connection of arc and sides, yellow - full contour*

2. Methodology

The methods and approaches used to accomplish the goal of this project are described in this section. The proposed approach can be broken into following steps:

1. step: *pre-processing* (subsection 2.2.)
2. step: search for approximate position (subsection 2.3.)
3. step: search for points on contour (subsection 2.4.)
4. step: *post-processing* (subsection 2.5.)
5. step: model adaptation (subsection 2.7.)

2.1. Programming language and tools

Python 3 [2] was used for development of whole software package for machine vision. This language was chosen because it has direct binding to OpenCV [3]. The OpenCV is one of the most popular tools to simplify machine vision tasks. It is open source project. Most of the OpenCV is running natively in *C* language. This is important because the final implementation of the machine vision software on the target industry machine should be done in language *C#*, Because of that reason, the re-implementation of the Python machine vision software in *C#* language should be relatively easy. The reason why the *C#* language was not used also for development is the coding difficulty of this language. In Python we can develop and test various solutions in much faster pace. Other Python libraries we used: SciPy [4] and Numpy [5].

2.2. Image pre-processing

The most of the commonly used approaches for image pre-processing is unavailable during this task because of the insufficient testing materials and incomplete information about target hardware and light conditions. Optimisation of the image pre-processing at this point would be way too premature. Because of these reasons we need to deal with images in the form we get them and try to overcome their differences without bigger processing.

Thus the whole pre-processing consist only from the filtration with *Sobel filter* [6]. This approach throws away big portion of the case-specific information (colour etc.) and somehow unify all the images. The output of the *Sobel filter* are two images - one is

*Kontakt na autora: matous.cejnek@fs.cvut.cz

for the horizontal differences in image, and the other one is for vertical differences in the image. The *Sobel filter* is convolution filter. This convolution can be written as:

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \tag{1}$$

and

$$\mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A} \tag{2}$$

where $\mathbf{A}$ is the source image to be filtered. These two resulting images (and image made from combination of the both outputs of *Sobel filter* by averaging) are used as input for following steps. The example combined image is shown in Fig. 2.

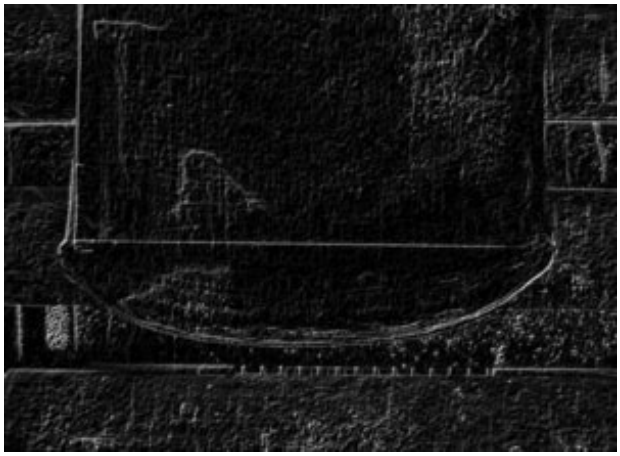

Fig. 2. *Output of the Sobel filter - combination of filtering in horizontal and vertical axis.*

2.3. Search for approximate position

After pre-processing with *Sobel filter* is search for approximate position - boundary box - of the metal sheet in the image. The approximate position is described by four values. Two values are on horizontal axis - the vertical lines locating the sides of the metal sheet. Another two values are distances on vertical axis - horizontal lines locating the end of the metal sheet. Schematics displaying mentioned lines is in Fig. 3.

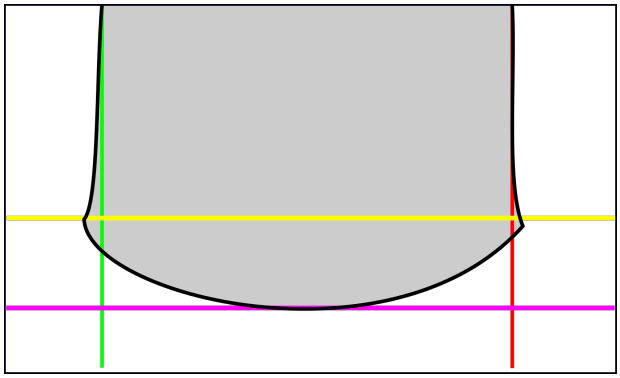

Fig. 3. *Approximate position of metal sheet described by four lines - first step of the metal sheet detection and measurement.*

The location of the lines is found by template matching. Simple pseudo-binary templates (only values in template are 0 and 255) are used. The templates used for detection of metal sheet sides (vertical lines) is create artificially by repeating sample row vector representing the sharp change in intensity in the image. The resulting template for the detection of the left side can be written as:

$$\mathbf{T} = \begin{bmatrix} 255 & 255 & \dots & 0 & 0 \\ 255 & 255 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 255 & 255 & \dots & 0 & 0 \\ 255 & 255 & \dots & 0 & 0 \end{bmatrix} \tag{3}$$

The template for the right side is obtain by horizontal flip of the left side template. These templates are moved along the horizontal axis and the error against the background is measured. The location with lowest error is selected as the resulting position of the particular line (left or right side). The height of the searched horizontal slice of the image (and thus the height of the template) is chosen as the first third of the image. This setup is display in Fig. 4.

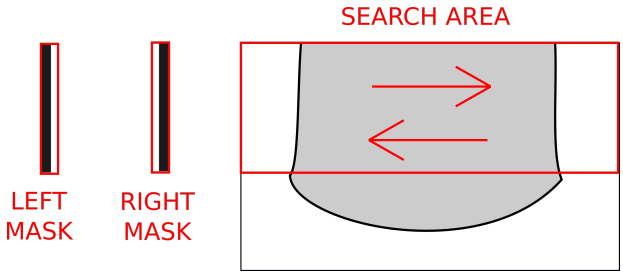

Fig. 4. *The red area over the metal sheet schematics describe where the patterns are searched. The patterns (masks) are on the left side of the Figure.*

The location of the horizontal lines (location of the metal sheet end) is also find with template matching. The used template was created manually according to the reference image. This template is stored as an image and is resized to fit the distance between left and right side of the metal sheet (vertical lines). This setup is display in Fig. 5.

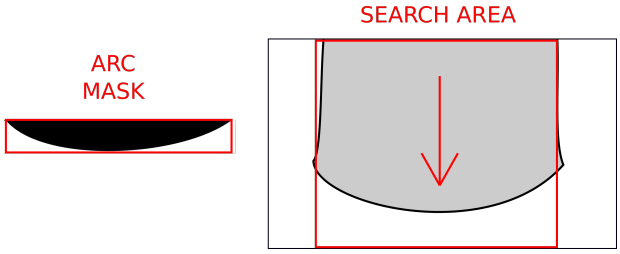

Fig. 5. *The red area over the metal sheet schematics describe where the pattern is searched. The pattern (mask) in on the left side of the Figure.*

The output of this step is four lines (locations), describing the approximate position of the metal sheet in the image. This output is used for the next steps, because they require the approximate position of the metal sheet.

2.4. Search for points on contour

Now, when the approximate position of the metal sheet is known, it is possible to use this information for more targeted search. During this second step smaller templates are used to find out the points, that are probably laying on the contour of the metal sheet. This approach is similar like the one in the previous step. The difference is, that in this step the target area is search line by line (row by row in case of sides and column by column in case of metal sheet end). This process is more computationally expensive, however it is done only in the surrounding of the main lines. The template used in this step is similar to template 3, however is much smaller (5x5). Again the template is flipped and/or rotated according required direction of use. This result is display in Fig. 6.

Fig. 6. *The output of the search for points that are probably laying on the metal sheet contour.*

This step was not possible without approximate location of the metal sheet boundary from the previous step.

2.5. Combination of previous outputs and post-processing

In this step the previous results are summarised. Also some post-processing operations are applied to reduce noise and highlight the probably contour. The combination of the *Sobel filter* output 2 and image with suspicious points 6 is obtained by averaging of these two images. The result is after that processed with two rounds of erosion and dilatation. Because the resulting image is rather dark, the lighter pixels are emphasised via simple threshold criteria. The final outcome of this post-processing is displayed in Fig. 7.

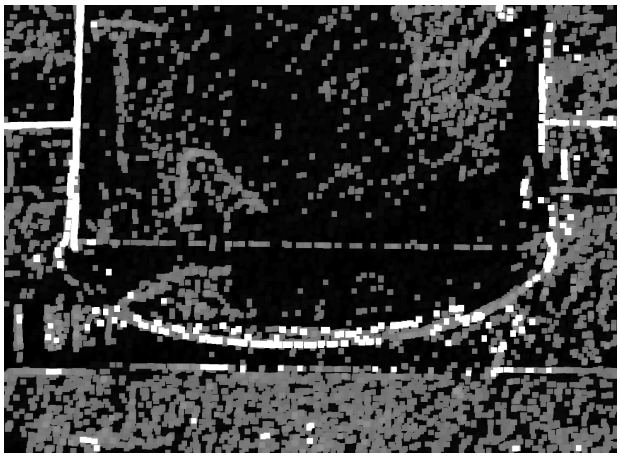

Fig. 7. *The outcome of previous results merge and post-processing.*

2.6. Metal sheet contour model

The model used for describing the metal sheet contour is described in this subsection. According to the shape complexity and limited information that we can acquire from the images we define simple model consisting from four *Bezier curves* and two short lines. The short lines are introduced to bridge the complex connection between the end arc with the metal sheet sides. All control points of any used curve are enforced into the boundary rectangle of the curve. The curve end points are in the most far corners (diagonal) of the rectangle. The middle control points cannot escape this rectangle. In fact, these points are given by two float numbers (in range 0-1) describing their position in the rectangle. This limitation was introduced, because a correctly adapted model should never need to exceed this boundary.

The raw schematics of the model is shown in Fig. 8.

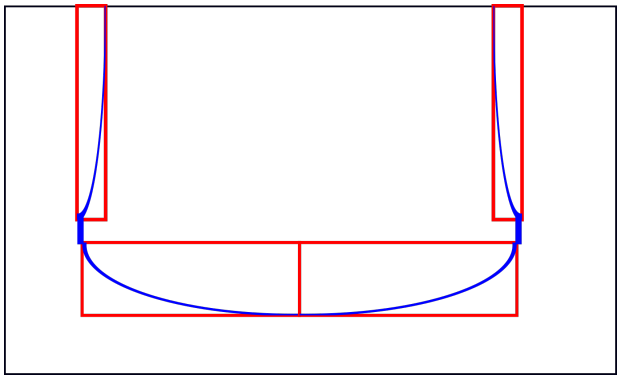

Fig. 8. *Default naive model of metal sheet contour. The model contour is in blue, boundary boxes of Bezier curves are in red.*

This model has multiple adaptable parameters: distance parameters (9 in total), displayed in Fig. 9; and control points of *Bezier curves* (2 points (with 2 dimensions - horizontal, vertical) per curve, 16 values in total).

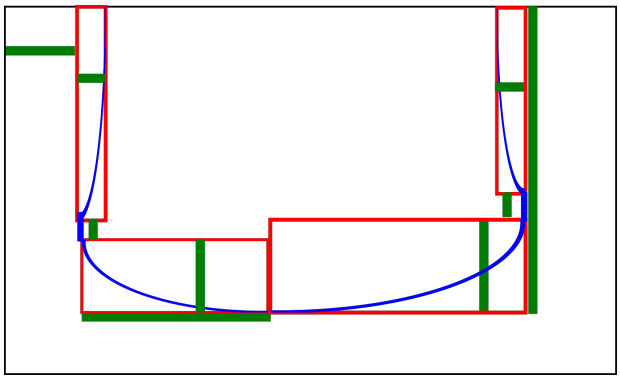

Fig. 9. *Adaptable distance parameters of the model. They are in green; the model contour is in blue, boundary boxes of Bezier curves are in red.*

Example schematics of poorly adapted model is shown in Fig. 10.

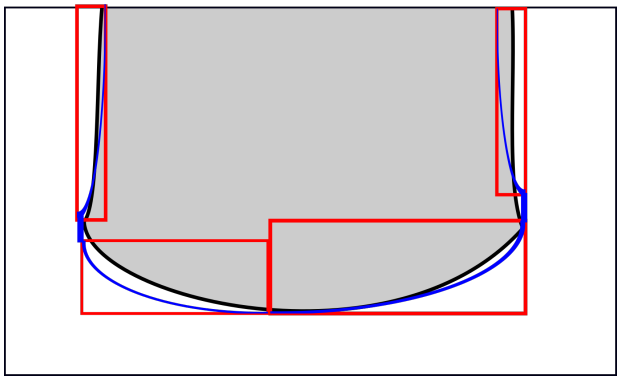

Fig. 10. *Demonstration of the model degrees of freedom - poorly adapted model. The model contour is in blue, boundary boxes of Bezier curves are in red.*

2.7. Model adaptation

The last step of the process is adaptation of the model - its adaptable parameters - to match the contour of the metal sheet in the given picture. An *evolutionary algorithm* [7] was designed for this purpose because the solution space is way to wide. The absolute difference between model picture representation and the outcome from the third step (last output after pre-processing) was used as the *cost function*. Every iteration of the *evolutionary algorithm* contain few actions:

- *individual* from the current *generation* are ordered from the most fit to the least fit *individual*
- some number of best *individuals* are added to the next *generation*
- some number of best *individuals* are slightly mutated and added to the next *generation*
- some number of best *individuals* are strongly mutated and added to the next *generation*
- the next *generation* is completed by few new randomly generated *individual*

Because of such an algorithm can lead to the state where all the best *individuals*, who advance to the next *generation* are *siblings*. To prevent this, every individual has a universally unique identifier (UUID). The *individuals* created by mutation get their UUID from their origin. Every *individual* that has more fit siblings is penalised during cost function evaluation. This concept enforce diversity among the best *individuals*. As a criteria to stop evolution can be used the difference between the best *individual* and the average of the best *individuals*. If this difference drops close to zero it is highly unlikely that better solution will be found.

3. Testing and results

Few testing images were obtained from the industry partner. The results are very much similar. Two the most distinct images were selected for publishing in this publication. We would like to highlight the difference between the images - different colour settings of camera, different position of the metal sheet on the background.

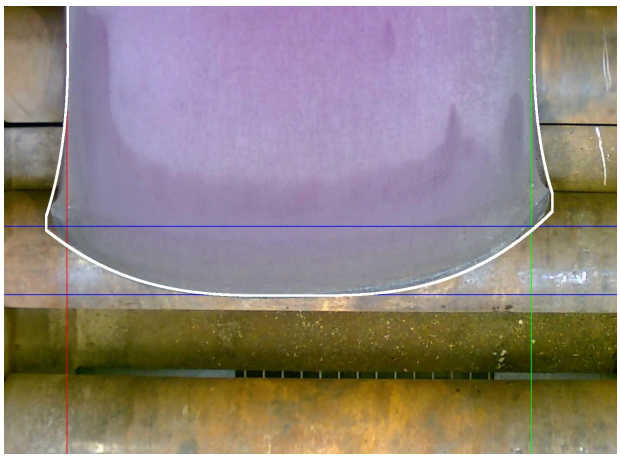

Fig. 11. *Contour of adapted model for testing result #1. Contour is in white colour.*

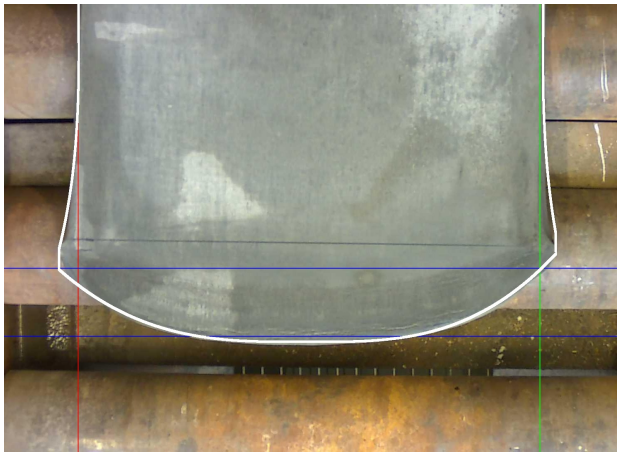

Fig. 12. *Contour of adapted model for testing result #2. Contour is in white colour.*

4. Conclusion

In this paper we present our solution for the problem of metal sheet object position detection on conveyor belt via means of machine vision. The results we achieved on the testing images are promising. The

solution we propose is complex, however it is easily implementable on the target machine in the industry, because most of the hard work is done by standard libraries ported to *C#* language.

Acknowledgement

This work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. *SGS18/177/OHK2/3T/12*. Authors would like to acknowledge our industry partner PT SOLUTIONS WORLDWIDE spol. s r.o.

Nomenclature

$UUID$	universally unique identifier (−)
$\mathbf{T}$	Template (−)
$\mathbf{G}_x$	Kernel of horizontal Sobel filter (−)
$\mathbf{G}_y$	Kernel of vertical Sobel filter (−)
$\mathbf{A}$	Source image (−)

References

[1] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image processing, analysis, and machine vision*. Cengage Learning, 2014.

[2] Guido Van Rossum and Fred L Drake Jr. *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[3] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[4] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. [Online; accessed]. 2001–. Available at: http://www.scipy.org/.

[5] Travis E Oliphant. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA, 2006.

[6] Nick Kanopoulos, Nagesh Vasanthavada, and Robert L Baker. "Design of an image edge detection filter using the Sobel operator". In: *IEEE Journal of solid-state circuits* 23.2 (1988), pp. 358–367.

[7] Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.